NMC Cookbook - Function Calls for Using *PIC-SERVO*, *PIC-STEP* and *PIC-I/O* Controllers

The library file NMCLIB04.DLL contains all the functions used for communication with the ***PIC-SERVO***, ***PIC-STEP*** and ***PIC-I/O*** controllers. This Cookbook does not attempt to explain the operation of these controllers; rather, it has many examples of executing a variety of common operations using function calls to this DLL. Before starting to write any code, you should look over the entire list to get an idea of the scope of options available. Also, the controller chips have many additional capabilities not covered by the examples below, and you are urged to read the data sheets associated with each type of controller to better understand their operation. For more information, visit the web page *www.jrkerr.com/docs.html*.

All of the programming example below are written in C. If you are using basic, however, the function calls and function names will be exactly the same - only the syntax will vary slightly. For complete example programs in either C or Basic, please see our web page *www.jrkerr.com/software.html*.

In general, your application will interact with the NMC controller functions in five steps:

1. Inclusion of header files and other details for accessing the DLL.
2. Network initialization and verification of the presence of the controllers.
3. Initialization of the individual controllers.
4. Operation of the individual controllers (executing motions, reading data, etc.).
5. Shutdown of the controller network before your program terminates.

The examples below illustrate each of these five parts.

1. Headers and Libraries

Depending on what programming environment you are using, you will need to do different things to give you access to the functions within the NMCLIB04.DLL. Described here is what you need to do for Borland's C++ Builder, Microsoft's Visual C++ .net, and Microsoft's Visual Basic .net. For other programming environments, you will have to check your documentation for usage of 3rd party DLL's.

Borland C++ Users:

a. Add the following #include's in your program's source code files:

```
#include "sio_util.h"
#include "nmccom.h"
#include "picservo.h"
#include "picstep.h"
#include "picio.h"
```

b. Copy he files SIO_UTIL.H, NMCCOM.H, PIC-SERVO.H, PIC_STEP.H, and PICIO.H into the same folder as the rest of your source code.

c. Add the file NMCLIB04.LIB to your project

Microsoft Visual C++ . net Users:

a. Add the namespace declaration:

```
using namespace System::Runtime::InteropServices;
```

to your code to allow runtime linking to the DLL. Please look in the file FORM1.H in any of the VC++ examples to see where to put this line.

b. Add the following #include's in your program's source code files:

```
#include "sio_util.h"
#include "nmccom.h"
#include "picservo.h"
#include "picstep.h"
#include "picio.h"
```

c. Copy he files SIO_UTIL.H, NMCCOM.H, PIC-SERVO.H, PIC_STEP.H, and PICIO.H into the same folder as the rest of your source code.

Microsoft Visual Basic .net Users

a. Add the file NMCDEFINES.VB to your project

b. Copy the file NMCDEFINES.VB into the same folder as your other source code files.

For all three programming environments, you will have to place a copy of the NMCLIB04.DLL in the Windows system folder or else in the same folder as the executable you are creating. Copies of the .H or .VB header files needed for your environment can be found in the example programs on our web page *www.jrkerr.com/software.html*.

2. Network Initialization and Shutdown Functions

a. Use the following code segment to initialize the network of controllers:

```
int nummod;

ErrorPrinting(0);    //suppress printing of internal error messages

//Initialize NMC controllers on COM1: using 19200 Baud
nummod = NmcInit("COM1:", 19200);
```

`nummod` should be equal to the number of controller modules connected to the network.

b. Use the following code to read the type of module at a given address:

```
unsigned char modtype;
unsigned char addr;        //module address (1 - 32)
unsigned char stat_items;  //specifies which data should be returned

addr = 1;
stat_items = SEND_ID;

NmcReadStatus(addr, stat_items); //Use NmcReadStatus to read ID from
                                 //  controller
modtype = NmcGetModType(addr);   //retrieve the module type
```

