MS 64E

C-842 API

Application Programming Interface
Windows 95/98
Windows NT/2000

Function Reference
&
Programming Instructions

This Document is valid for the Products:

C-842.xx DC-Motor Controller

Release: 4.10
Release Date: 20 December 2000

© PI (Physik Instrumente) **GmbH & Co.**
D-76337 Waldbronn, Germany FAX: (+49)7243-604-145
EMail:info@physikinstrumente.com http://www.physikinstrumente.com

Table of Contents :

0. Manufacturer Declarations

The contents of the manual is furnished for informational use only, is subject to change without notice and should not be construed as a commitment by Physik Instrumente (PI).

Physik Instrumente (PI) assumes no responsibility or liability for any errors or inaccuracies that may appear in this manual.

0.1.1. Disclaimers

The software described in this manual is distributed as it is.

Physik Instrumente (PI) expressly disclaims any and all warranties including any implied warranty of merchantability and fitness for a particular purpose. Physik Instrumente (PI) does not warrant, guarantee, or make any obligations regarding the use or the results of the use of this software, in terms of correctness, accuracy, reliability or otherwise, and does not warrant that operation of the software will be uninterrupted or error-free.

© Copyright 2000 by Physik Instrumente GmbH

Release: 4.10
File:MS64E410.doc, 129536 Bytes
PDate: 20.12.00 12:13

1. C-842 Library Support

The functionality of the C-842 motor controller is available for windows programmers by *Dynamic Link Libraries* offering a variety of functions for board handling and motion control.

Since the C-842 controller card is an ISA bus card, programmers have to deal with the fact, that Windows NT does not offer direct access to hardware components. So different approaches are required for both developing platforms:

1.1.1. Windows 95/98

The dynamic link library offers functions accessing directly the I/O port space of the ISA bus. No special driver is required and the library can be used from all 32 bit developing environments like VC++, Delphi, VB and others.

1.1.2. Windows NT

Windows NT does not allow direct hardware access. for this reason a NT kernel null driver is available that has to installed in the NT system prior using the NT-DLL. Unlike the Windows 95 DLL this library calls the driver functions in order to access the C-842 card. The functionality is almost the same but functions may differ in details depending on the evolution level of the file.

2. Library for Windows 95/98

QFLW400.DLL

Version 4.00, January 2000

32-bit DLL for C-842 DC-Motor Controllers

Frequently used commands like *Move Absolute*, *Move Relative* or position queries like *Tell Position* can be called by library functions. Other commads like *Set P-gain* or *Tell following error* can be accessed via the TRANSLATE and/or EXECUTE function. The actual command then is passed as a parameter to the library function.

More than 100 commands are defined as constants to be accessed with the EXECUTE function. Packaging the command in a string in conjunction with axis identifier and parameter is the other way to send a command via the TRANSLATE function.

Examples: execute(3,MR,45000,report);

translate("3MR45000",report);

Both functions perform the same action, moving motor#3 relative for 45000 counts.

In the same manner, the complete QMove commands set of more than 100 commands can be called.

The command transfered is executed immediately when received.

For detailed description on the commands implemented refer to
***Programming Manuals MS45E and MS46E**.*

2.1. QWLF400.DLL Function Overview:

Ordinal	Entry Point	Name
0045	00000000	AutoFindEdge
0011	0001f548	Axis_Installed
0017	00000000	Board_Installed
0015	0001f58c	InitAxis
0013	000234f8	InitBoard
001f	0001f5ac	Init_LS
0019	0001f070	MoveA
0031	0001f5cc	MoveA_12
0032	00000000	MoveA_123
001b	0001fb60	MoveR
0033	00000000	MoveR_12
0034	00021c04	MoveR_123
000f	00000000	Set_BaseAddress
0021	00023440	StatusBoard
003b	00000000	VectorA
003c	0001f624	VectorR
001d	00000000	WaitStop
0029	0001f658	autodetect
0023	00000000	axis_select
0009	00021a5c	execute
0027	00000000	getQMC
0004	00021af8	get_ErrorStatus
0003	00000000	get_PosErr
0006	000233f4	get_RefSignal
0001	00000000	get_pos
0007	0001f7a4	get_pos4
0005	00000000	get_stat
002d	0001f8fc	get_v
000d	00000000	moving
0025	0001faac	setQMC
002b	00000000	set_pos
000b	00021b50	translate

2.2. Types, Constants and Structures

Types, constants and functions are defined in a definition text files that can be used directly by Pascal and Delphi programmers:

QFLW400.DEF

Types used by the QFL library:

```
Tstr = string[5];

   Tcmd = record
             code   : byte;
             qclass : byte;
             lab    : Tstr;
          end;

   TQ_long    = array[1..4] of longint;
   TQ_integer = array[1..4] of integer;
```

All commands (cmd) used with the **execute** procedure are defines as constants of the type TCmd. Names of the variables comply with command names used in

the QMove or WinMove program and are documented in the MS 45E operating manual.

