Notes on Using NMCLIB04.DLL

This document describes briefly the functions in NMCLIB04.DLL and compliments the data sheets for the PIC-SERVO, PIC-STEP and PIC-I/O controller chips. You will need to refer to these data sheets for detailed descriptions of the parameters referenced by the DLL functions.

NMCLIB04.DLL is a library of functions for using NMC (Networked Modular Control) compatible modules. It is an upgrade from NMCLIB0X.DLL, and includes support for the ***PIC-SERVO*** (v.4, v.5 and v.10) motor control modules, ***PIC-STEP*** stepper motor control modules, and ***PIC-IO*** multi-function I/O modules.

NMCLIB04.DLL consists of three levels of functions. At the lowest level is the group of serial I/O functions listed in SIO_UTIL.H. These functions provide basic (non-overlapped) COM port support independent of the NMC communication protocol. It includes functions for opening and closing COM ports, sending and receiving characters, etc. Typically, you will never need to call these functions directly.

The next level of functions, listed in NMCCOM.H, provide basic support of the NMC communication protocol. The functions at this level are independent of the types of modules used. They include initialization and reset functions for the entire network of modules, module control functions common to all module types such as reading or defining status data packet, and functions for retrieving data common to all module types.

The last level of functions are those specific to particular types of modules. These are described in PICSERVO.H, PIC-STEP.H and PICIO.H. They include functions for operations specific to each module type, and functions for retrieving module-type specific data.

In general, there are two types of module specific commands - ones for setting parameters or executing actions, and ones for retrieving data. Data for a module is located in one of two places: on the host PC, or in the module itself. Certain data sent to a module, like command velocities, for example, are not retrievable from the module. When such data are sent, however, the DLL stores the current value locally on the PC. This data can be retrieved at any time using the appropriate "Get" function such as `ServoGetCmdVel()`.

Data stored within the controller modules, such as position data, must first be read into the host PC. There are two ways to read in this data: you can use `NmcDefineStatus()` to have the data of interest returned to the host with every command sent to a module, or you can use `NmcReadStatus()` to read a particular bit of data just once. Once the current value of the data has been read from the controller, you can again use the appropriate "Get" function, such as `ServoGetPos()`, to retrieve the data from the DLL.

In addition to the function descriptions for each header file below, users should look at the example programs provided, and may even look at the source code provided for the DLL itself. The source code and make files are for Borland C++ Builder, although they should be portable to other C compilers.

SIO_UTIL.H (listed for reference):

`int SimpleMsgBox(char *msgstr)`
Displays a simple message box with the null-terminated string pointed to by ‘msgstr’. (This has nothing to do with serial I/O, but it is a generically useful function.)

`void ErrorPrinting(int f)`
Controls whether low-level error messages are printed or suppressed. Setting ‘f’ to zero will suppress error messages, setting ‘f’ to a non-zero value will enable the display of error messages. By default, error message printing is enabled.

`HANDLE SioOpen(char *name, unsigned int baudrate)`
Opens a COM port at the specified baud rate. Valid com port names are “COM*n*:”, where *n* = 1 - 8. Valid baud rates for use with NMC modules are 19200, 57600 or 115200 (also 230400 for ***PIC-SERVO SC***). Returns a handle for the COM port stream.

`BOOL SioPutChars(HANDLE ComPort, char *stuff, int n)`
Shoves n characters out the COM port specified by the handle ComPort.

`DWORD SioGetChars(HANDLE ComPort, char *stuff, int n)`
Read in n characters from the specified COM port and puts them in the array pointed to by 'stuff'. This function has a timeout value of approximately 100 millisec. It returns the number of characters actually read.

`DWORD SioTest(HANDLE ComPort)`
Returns the number of characters in the ComPort's input buffer.

`BOOL SioClrInbuf(HANDLE ComPort)`
Clears all of the characters in ComPort's input buffer.

`BOOL SioChangeBaud(HANDLE ComPort, unsigned int baudrate)`
Changes the baud rate of ComPort. Valid baud rates for use with NMC modules are 19200, 57600 or 115200 (also 230400 for ***PIC-SERVO SC***).

`BOOL SioClose(HANDLE ComPort)`
Closes a COM port.

NMCCOM.H

typedef unsigned char byte
The "byte" data type is defined as an unsigned char.

`int NmcInit(char *portname, unsigned int baudrate)`
Opens the COM port specified by 'portname' ("COM*n*:'', *n* = 1 - 8) at the specified baud rate (19200, 57600, 115200 or 230400*) and initializes the network of motor controllers. Controller addresses are dynamically assigned, starting with the furthest controller with address 1. All group addresses are set to 0xFF. Returns the number of controllers found on in the network.