`modtype` should equal 0 for a ***PIC-SERVO*** controller, 2 for a ***PIC-I/O*** and 3 for a ***PIC-STEP***.

c. Reset controllers to power-up state and shutdown the network:

```
//Reset the network of controllers - 0xFF resets all controllers
NmcHardReset(0xFF);

//Clean-up and close the COM port
NmcShutdown();
```

3. *PIC-SERVO* Motor Initialization

a. Use the following code segment to set the servo gains, enable the amplifier, and start the motor servoing to its current position:

```
unsigned char addr;        //module address
unsigned char ol, sr, dc;  //output limit, servo rate, deadband comp.
short int kp, kd, ki, il;  //position gain, derivative gain, intergral
                           //  gain, integration limit
unsigned char sm;          //step rate multiplier
unsigned char mode;        //specifies stopping options

addr = 1;            //set address for module 1
kp = 100;            //following gains are a good starting point
kd = 1000;           // for most motors and encoders
ki = 0;
il = 0;
ol = 255;
sr = 1;
dc = 0
mode = AMP_ENABLE | STOP_ABRUPT;

ServoSetGain2(addr, kp, kd, ki, il, ol, cl, el, sr, dc, sm);
ServoStopMotor(addr, mode);
```

The `AMP_ENABLE` flag should always be included in the `mode` byte when issuing and subsequent stop commands, unless you are actually wishing to disable the amplifier. (Note: If your amplifier requires the `AMP_ENABLE` bit to be low for normal operation, you should NOT include it.) The `STOP_ABRUPT` flag tells the ***PIC-SERVO*** to start servoing to its current position.

b. The following function can be used to reset the motor encoder position counter to zero:

```
ServoResetPos(addr);
```

c. If you are using the ***PIC-SERVO SC***, you have the option of setting the output mode and limit switch options. The following example sets the output mode for 3-phase commutation and enables the limit switch protection to stop the motor abruptly when a limit switch is hit. Note that the `ServoSetIoCtrl()` function should be called before enabling the amplifier as in the example above.

```
unsigned char addr;        //module address
unsigned char mode;        //specifies I/O Control options

addr = 1;                  //set address for module 1
mode = THREE_PHASE | LIMSTOP_ABRUPT;

ServoSetIoCtrl(addr, mode);
```

4. *PIC-SERVO* Motion Commands

a. Move to a goal position with acceleration ramping:

```
unsigned char addr;        //module address
unsigned char mode;        //motion options
long     pos, vel, acc;    //position, velocity & acceleration
unsigned char pwm;         //raw PWM value - unused

addr = 1;
mode = LOAD_POS | LOAD_VEL | LOAD_ACC | ENABLE_SERVO | START_NOW;
pos = 5000;          //move to position 5000 encoder counts
vel = 100000;        //100,000 ~= 3000 encoder counts per second
acc = 100;           //acceleration value
pwm = 0;             //pwm value is not used

ServoLoadTraj(addr, mode, pos, vel, acc, pwm);
```

Note that the position is an absolute (not relative) value referenced to the encoder's zero position. You should always allow one position move to complete before attempting to move to a new position (except for the ***PIC-SERVO SC***).

b. Detect when a motion is complete:

```
unsigned char addr;        //module address
unsigned char current_status_byte;
BOOL move_complete;

NmcNoOp(addr);       //update the status byte data
current_status_byte = NmcGetStat(addr);
move_complete = current_status_byte & MOVE_DONE;
```

Note that the `MOVE_DONE` bit is part of the status byte.

c. Move at a constant velocity with acceleration ramping:

```
unsigned char addr;        //module address
unsigned char mode;        //motion options
long     pos, vel, acc;    //position, velocity & acceleration
unsigned char pwm;         //raw PWM value - unused

addr = 1;
```

```
mode = LOAD_VEL | LOAD_ACC | ENABLE_SERVO | VEL_MODE | START_NOW;
pos = 0;             //position not used
vel = 100000;        //100,000 ~= 3000 encoder counts per second
acc = 100;           //acceleration value
pwm = 0;             //pwm value is not used

ServoLoadTraj(addr, mode, pos, vel, acc, pwm);
```