Example of a command constanf of TCmd type, here SA(Set Acceleration)

SA : Tcmd = (code:$12; qclass:wr_U1616; lab:'SA:'); (* SET_ACC *)

In the same manner more than 100 QFL-commands are defined. For further information see the definition QFLW400.DEF file.

General structure of a program using the QFLW library:

```
Set_BaseAddress($214)          Define board address (optional if not default address hex210)
MaxAxis := Axis_Installed      Detects max axis number and adjusts sample rate to 400 µs
status := InitBoard(9)         Sets the board function register
status := autodetect()         Reads and defines the limit switch configuration
InitAxis(1)                    Sets Motor #1 to default parameters
InitAxis(2)                    Sets Motor #2 to default parameters

translate('1SV80000',report)   main program
translate('2SV120000',report)
translate('1SA250',report)
translate('2SA190',report)
translate('1DP120',report)
translate('2DP250',report)
translate('1DI12',report)
translate('2DI12',report)
translate('1DD230',report)
translate('1DD350',report)
MoveA(1,5000)   //move command
MoveA(2,5000)   //move command
...
...
end.
```

2.2.1. For Delphi Programmers

Declaration of functions: (Pascal style)

```
(*************************************************************************)
(* Function Declarations  :
(*************************************************************************)
// Case sensitive writing required !!
// ExtLib = DLL_FileName
function  get_pos(axis:byte):longint; stdcall external ExtLib; //2
function  get_PosErr(axis:byte):longint;  stdcall external ExtLib; //4
function  get_ErrorStatus:integer; stdcall external ExtLib; //5
function  get_stat(axis:byte):integer;stdcall external ExtLib; //6
function  get_RefSignal(axis:byte):word; stdcall external ExtLib; //7
procedure execute(axis:byte;command:TCmd;para:longint;report:Pchar);
                  stdcall external ExtLib; //10
procedure translate(cmd:Pchar;rep: Pchar); stdcall external ExtLib; //12
```

```
function  moving(axis:byte):boolean; stdcall external ExtLib; //14
procedure Set_BaseAddress(address:word); stdcall external ExtLib; //16
function  Axis_Installed:byte; stdcall external ExtLib; //18
function  InitBoard(mode:byte):integer; stdcall external ExtLib; //20
procedure InitAxis(axis:byte); stdcall external ExtLib; //22
function  Board_Installed:boolean; stdcall external ExtLib; //24
procedure MoveA(axis:byte; position:longint); stdcall external ExtLib; //26
procedure MoveR(axis:byte; counts:longint); stdcall external ExtLib; // 28
function  WaitStop(axis:byte):integer; stdcall external ExtLib; //30
procedure    Init_LS(drive1,drive2,drive3,drive4:byte);    stdcall    external
ExtLib;//32
function  StatusBoard:word; stdcall external ExtLib; //34
procedure axis_select(axis:byte); stdcall external ExtLib; //36
procedure setQMC(cmd:byte; para:longint); stdcall external ExtLib; //38
function  getQMC(cmd:byte):longint; stdcall external ExtLib; //40
function  autodetect:integer; stdcall external ExtLib; //42
procedure set_pos(axis:byte; newpos:longint);stdcall external ExtLib; //44
function  get_v(axis:byte):longint;stdcall external ExtLib; //46
procedure VectorA(axis_1, axis_2:byte; pos_1, pos_2, v_path:integer);
          stdcall external ExtLib; //60
procedure VectorR(axis_1,axis_2:byte; move_1,move_2,v_path:integer);
          stdcall external ExtLib; //61
procedure AutoFindEdge(axis:byte; mode:byte); stdcall external ExtLib; //70
```