*230400 Baud is only valid for PIC-SERVO SC modules

`BOOL NmcSendCmd(byte addr, byte cmd, char *datastr, byte n, byte stataddr)`
Sends to controller 'addr' the command byte 'cmd' followed by n additional data bytes in 'datastr'. 'stataddr' is the address of the module to which the returning status data belongs. Normally stataddr is the same as addr. For group commands with no group leader, addr is the group address and stataddr = 0. For group commands with a group leader, addr is the group address and stataddr is the individual address of the group leader. Returns true on success, false of failure.

`void NmcShutdown(void)`
Resets controllers assuming a default group address of 0xFF, and then closes the COM port in use.

`BOOL NmcSetGroupAddr(byte addr, byte groupaddr, BOOL leader)`
Sets the group address of a controller from the default of 0xFF. Valid group addresses are between 0x80 and 0xFF. If 'leader' is true, the specified controller becomes the group leader. Only one leader should be specified for any group address. Returns true on success, false on failure.

`BOOL NmcDefineStatus(byte addr, byte statusitems)`
For module 'addr', defines which status data will always be sent back with each command. 'statusitems' should be set to the bitwise OR of the status items listed in PICSERVO.H, PICSTEP.H or PICIO.H. Returns true on success, false on failure.

`BOOL NmcReadStatus(byte addr, byte statusitems)`
Reads status data from a module without changing the default status data. 'statusitems' should be set to the bitwise OR of the status items listed in PICSERVO.H, PICSTEP.H or PICIO.H. Returns true on success, false on failure.

`BOOL NmcChangeBaud(byte groupaddr, unsigned int baudrate)`
Sends a group command to all modules on the network to change their baud rate, and also changes the COM port's baud rate. 'groupaddr' should be 0xFF, unless the default group address

for all modules has been changed. All modules must have the same group address (with no group leader) before calling this function. Returns true on success, false on failure.

BOOL NmcNoOp(byte addr)
Issues a "No Operation" command to module 'addr', simply causing its currently defined data to be returned and stored in the module's local data structure. Returns true on success, false on failure.

BOOL NmcHardReset(byte addr)
Resets a controller to its powerup state. Under almost all circumstances, this is issued to a group address including all control modules. Returns true on success, false on failure.

byte NmcGetStat(byte addr) *
Returns the current status byte (stored locally) of a controller. Any command to a controller will update the locally stored value of the status byte.

byte NmcGetStatItems(byte addr)
Returns the byte specifying the default status items to be returned in the status data packet.

byte NmcGetModType(byte addr) †
Returns the module type of a particular module. This value will either be SERVOMODTYPE (0), STEPMODTYPE (3) or IOMODTYPE (2).

byte NmcGetModVer(byte addr) †
Returns the firmware version number of a particular module.

byte NmcGetGroupAddr(byte addr)
Returns the group address of a particular module.

BOOL NmcGroupLeader(byte addr)
Returns true if the specified module is a group leader, false if not.

BOOL NmcSynchOutput(byte addr, byte leaderaddr)
Causes a group of modules to synchronously output their previously buffered output commands. This will cause a ***PIC-SERVO*** or ***PIC-STEP*** module to execute a previously loaded trajectory and it will cause a ***PIC-IO*** module to output previously stored output bit and PWM values. 'leaderaddr' should be the individual address of the group leader, or 0 if there is no group leader.

BOOL NmcSynchInput(byte groupaddr, byte leaderaddr)
Causes a group of modules to synchronously capture input values. This will cause a ***PIC-SERVO*** or ***PIC-STEP*** module to store its current position as the home position and it will cause a ***PIC-IO*** module to store its input bits and timer values. 'leaderaddr' should be the individual address of the group leader, or 0 if there is no group leader.

PICSERVO.H

`long ServoGetPos(byte addr)` †
Returns the current motor position (stored locally) of a ***PIC-SERVO*** module. Use `NmcReadStatus()` or `NmcDefineStatus()` to update the locally stored position data.

`byte ServoGetAD(byte addr)` †
Returns the current A/D value (stored locally) of a ***PIC-SERVO*** module. Use `NmcReadStatus()` or `NmcDefineStatus()` to update the locally stored position data.

`short int ServoGetVel(byte addr)` †
Returns the current motor velocity (stored locally) of a ***PIC-SERVO*** module. Use `NmcReadStatus()` or `NmcDefineStatus()` to update the locally stored position data.