Note that the velocity value should always be positive. To move in the reverse direction, you should include the mode flag `REVERSE` when setting the `mode` byte.

d. Start a motion and decelerate to a stop automatically when a limit switch is hit:

```
unsigned char addr;        //module address
unsigned char mode;        //motion options
long     pos, vel, acc;   //position, velocity & acceleration
unsigned char pwm;         //raw PWM value - unused
unsigned char homing_mode; //homing command options
BOOL still_homing;         //flag for checking homing

//First, use the ServoSetHoming function to activate the homing function

addr = 1;
homing_mode = ON_LIMIT1 | HOME_STOP_SMOOTH;  //stop smoothly when limit
                                             // switch 1 is activated
ServoSetHoming(addr, homing_mode);


//Next, execute a motion which will move the motor towards the limit switch

mode = LOAD_POS | LOAD_VEL | LOAD_ACC | ENABLE_SERVO | START_NOW;
pos = 100000;        //move to position beyond the limit switch
vel = 100000;        //100,000 ~= 3000 encoder counts per second
acc = 100;           //acceleration value
pwm = 0;             //pwm value is not used

ServoLoadTraj(addr, mode, pos, vel, acc, pwm);

//Lastly, poll the HOME_IN_PROG bit to determine when the
//switch has been hit

do
   NmcNoOp(addr);          //update the status byte
   still_homing = NmcGetStat(addr) & HOME_IN_PROG;
while (still_homing);
```

e. Stop a motor with a deceleration ramp:

```
unsigned char addr;        //module address
unsigned char mode;        //specifies stopping options

mode = AMP_ENABLE | STOP_SMOOTH;
ServoStopMotor(addr, mode);
```

Note that you should use the procedure described in **4b** to determine when the motor has actually come to a stop. To stop abruptly, use `STOP_ABRUPT` instead of `STOP_SMOOTH`.

f. Turn the motor off and disable the amplifier:

```
unsigned char addr;        //module address
unsigned char mode;        //specifies stopping options

mode = MOTOR_OFF;
ServoStopMotor(addr, mode);
```

5. *PIC-SERVO* Reading Status Data

a. Read limit switches:

```
unsigned char addr;              //module address
unsigned char current_status;    //status byte value
BOOL switch1, switch2;           //boolean switch values

addr = 1;

NmcNoOp(addr);       //Limit switch values are in the status byte, so the
                     //  NoOp command can be used to update the status

current_status = NmcGetStat(addr);     //fetch the value of the status byte
switch1 = current_status & LIMIT1;
switch2= current_status & LIMIT2;
```

b. Read the position and velocity at the same time:

```
unsigned char addr;        //module address
long pos;                  //motor position in encoder counts
short int vel;             //motor vel. In encoder counts/servo tick
unsigned char stat_items;  //specify which data should be returned

stat_items = SEND_POS | SEND_VEL;  //specify both position and velocity
                                   //  should be read from contorller
NmcReadStatus(addr, stat_items);   //Read data from controllers

pos = ServoGetPos(addr);   //retrieve the position and velocity data
vel = ServoGetVel(addr);   //from the local internal data structure
```

Note that `NmcReadStatus()` does something very different from `NmcGetStat()`! You should also note that the velocity returned is only a 16 bit quantity, whereas the commanded velocity is 32 bits. The velocity returned should be approximately equal to the commanded velocity divided by 65,536 (2^{16}).