2.2.2. For MSVC++ Programmers

Programming examples using qflwrun.cpp, qflwrun.H, qfl400_VC.LIB

Declaration of functions: (C style)

```
long get_pos(char axis);
long get_PosErr(BYTE axis);
long get_v(BYTE axis);
int  get_ErrorStatus(void);
int  get_stat(BYTE axis);
void get_pos4(TQ_long *pos4);
void execute(BYTE axis,TCmd command,long para,;char *report);
void translate(char *cmd, char *rep);
BOOL moving(BYTE axis);
void Set_BaseAddress(unsigned short address);
BYTE Axis_Installed(void);
int InitBoard(BYTE mode);
void InitAxis(BYTE axis);
BOOL Board_Installed(void);
void MoveA(BYTE axis, long position);
void MoveA_12(long pos1, long pos_2);
void MoveA_123(long pos1, long pos_2, long pos_3);
void VectorA(BYTE axis1, BYTE axis2, long position1, long position2, long vel)
void MoveR(BYTE axis, long counts);
void MoveR_12(long shift_1, long shift_2);
void MoveR_123(long shift_1, long shift_2, long shift_3);
void VectorR(BYTE axis1, BYTE axis2, long shift1, long shift2, long vel)
int WaitStop(BYTE axis);
void Init_LS(BYTE drive1,BYTE drive2,BYTE drive3,BYTE drive4);
unsigned short StatusBoard(void);
void axis_select(BYTE axis);
void setQMC(BYTE cmd,long para);
long getQMC(BYTE cmd);
int autodetect(void);
void set_pos(BYTE axis, long newpos);
void AutoFindEdge(BYTE axis, BYTE mode);
```

2.3. QFLW400 Library Function Reference :

Function reference uses C syntax.

Definitions:

BYTE represents unsigned char

WORD represents unsigned short

2.3.1. General Functions calling Sub Functions

void **execute**(BYTE axis, Tcmd SubCommand, long parameter, char *report)

This function calls one of the more than 100 sub-functions of the QFL library interface. The sub command of the type TCmd has a structure defined in the header or in the Pascal definition file. The *report parameter points to a 128 bit character array to hold a PChar type.

Ordinal index: 10

Input: *axis* as char (byte)
command as Tcmd
parameter as long
report as pointer

Return: none

void **translate**(char *command, char *report)

This function can be used instead of the execute function if it is more convenient to work with strings.

Ordinal index: 12

input: command:
Pointer to character array (PChar, zero terminated string), holds command string including axis identifier, command and parameter. Format example "1MR5000" with 1(motor identifier), MR(command) and 5000(parameter).
report:
Pointer to character array (PChar), supplied by the calling function. The calling program should define a character array of 256 characters to take the report message.

2.3.2. System Initializing Functions

void **Set_BaseAddress**(unsigned short address)

This procedure allows to define the hardware I/O address selected by the DIP switches on the C-842 board. Default setting is 0x210.

If the C-842 board is used with the default address setting (0x210), this function is not required.

Input: address as unsigned short (word, 16 bit)

Return: none

BOOL **Board_Installed**(void)

This function returns a boolean value depending on a card is installed or not. The function looks for the board at the address defined by Set_BaseAddress function.

Ordinal index: 24

Input: none

Return: boolean (0 or 1), 1=installed, 0=not installed

BYTE **Axis_Installed**(void)

This function returns the number of motor axes supported by the processor. There are 2-axes and 4-axes version available (C-842.20 and C-842.40).

Ordinal index: 18

Input : none

Return: number of axes as BYTE

int **InitBoard**(BYTE mode)

This function initializes the C-842 and returns the board status. It has to be sent after the C-842 is powered up to enable the amplifiers and to define the position encoder functions. The mode value is transferred to the board function register. For normal operation set mode to 9.

Following actions are performed:

- Set board function register to mode value
- Board reset
- Set output to DAC16
- Set limit sense register to 0x0FF

Ordinal index: 20
Input: mode as BYTE
Return: status as int

void **Init_LS** (BYTE axis1, BYTE axis2, BYTE axis3, BYTE axis4)

This function sets the Limit_Sense Register.

After the system is powered up, the limit switch wiring and logical levels have to be defined. If no limit switches are connected, axis parameters have to be set to 0 for the corresponding axis.

If limit switches are wired compatible with PI standard stages, the corresponding Byte has to be set to 1.

Example: One PI-compatible stage is connected to axis #2. All other axes are not connected or have motors without limit switch connection, call Init_LS(0,1,0,0)

Note: Init_LS() overwrites autodetect settings if called afterwards.

Ordinal index: 32

Input: mode as BYTE

Return: status as int

int **autodetect**(void)

This function reads the current limit switch lines levels of all axes and defines the reading as normal state (no limit switch tripped). Any change of that status is interpreted as a limit switch hit event, triggered by the stage.

Note: When calling autodetect, no limit switch must be hit.
Note: autodetect overwrites prior Init_LS settings.

Actions performed:

reads the current limit switch state

analyses whether one limit switch is hit and returning error code in that case

sets the limit_sense state of the board according to the values found

sets LimitMode variable to limit_sense value

Ordinal index: 42
Input: none
Return: value of detected LS configuration (e.g. 255 if no LS connected)
-1 if error (LS was hit)

void **InitAxis**(BYTE axis)

This function initializes the specified axis and sets default values. The settings may not match your specific application, so it might be required to set individual servo parameters, velocities and accelerations.