`byte ServoGetAux(byte addr)` †
Returns the current auxiliary status byte (stored locally) of a ***PIC-SERVO*** module. Use `NmcReadStatus()` or `NmcDefineStatus()` to update the locally stored position data.

`short int ServoGetPError(byte addr)` †
Returns the servo positioning error (stored locally) of a ***PIC-SERVO*** (***CMC*** or ***SC***) module. Use `NmcReadStatus()` or `NmcDefineStatus()` to update the locally stored position data.

byte ServoGetNPoints(byte addr) †
Returns the number of path points (stored locally) remaining in the ***PIC-SERVO*** (***CMC*** or ***SC***) path-point buffer. Use `NmcReadStatus()` or `NmcDefineStatus()` to update the locally stored position data.

`long ServoGetHome(byte addr)` †
Returns the current motor home position (stored locally) of a ***PIC-SERVO*** module. Use `NmcReadStatus()` or `NmcDefineStatus()` to update the locally stored position data.

`long ServoGetCmdPos(byte addr)`
Returns the most recently issued command position for a ***PIC-SERVO*** module.

`long ServoGetCmdVel(byte addr)`
Returns the most recently issued command velocity for a ***PIC-SERVO*** module.

`long ServoGetCmdAcc(byte addr)`
Returns the most recently issued command acceleration for a ***PIC-SERVO*** module.

`long ServoGetStopPos(byte addr)`
Returns the most recently issued stop position (by a ‘stop here’ command) for a ***PIC-SERVO*** module.

byte ServoGetCmdPwm(byte addr)
Returns the most recently issued command PWM value for a ***PIC-SERVO*** module.

byte ServoGetMoveCtrl(byte addr)
Returns the most recently issued move command control byte for a ***PIC-SERVO*** module.

byte ServoGetStopCtrl(byte addr)
Returns the most recently issued stop command control byte for a ***PIC-SERVO*** module.

byte ServoGetHomeCtrl(byte addr)
Returns the most recently issued home command control byte for a ***PIC-SERVO*** module.

byte ServoGetIoCtrl(byte addr)
Returns the most recently issued I/O command control byte for a ***PIC-SERVO*** module.

```
void ServoGetGain(byte addr, short int * kp, short int * kd,
                  short int * ki, short int * il, byte * ol,
                  byte * cl, short int * el, byte * sr,
                  byte * dc)
```
Returns the most recently issued servo gain values for a ***PIC-SERVO*** module.

```
void ServoGetGain2(byte addr, short int * kp, short int * kd,
                  short int * ki, short int * il, byte * ol,
                  byte * cl, short int * el, byte * sr,
                  byte * dc, byte * sm)
```
Returns the most recently issued servo gain values for a ***PIC-SERVO*** module. Use this version for the ***PIC-SERVO SC***.

```
BOOL ServoSetGain(byte addr, short int kp, short int kd,
                  short int ki, short int il, byte ol, byte cl,
                  short int el, byte sr, byte dc)
```
Sets the servo gains for a ***PIC-SERVO*** module. Returns true on success, false on failure. New applications should use `ServoSetGains2()` below for all versions of the ***PIC-SERVO***.

```
BOOL ServoSetGain2(byte addr, short int kp, short int kd,
                  short int ki, short int il, byte ol, byte cl,
                  short int el, byte sr, byte dc, byte sm)
```
Sets the servo gains for a ***PIC-SERVO*** module. New applications should this version of the Set Gains command for all versions of the ***PIC-SERVO***. Returns true on success, false on failure.

BOOL ServoResetPos(byte addr)
Resets the position counter for a ***PIC-SERVO*** module to zero. Returns true on success, false on failure.

BOOL ServoResetRelHome(byte addr)

Resets the position for a ***PIC-SERVO*** (***CMC*** or ***SC***) module relative to the home position register. (*i.e.*, the home position becomes position zero.) Returns true on success, false on failure.

BOOL ServoSetPos(byte addr, long pos)

Set the position of a ***PIC-SERVO v.10***) module to a specific value. Returns true on success, false on failure.

BOOL ServoClearBits(byte addr)

Clears the latched status bits (OVERCURRENT, POS_ERR, POS_WRAP) in a ***PIC-SERVO*** module. Returns true on success, false on failure.

BOOL ServoStopMotor(byte addr, byte mode)

Stops a motor in the manner specified by mode. Mode is the stop control byte, and should be set to the bitwise OR of the stop control bits defined in PICSERVO.H. By setting the ADV_FEATURE bit in the mode byte, this command can be used to enable the advanced features of the ***PIC-SERVO CMC***. Returns true on success, false on failure.