6. *PIC-STEP* Motor Initialization

a. Initialize the controller parameters and enable the amplifier:

```
unsigned char addr;        //module address
unsigned char param_byte;  //for setting operating parameters
unsigned char min_speed;   //minimum stepping speed
unsigned char run_cur, hold_cur;  //running and holding current
unsigned char therm_limit; //thermal limit parameter
unsigned char output_byte; //output bits required for configuring amplifier

addr = 1;

//SPEED_1X = use speed units of 25 steps/second
//Limit switch & e-stop inputs will have no effect on the motion
param_byte = SPEED_1X | IGNORE_LIMITS | IGNORE_ESTOP;

min_speed = 20;
run_cur = 100;       //running current is  approximately 1 amp
hold_cur = 20;       //holding current is approximately 200 ma.
therm_lim = 0;       //disable thermal limit
StepSetParam(addr, param_byte, nim_speed, run_cur, hold_cur, therm_lim);

output_byte = 0x02 + 0x04 + 0x08;  //set specific output bits as required
                                   //by the amplifier
StepSetOutputs(addr, output_byte);

StepStopMotor(1, AMP_ENABLE);    //Use StepStopMotor to enable amplifier
                                 //This will lock the motor into position
```

b. The following function can be used to reset the motor encoder position counter to zero:

```
StepResetPos(addr);
```

7. *PIC-STEP* Motion Commands

a. Move to a goal position with acceleration ramping:

```
unsigned char addr;        //module address
unsigned char mode;
long pos;
byte vel, acc;             //velocity and acceleration time
float raw_speed;           //raw speed values not used

addr = 1;

mode = LOAD_POS | LOAD_SPEED | LOAD_ACC | START_NOW;
pos = 5000;
vel = 100;     //with speed setting of SPEED_1X, vel of 100 = 2500 step/sec
acc = 10;
raw_speed = 0.0;  //raw_speed not used

StepLoadTraj(addr, mode, pos, vel, acc, raw_speed);  //start the motion
```

You should always wait for one position motion to finish before commanding another.

b. Detect when a motion is complete:

```
unsigned char addr;        //module address
unsigned char current_status_byte;
BOOL move_complete;

NmcNoOp(addr);       //update the status byte data
current_status_byte = NmcGetStat(addr);
move_complete = !(current_status_byte & MOTOR_MOVING);
```

Note that the `MOTOR_MOVING` bit is part of the status byte. You should also note that the `MOTOR_MOVING` flag which reports whether the motor is moving or not is used differently than the ***PIC-SERVO's*** `MOVE_DONE` flag.

c. Move at a constant velocity with acceleration ramping:

```
unsigned char addr;        //module address
unsigned char mode;
long pos;
byte vel, acc;             //velocity and acceleration time
float raw_speed;           //raw speed values not used

addr = 1;

mode = LOAD_SPEED | LOAD_ACC | START_NOW;
pos = 0;       //position is not used
vel = 100;     //with speed setting of SPEED_1X, vel of 100 = 2500 step/sec
acc = 10;
raw_speed = 0.0;  //raw_speed not used

StepLoadTraj(addr, mode, pos, vel, acc, raw_speed);  //start the motion
```

Note that the velocity value should always be positive. To move in the reverse direction, you should include the mode flag `STEP_REV` when setting the `mode` byte.

d. Start a motion and stop automatically when a limit switch is hit:

```
unsigned char addr;        //module address
unsigned char mode;
long pos;
byte vel, acc;             //velocity and acceleration time
float raw_speed;           //raw speed values not used
unsigned char home_mode;   //data used for homing
BOOL still_homing;

addr = 1;

home_mode = ON_LIMIT1 | HOME_STOP_ABRUPT;  //set to stop abruptly when
                                           // limit switch 1 is hit
StepSetHoming(addr, home_mode);            //start homing process

/* continued on next page */
```

```
//Now execute a motion towards the limit switch:
mode = LOAD_POS | LOAD_SPEED | LOAD_ACC | START_NOW;
pos = 50000;
vel = 100;     //with speed setting of SPEED_1X, vel of 100 = 2500 step/sec
acc = 10;
raw_speed = 0.0;  //raw_speed not used

StepLoadTraj(addr, mode, pos, vel, acc, raw_speed);  //start the motion

//Lastly, poll the HOME_IN_PROG bit to determine when the
//switch has been hit

do
   NmcNoOp(addr);          //update the status byte
   still_homing = NmcGetStat(addr) & HOME_IN_PROG;
while (still_homing);
```

e. Stop a motor with a deceleration ramp:

```
unsigned char addr;        //module address
unsigned char mode;        //specifies stopping options

mode = AMP_ENABLE | STOP_SMOOTH;
StepStopMotor(addr, mode);
```