Actions performed: SET_KP, Set_VEL, Set_ACC, AUTO_STOP_OFF, Upate_Filter, CLR_STATUS

Ordinal index: 22
Input: axis as BYTE
Return: none

void **set_pos**(BYTE axis, long newpos)

This function redefines the current position.

Ordinal index: 44
Input: axis BYTE,
newpos as long
Return: none

void **AutoFindEdge**(BYTE axis, BYTE mode)

This function can be used with stages having automatic reference heading option, e.g. M-500 or M-400 series. After this function is called, the stage starts in the right direction towards the origin point.

Actions performed: After analyzing which side the stage is located, the FEP or FEN commands are called.

Ordinal index: 70
Input: axis as BYTE,
mode as BYTE
Return: none

2.3.3. Moving Functions

void **MoveA**(BYTE axis, long position)

Move Absolute: This function starts one or all motors to move to an absolute position. If the axis identifier is 0, all axes available are moved to the same position.

After the function is called, the motion is started and the function is left immediately. If the next call of MoveA occurs before the motor(s) have terminated their moves, the old target is replaced by the new one and the motors immediately head towards their new targets, even if the previous target(s) were not yet reached. In order to synchronize motions and to start the next movement not before the first target is reached, the WaitStop command has to be called before the next move command is executed.

Example 1 :At first, motor#1 will run to position 5000 and stop there, then run to zero position:

MoveA(1,5000)
WaitStop(1)
MoveA(1,0)

Example 2: Assume the motor is at zero position. Without the WaitStop command the first move to 85000 will not be performed and the motor will already stop at position 1000:

MoveA(1,85000)
MoveA(1,1000)

Ordinal index: 26
Input: axis as BYTE, position as long
Return: none

void **MoveA_12**(long position1, long position2)

Move Absolute XY with synchronized start. This function starts axis 1 and 2 with individual velocities at the same time.

After the function is called, the 2-axis motion is started and the function is left immediately. The general behavior is similar like the MoveA function.

Example : Set velocities of axes 1 and 2 and start simultaneous motion:

translate("1SV20000",report)
translate("2SV80000",report)
MoveA_12(25000,75000)

Ordinal index: 50
Input: position1 as long, position 2 as long
Return: none

void **MoveA_123**(long position1, long position2, long position3)

Description same as MoveA_12()

void **MoveR**(BYTE axis, long counts)

This function performs a relative move of a given number of steps (counts). The specified count number is added to the current position and the result is commanded as the new target.

After the function called, the motion is started and the function is left immediately. If the next call of MoveR occurs before the motor(s) have terminated their moves, a continuous motion to the new target is performed. In order to synchronize motions and to start the next movement not before the last move is terminated, the WaitStop command has to be used.

If the axis identifier is 0, all axis available are moved by the same number of counts.

Ordinal index: 28
Input: axis as BYTE, counts as long
Return: none

void **MoveR_12**(long position1, long position2)

Move Relative XY with synchronized start. This function starts axis 1 and 2 with individual velocities at the same time.

After the function is called, the 2-axis motion is started and the function is left immediately. The general behavior is similar like the MoveR function.

Example : Set velocities of axes 1 and 2 and start simultaneous motion:

```
translate("1SV20000",report)
translate("2SV80000",report)
MoveR_12(25000,75000)
```

Ordinal index: 52
Input: position1 as long, position 2 as long
Return: none

void **MoveR_123**(long position1, long position2, long position3)

Description same as MoveR_12()

void **VectorA**(BYTE axis1, BYTE axis2, long position1, long position2, long vel)

This functions performs a linear interpolation and starts axis1 and axis2 at the same time towards the coordinate point (position1/position2) with the path velocity vel.

Example : Vector move of axes #1 and #2 from the current position to the absolute point 12000/30000 with the path velocity of 6500 counts/s:

```
VectorA(1,2,12000,30000,6500)
```

Ordinal index: 60

Input: axes as BYTE, positions and velocity as long
Return: none

void **VectorR**(BYTE axis1, BYTE axis2, long shift1, long shift2, long vel)

This functions performs a linear interpolation and starts axis1 and axis2 at the same point in time for a vector with the components shift1/shift2 with the path velocity vel.

Example : Relative Vector move of axes #3 and #4 relative from the current position for 15000/19500 counts with the path velocity of 26500 counts/s:

VectorR(3,4,15000,19500,26000)

Ordinal index: 61
Input: axes as BYTE, shifts and velocity as long
Return: none

long **WaitStop**(BYTE axis)

The WaitStop function controls the dynamic position of the specified axis and returns if the motor has reached the target. It can be used to synchronize multiple movements.