BOOL ServoSetIoCtrl(byte addr, byte mode)

Controls the configuration of the LIMIT1 and LIMIT2 I/O pins, as well as other miscellaneous functions. `mode` bits IO1_IN and IO2_IN determine whether the LIMIT1 and LIMIT2 pins are inputs (default) or outputs, bits SET_OUT1 and SET_OUT2 are used to set the state of any pins defined as outputs, and the bit FAST_PATH is used to enable "fast path" mode. Returns true on success, false on failure. Note that the bit definitions for the ***PIC-SERVO SC*** and for earlier versions are significantly different, and not generally compatible.

BOOL ServoLoadTraj(byte addr, byte mode, long pos, long vel, long acc, byte pwm)

Loads motion trajectory information for a ***PIC-SERVO*** module. 'mode' is the load trajectory control byte, and should be set to the bitwise OR of the trajectory control byte bits defined in PICSERVO.H. Note that if the START_NOW bit is set, the motion will begin immediately; if it is not set, the motion will be started when NmcSynchOutput() is called with the module's address. Returns true on success, false on failure.

void ServoInitPath(byte addr)

Calls NmcReadStatus and then initializes the starting point of a path to the current motor command position. This function should be called just prior to starting the path mode motion. This function requires that the advanced features of the ***PIC-SERVO CMC*** be first enabled using the ServoStopMotor command.

BOOL ServoAddPathpoints(byte addr, int npoints, long *path, int freq)

Adds between 1 and 7 path points to the ***PIC-SERVO CMC's*** internal path point buffer. 'npoints' is the number of points to be added, and the *absolute* path point positions should be loaded into the array 'path'. The variable 'freq' should be set to one of the constant values P_30HZ, P_60HZ or P_120HZ. This function will convert the absolute path data into incremental path data and

perform the proper bit formatting prior to downloading. This function requires that the advanced features of the ***PIC-SERVO CMC*** be first enabled using the ServoStopMotor command.

`BOOL ServoStartPathMode(byte groupaddr, byte leaderaddr);`
Starts the execution of a the path loaded into the ***PIC-SERVO CMC***'s internal path point buffer. 'groupaddr' is the group address for all controllers to be started and 'leaderaddr' is the individual address for the group's leader (use leaderaddr = 0 if there is no leader). This function requires that the advanced features of the ***PIC-SERVO CMC*** be first enabled using the ServoStopMotor command.

`BOOL ServoSetHoming(byte addr, byte mode)`
Starts the homing for ***PIC-SERVO*** module. 'mode' is the homing control byte and should be set to the bitwise OR of the homing control bits defined in PICSERVO.H. This command sets the homing conditions but does not start any motion. The HOME_IN_PROG bit of the status byte should be monitored to detect when the home position has been captured. Returns true on success, false on failure.

`BOOL ServoHardReset(byte addr, byte mode)`
Performs a hard reset for the ***PIC-SERVO SC*** with the option of saving or clearing data in the ***PIC-SERVO SC***'s EEPROM.

PICTEP.H

`long StepGetPos(byte addr)` †
Returns the current motor position of a ***PIC-TEP*** module.

`byte StepGetAD(byte addr)` †
Returns the current A/D value of a ***PIC-SERVO*** module.

`short int StepGetStepTime(byte addr)` †
Returns the current timer count for a ***PIC-STEP*** module.

`byte StepGetInbyte(byte addr)` †
Returns the current input byte of a ***PIC-SERVO*** module. This includes the E-stop, limit switch, homing switch and aux. input bits.

`long StepGetHome(byte addr)` †
Returns the current motor home position of a ***PIC-STEP*** module.

`long StepGetCmdPos(byte addr)`
Returns the most recently issued command position for a ***PIC-STEP*** module.

`byte StepGetCmdSpeed(byte addr)`
Returns the most recently issued command speed for a ***PIC-STEP*** module.

`byte StepGetCmdAcc(byte addr)`
Returns the most recently issued command acceleration time for a ***PIC-STEP*** module.

`unsigned short int StepGetCmdST(byte addr)`
Returns the most recently commanded timer count for a ***PIC-STEP*** module.

`byte StepGetMinSpeed(byte addr)`
Returns the minimum stepping speed for a ***PIC-STEP*** module.

`byte StepGetOutputs(byte addr)`
Returns the most recently commanded output byte for a ***PIC-STEP*** module.