Note that you can use the procedure described in **7b** to determine when the motor has actually come to a stop. To stop abruptly, use `STOP_ABRUPT` instead of `STOP_SMOOTH`.

f. Turn the motor off and disable the amplifier:

```
unsigned char addr;        //module address
unsigned char mode;        //specifies stopping options

mode = 0;
ServoStopMotor(addr, mode);
```

8. *PIC-STEP* Reading Status Data

a. Read limit switches:

```
unsigned char addr;              //module address
unsigned char input_byte;        //input byte value
BOOL switch1, switch2;           //boolean switch values

addr = 1;

NmcReadStatus(addr, SEND_INBYTE); //Limit switch values are in the
                                   // input byte

input_byte = StepGetInbyte(addr);      //fetch the value of the input byte
switch1 = current_status & FWD_LIMIT;
switch2= current_status & REV_LIMIT;
```

b. Read the position and input byte at the same time:

```
unsigned char addr;        //module address
long pos;                  //motor position in encoder counts
byte input_byte;           //input byte
unsigned char stat_items;  //specify which data should be returned

stat_items = SEND_POS | SEND_INBYTE;  //specify position and input byte
                                      //  should be read from controller
NmcReadStatus(addr, stat_items);      //Read data from controllers

pos = StepGetPos(addr);                //retrieve the position data
input_byte = ServoGetInbyte(addr);     //retrieve input byte
```

9. *PIC-I/O* Initialization

a. Set I/O bits as inputs or outputs:

```
unsinged char addr;        //module address

addr = 1;

IoBitDirIn(addr, 0);       //set I/O bit 0 to be an input
IoBitDirIn(addr, 1);       //set I/O bit 1 to be an input
IoBitDirOut(addr, 3);      //set I/O bit 1 to be an output
```

10. *PIC-I/O* Setting Outputs

a. Set or clear output bits:

```
unsigned char addr;        //module address

addr = 1;

IoSetOutBit(addr, 3);      //Set output bit 3 to logic 1
IoClrOutBit(addr, 3);      //Clear output bit 3 to logic 0
```

b. Set PWM output values:

```
unsigned char addr;        //module address
unsigned char pwm1, pwm2;  //output pwm values

addr = 1;
pwm1 = 128;          //set PWM 1 to a 50% duty cycle (128/256)
pwm2 = 64;           //set PWM 2 to a 25% duty cycle (64/256)

IoSetPWMVal(byte addr, byte pwm1, byte pwm2);
```

11. *PIC-I/O* Reading Inputs

a. Read input bits:

```
unsigned char addr;        //module address
BOOL bit0, bit1;           //input bit values

addr = 1;

NmcReadStatus(addr, SEND_INPUTS);  //read input bits from controller

bit0 = IoInBitVal(addr, 0);;     //retrieve bit 0 value
bit1 = IoInBitVal(addr, 1);;     //retrieve bit 1 value
```

b. Read all three analog inputs at the same time:

```
unsigned char addr;        //module address
unsigned char analog0, analog1, analog2;  //analog input values
unsigned char status_items;

addr = 1;

status_items = SEND_AD1 | SEND_AD2 | SEND_AD3;

NmcReadStatus(addr, status_items);  //Read analog values from controller

analog0 = IoGetADCVal(addr, 0);  //retrieve analog values
analog1 = IoGetADCVal(addr, 1);
analog2 = IoGetADCVal(addr, 2);
```