Ordinal index: 30
Parameter: axis as BYTE
Return: none

BOOL **moving**(BYTE axis)

This function returns true if the trajectory of the specified axis has not yet reached the target position. The function reads the on-target flag of the condition register.

Ordinal index: 14
Parameter: axis as BYTE
Return: boolean

2.3.4. Reading Functions

long **get_pos**(BYTE axis)

This function reads the current position of the indicated motor.

Ordinal index: 2
Parameter: axis as BYTE
Return: Current encoder position as long

long **get_PosErr**(BYTE axis)

This function returns the current deviation of the encoder position from the profile position in encoder counts.

Ordinal index: 4

Parameter: axis as BYTE
Return: error counts as *long*

long **get_v**(BYTE axis)

This function returns the current velocity of the defined axis and returns it in counts/s.

Ordinal index: 46
Input: axis as BYTE
Return: velocity as long

int **get_stat**(BYTE axis)

This function returns the status word of the indicated axis.

Ordinal index: 6
Input: axis as BYTE
Return: status value as integer

void **get_pos4**(TQ_long *Pos4)

This function reads the current posiltions of all 4 axes and writes it into an array of 4 long integers.

Ordinal index: 8
Input : array[1..4] of long as TQ_long
Return: none

WORD **StatusBoard**(void)

This function reads the board status register and returns it as unsigned integer.

Ordinal index: 33
Input : none
Return: board status register as WORD (unsigned integer)

WORD **get_RefSignal**(BYTE axis)

This function reads the current reference signal line level and returns 0 if the line is low or 1 if the line is high.

Ordinal index: 6
Input : axis as BYTE
Return: 0 as WORD (unsigned integer) if line is low
1 as WORD (unsigned integer) if line is high

int **get_ErrorStatus**(void)

This function reads error status number.

```
  0 : no error
  1 : setQMC : chipset
 10 : WaitQMC timeout
 11 : setQMC cmd not found
```

```
 12 :        timeout
 13 : getQMC cmd not found
101 : translate: command not found
230 : FEP/FEN limit hit
231 :         user break
232 :         wrong axis
```

Ordinal index: 4
Input : none
Return: error status number as integer

2.3.5. Low Level Functions

void **axis_select** (BYTE axis)

This function activates the specified axis and updates an internal axis flag. Normal move commands do not need the axis_select function, except setQMC or getQMC are used.

Ordinal Index: 36
Input: axis as BYTE
Return: none

void **setQMC**(BYTE hexcommand, long parameter)

This function writes a genuine command and the related parameter to the activated axis. Use **axis_select** function to choose one axis prior to call setQMC. axis_select is only required if the axis should be changed.

Ordinal index: 38
Input: hexcommand as BYTE
Return: none

long **getQMC**(BYTE hexcommand)

This function returns the questioned value. as long as the query does not concern a global command, the value for the activated axis is returned. Use axis_select function to choose one axis prior to call getQMC. axis_select is only required if the axis should be changed.

Ordinal index: 40
Input : command in hexcode as char (byte)
Return: value as long integer

Special Application :

ClearAxisStatus

In order to clear the status word of one axis, the follwing sequence should be used:

axis_select (axisnumber)
setQMC (0x33,0)

3. Use of C-842 under NT

3.1. How can I work under NT?

Due to restrictions of hardware access, NT systems do not allow direct bus communication with the C-842 card. This is the reason that the Win95 DLL will not word.

Accessing the C-842 card in a NT system, the program or DLL has to access a kernel null driver that enables the actual hardware access to the board. So first step of NT users is to install the C-842 NT driver, supplied on an driver setup disk.

3.2. Installing the C-842 NT Driver

The C-842 NT Application Package allows to run the C-842 DC-Motor Controller with NT operating systems.

The package includes all installation files required to install and to register the driver in the NT environment. The supplied kernel driver interfaces via a DLL with the application program. So programmers who already have used other QFL libraries with C-842 applications may use the same calls with the new 32-bit DLL.

Disk Contents

Name	Größe	Typ	Geändert am
layout.bin	1 KB	BIN Datei	29.07.98 14:35
setup.lid	1 KB	LID Datei	29.07.98 14:35
Data.tag	1 KB	TAG Datei	29.07.98 14:35
data1.cab	129 KB	CAB Datei	29.07.98 14:35
Setup.ini	1 KB	Konfigurati...	29.07.98 14:35
_user1.cab	46 KB	CAB Datei	29.07.98 14:35
_sys1.cab	401 KB	CAB Datei	29.07.98 14:34
setup.ins	56 KB	Internet Co...	29.07.98 14:34
setup.bmp	112 KB	Bitmap	28.07.98 14:01
Setup.exe	59 KB	Anwendung	15.07.97 11:56
_isdel.exe	8 KB	Anwendung	14.07.97 17:37
_setup.dll	11 KB	Programm...	14.07.97 17:35
inst16.ex	275 KB	EX_ Datei	14.07.97 17:29
inst32i.ex	313 KB	EX_ Datei	14.07.97 14:08
lang.dat	5 KB	DAT Datei	30.05.97 12:31
os.dat	1 KB	DAT Datei	06.05.97 15:15

Note: This installation package may copy not the latest version of the DLL. Just add the latest release to you project directory.

This document refers to this library:

Current DLL version: C842NT30.DLL, version 3.0, (December 2000)

For update package, email to info@physikinstrumente.com

3.3. NT-Driver Installation

To install the NT application package in the system, run the SETUP.EXE from the C842 NT installation disk. The setup routine installs the driver "UIO.SYS", modifies the registry and copies die DLL with an example application **DLLTEST** into the destination directory. The project is established with Visual C++ 4.0 and contains all sources required to build the application.

You can not run the installation on Windows NT and Windows 2000 systems.

4. NT-DLL Library (Version 3.00)

FileName: QFLNT30.DLL, Version : 3.00 (December 2000)

The full complexity of the command set of the motion control processor used with the C-842 DC-Motor controller board can be handled easily using the Library functions.

More than 100 commands are supported by the library. The command set includes all commands with direct reference to the processor´s command set and emulates over that many additional commands.

Error Messages:
Error messages are printed into the stream ‘stderr’.

Constants defined:
All commands available like MR, MA, DH,GH, SOP.....(there are more than 100 of these commands) are predefined as constants of the type TCMD. For full command reference see operating manual MS45E.

All commands are defines as variables of the type TCMD. The name of the variable complies with the command names used in the WinMove program and documented in the MS45E operating manual.

4.1. DLL Function List

Ordinal	Entry Point	Name
0000	000010af	Axis_Installed
0001	00001014	Board_Installed
0002	00001023	ClearAxisStatus
0003	00001005	InitAxis
0004	00001019	InitBoard
0005	00001037	Init_LS
0006	00001046	MoveA
0007	00001096	QMCBusy
0008	0000100f	QMCReady
0009	00001032	Set_BaseAddress
000a	000010a0	Set_Board
000b	0000104b	StatusBoard
000c	00001082	autodetect
000d	00001064	axis_select
000e	0000109b	c842_close
000f	0000102d	c842_open
0010	000010aa	execute
0011	00001073	getQMC
0012	00001055	get_BoardStatus
0013	0000101e	get_DLLversion
0014	0000108c	get_RefStatus
0015	00001091	get_pos
0016	00001028	get_pos4
0017	00001087	moving
0018	0000106e	setQMC
0019	00001050	translate

4.2. Building Applications

- Include the header "C842NT30.H" into your application.
- Link or add the import library C842NT30.LIB into your project
- Prior to access the C-842 by the translate function, call the driver initializing function "c842_open(void)". When leaving the application program, close the driver with the function "c842_close(void)". These functions create the connection to the UIO.SYS driver.

4.3. Header File

```
// C842NT30.H
// 2000-12-19
// header for NT-DLL for C-842
//----------------------------
#include <windows.h>
#include "C842CMD26.H"

#ifdef __cplusplus
 extern "C" {
#endif

#ifdef C842_DLL
 #define FUNC_DECL __declspec(dllexport)
#else
 #define FUNC_DECL __declspec(dllimport)
#endif


// Functions :

BOOL FUNC_DECL c842_open(void);
BOOL FUNC_DECL c842_close(void);

void  FUNC_DECL Init_LS(BYTE ax1, BYTE ax2, BYTE ax3, BYTE ax4);
short FUNC_DECL Board_Installed(void);
short FUNC_DECL InitBoard(BYTE mode);
void  FUNC_DECL InitAxis(BYTE axis);
int   FUNC_DECL autodetect(void);

void  FUNC_DECL translate(char *comstr, char *report);
short FUNC_DECL execute(BYTE axis, TCMD command, long para, char *report);
void  FUNC_DECL ClearAxisStatus(BYTE axis);
short FUNC_DECL axis_select(BYTE axis);
short FUNC_DECL get_pos4(long *position);
long  FUNC_DECL get_pos(BYTE axis);
short FUNC_DECL get_BoardStatus(void);    // added V25
short FUNC_DECL get_RefStatus(BYTE axis); // added V29
void  FUNC_DECL get_DLLversion(char *version);  // added V25
short FUNC_DECL setQMC(BYTE cmd, long para);
long  FUNC_DECL getQMC(BYTE cmd);
short FUNC_DECL QMCBusy(void);
short FUNC_DECL QMCReady(void);
void  FUNC_DECL Set_BaseAddress(unsigned address);
void  FUNC_DECL Set_Board(BYTE boardnumber);  // added V28

BYTE  FUNC_DECL Axis_Installed(void);
BYTE  FUNC_DECL moving(BYTE axis);

void  FUNC_DECL MoveA(BYTE axis, long position);
// short FUNC_DECL StatusBoard(void); use get_BoardStatus instead

#ifdef __cplusplus
 }
#endif
```

4.4. Function Reference

short **execute**(BYTE axis, TCMD command, long parameter, char *report);

execute an universal function that allows to transfer all commands to the board and to read the report string.

The full command set of the processor can be addressed as well as additional commands are available, having no direct implementation in the processor firmware.

Return: always 0

Example:

Move motor #3 for 5000 counts in positive direction relative to the present position, then wait until the new position is reached, then return home and indicate reaching the home position by a +5V signal at output channel # 8: :

execute(3, MR, 5000, report);	(move motor # 3 relative 5000 counts)
execute(3, WS, 0, report);	(wait until motor has reached the position)
execute(3, GH, 0, report);	(Start motor # 3 to run home)

void **translate**(char *CMDstr, char *report)

Like the execute function also the translate function has an universal character allowing to address all commands as well as the group of emulated commands. Unlike the execute function, translate works with strings containing the axis identifier, the command and the parameter.

Return: none

Examples:

translate("3MR5000",rep);	(move motor # 3 relative 5000 by counts)
translate("3WS",rep);	(wait until motor has reached its position)
translate("3GH",rep)	(Start motor # 3 to run home)
translate("3WS",rep)	(Wait until motor has reached its home)
translate("3TP",rep)	(report the position)

short **axis_select** (BYTE axis)

This procedure activates the selected axis. All consecutive issued commands are directed to this axis. Using the execute and the translate procedures, axis_select() is not required.

Return: TRUE if success, FALSE if failed

long **get_pos** (BYTE axis)

This function offers a fast access to the current motor encoder position of the specified axis. The return value is the position in counts.

Return: position in counts

short **get_pos4** (long *position)

This function offers a fast access to the current motor encoder positions of all 4 axes in counts. The parameter is a pointer to an array of 4 long integers representing the 4 motor positions.

Return: 0 if success, –1 if failed.

short **setQMC** (BYTE cmd, long para)

This procedure is used to transfer a processor command with the associated parameter value to the board.

Return: TRUE if success, FALSE if failed

long **getQMC** (BYTE cmd)

This function sends a query command to the C-842 and returns the numerical value of the requested register.

Return: long integer value

short **QMCBusy**(void)

This function waits until the controller is ready to accept new data.

Return: TRUE: (not busy) number of timeout counter, max 2500,
FALSE (busy) if timeout

short **QMCReady**(void)

This function waits until the C-842 becomes ready to accept data.

Return: TRUE if ready
FALSE if not ready

short **InitBoard**(BYTE mode)

This function initializes the C-842 performing these functions:

Sets the output to analog,
sets limit sense to 255,
sets the boardmode to the specified value

This function has to be called after the C-842 is powered up or after a reset was performed.

For details see Operating Manual MS 45E, keyword "board function register".

Return: board status byte if success
-1 if failed

void **Init_LS** (BYTE axis1, BYTE axis2, BYTE axis3, BYTE axis4)

This function sets the Limit_Sense Register Byte. For unknows configurations it can be replaced by the autodetect function.

The purpose of this function is that after power up, limit switch wiring and levels have to be defined. If the limit switches are not used, the axis parameter has to be set to 0, otherwise they have to be set to 1.

Example: One PI-stage is connected to axis #2. All other axes are not used or have stages without limit switch option. Then call Init_LS(0,1,0,0).

Return : none

---

void **InitAxis** (BYTE axis)

This procedure initializes the specified axis with default parameters.

Default parameters are not optimized to the current stages connected. Parameter optimization has to be done by individual setting of terms using the translate or execute functions.

This function can be called after InitBoard to set the axis to a operational state.

Return: none

---

short **Board_Installed** (void)

Return: TRUE if a C-842 is installed in a PC slot
FALSE if no controller is found.

---

void **Set_BaseAddress** (unsigned address)

This function sets the base address of the C-842. With this function multiple boards can be accessed in one PC.

If 0x210 is used as address (default address) and only this board is installed, this function is not needed.

See MS 45E Operating Manual for details.

BYTE **Axis_Installed** (void)

Return: Number of Axis supported by the processor of the C-842. If a C842.40 is installed, 4 is returned, in case of C-842.20 the return value is 2.

Int **autodetect** (void)

This function sets the limit sense register to the current level of the limit switch lines. It can be called once at the beginning of the program and replaces the function Init_LS.

No limit switch must be hit when autodetect is called !

Return: Limit_Sense value

BYTE **moving** (BYTE axis)

Depending on the moving status of the specified axis, this function returns TRUE if the motor is moving or FALSE if the motor profile position is on its target.

---

void **MoveA** (BYTE axis, long position)

This function moves the motor to a new position.

Return: none

short **get_BoardStatus** (void)

Return: Board status register.

For more information see MS 45E Operating Manual, page 10.

short **get_RefStatus** (BYTE axis)

New function in version 3.0

Parameter: Axis, range 1 to 4

Return: 0 if reference input signal is low
1 if reference input signal is high

Note: The genuine processor command GET_HOME reads the home input if no static level is applied.

For more information see MS 45E Operating Manual, page 10.

void **get_DLLversion** (char *version)

This function delivers a string with library version information. Max. 80 characters.

Return: none

5. List of QFL Commands

This list is valid for both, Win95 and NT libraries.

```
1    UP ,  // Update
     TP ,  // Tell Position
     GLS,  // Get Limit Switch
     TT ,  // Tell Target
     MUP,  // Multiple Update
     TS ,  // Tell Status
     GIP,  // Get Index Position
     MA ,  // Move Absolute
     SV ,  // Set Velocity
10   GMO,  // Get Mode of axis
     MR ,  // Move Relative
     GH ,  // Go Home
     TX ,  // Tell reference position
     WS ,  // Wait Stop
     MN ,  // Motor On
     MF ,  // Motor off
     TV ,  // Tell Velocity
     CLR,  // Clear Status
     AB    // Abort motion
20   ST ,  // Stop
     STS   // Stop Smooth
     SA ,  // Set Acceleration
     DH ,  // Define Home
     TY ,  // Tell programmed Velocity
     GPP,  // Get Profile Position
     GPV,  // Get Profile Velocity
     TF ,  // Tell actual Position Error
     RT ,  // Reset Board
     GB ,  // Get Breakpoint
30   GIR,  // Get status of interrupting axis
     SAN,  // Set Auto update on
     SAF,  // Set Auto update off
     TL ,  // Tell acceleration
     ASN,  // Auto Stop on
     ASF,  // Auto Stop off
     SMO,  // Set Motor Output
     TE ,  // Tell Error
     DP ,  // Define Proportional gain
     DD ,  // Define Differential gain
40   DI ,  // Define Integral gain
     DF ,  // Define velocity Feedforward gain
     DL ,  // Define integration Limit
     GP ,  // Get integral gain
     GD ,  // Get Differential gain
     GI ,  // Get Integral gain
     GF ,  // Get velocity Feedforward gain
     GL ,  // Get integration limit
     GPE,  // Get Position Error
     GSI,  // Get integrated position error
50   RTI,  // Get Time
     GIM,  // Get irq mask
     SCH,  // Set Capture Home
     SCI,  // Set Capture Index
     LN ,  // Set limit switch sensing on
     LF ,  // Set limit switch sensing off
     SPS,  // Set Profile to S-curve
     SPT,  // Set Profile to trapezoidal
     SMA,  // Set Maximum Acceleration
     GMA,  // Get Maximum Acceleration
```

```
60   SYN,  // Synchronize profile
     SPE,  // Set max Position Error Limit
     SLS,  // Set Limit Sense byte
     VE ,  // Tell Version
     GTI,  // Get system Time
     WA ,  // Wait Absolute
     SBT,  // Set Breakpoint to Time
     SBP,  // Set Breakpoint Positive Target
     SBN,  // Set Breakpoint Negative Target
     SBA,  // Set Breakpoint to positive act position
70   SBC,  // Set Breakpoint to negative act position
     SBF,  // Set Breakpoint mode off
     SB ,  // Set Breakpoint
     SIM,  // Set interrupt mask
     GJ ,  // Get Jerk
     SJ ,  // Set Jerk
     SRA,  // Set gear Ratio
     GRA,  // Get gear Ratio
     SPV,  // Set Profile mode to Velocity
     SPG,  // Set Profile mode to electronic Gear
80   AN ,  // Axis on
     AF ,  // Axis off
     SOP,  // Set output to PWM
     SOH,  // Set output to 16bit DAC
     SST,  // Set Sample Time
     GT ,  // Get sample Time
     GOM,  // Get current output mode
     TZ ,  // Tell channels in decimal
     TC ,  // Tell channel
     CN ,  // Channel on
90   CF ,  // Channel off
     IV ,  // Increment Velocity
     WC ,  // Wait Channel
     WT ,  // Wait Trigger
     FEP,  // find edge in positive direction
     FEN,  // find edge in negative direction
     RET   // return from motion error condition
```