`byte StepGetCtrlMode(byte addr)`
Returns the control mode byte (set with StepSetParam) for a ***PIC-STEP*** module.

`byte StepGetRunCurrent(byte addr)`
Returns the running current (set with StepSetParam) for a ***PIC-STEP*** module.

`byte StepGetHoldCurrent(byte addr)`
Returns the holding current (set with StepSetParam) for a ***PIC-STEP*** module.

byte StepGetThermLimit(byte addr)
Returns the thermal limit (set with StepSetParam) for a ***PIC-STEP*** module.

byte StepGetHomeCtrl(byte addr)
Returns the homing control byte for a ***PIC-STEP*** module.

byte StepGetStopCtrl(byte addr)
Returns the stopping control byte for a ***PIC-STEP*** module.

BOOL StepSetParam(byte addr, byte mode, byte minspeed,
byte runcur, byte holdcur, byte thermlim)
Sets the motion parameters for a ***PIC-STEP*** module. Returns true on success, false on failure.

BOOL StepResetPos(byte addr)
Resets the position counter for a ***PIC-STEP*** module. Returns true on success, false on failure.

BOOL StepStopMotor(byte addr, byte mode)
Stops a motor in the manner specified by mode. Mode is the stop control byte, and should be set to the bitwise OR of the stop control bits defined in PICSTEP.H. Returns true on success, false on failure.

BOOL StepSetOutputs(byte addr, byte outbyte)
Sets the general purpose output bits of a ***PIC-STEP*** module. Returns true on success, false on failure.

BOOL StepLoadTraj(byte addr, byte mode, long pos, byte speed,
byte acc, float raw_speed)
Loads motion trajectory information for a ***PIC-STEP*** module. 'mode' is the load trajectory control byte, and should be set to the bitwise OR of the trajectory control byte bits defined in PICSTEP.H. Note that if the START_NOW bit is set, the motion will begin immediately; if it is not set, the motion will be started when NmcSynchOutput() is called with the module's address. Returns true on success, false on failure.

BOOL StepSetHoming(byte addr, byte mode)
Starts the homing for ***PIC-STEP*** module. 'mode' is the homing control byte and should be set to the bitwise OR of the homing control bits defined in PICSTEP.H. This command sets the homing conditions but does not start any motion. The HOME_IN_PROG bit of the status byte should be monitored to detect when the home position has been captured. Returns true on success, false on failure.

PICIO.H

`BOOL IoInBitVal(byte addr, int bitnum)` †
Returns the value of an input bit (0 - 11) from a PIC-IO module. Note: this data is only valid if the SEND_INPUTS bit has been set in the most recently issued NmcDefineStat() command.

`BOOL IoInBitSVal(byte addr, int bitnum)` †
Returns the value of a synchronously captured input bit (bitnum = 0 - 11) from a PIC-IO module. Note: this data is only valid if the SEND_SYNCH_IN bit has been set in the most recently issued NmcDefineStat() command.

`byte IoGetADCVal(byte addr, int channel)` †
Returns the A/D value from channel 0, 1 or 2 from a PIC-IO module. Note: this data is only valid if the SEND_ADn (n=1,2 or 3) bit has been set in the most recently issued NmcDefineStat() command.

`unsigned long IoGetTimerVal(byte addr)` †
Returns the timer value from a PIC-IO module. Note: this data is only valid if the SEND_TIMER bit has been set in the most recently issued NmcDefineStat() command.

`unsigned long IoGetTimerSVal(byte addr)` †
Returns the synchronously captured timer value from a PIC-IO module. Note: this data is only valid if the SEND_SYNC_TMR bit has been set in the most recently issued NmcDefineStat() command.

`BOOL IoGetBitDir(byte addr, int bitnum)`
Returns 0 if a PIC-IO module I/O bit (bitnum = 0 - 11) is defined as an output, 1 if defined as an input.

`BOOL IoOutBitVal(byte addr, int bitnum)`
Returns the most recently set state of an output bit (bitnum = 0 - 11) of a PIC-IO module.

`byte IoGetPWMVal(byte addr, int channel)`
Returns the most recently set PWM value for channel 0 or 1 of a PIC-IO module.

`byte IoGetTimerMode(byte addr)`
Returns the most recently set timer control byte for a PIC-IO module.

`BOOL IoSetOutBit(byte addr, int bitnum)`
Sets the value of a PIC-IO module output bit (bitnum = 0 - 11) to 1. This has no effect if the bit is defined as an input. Returns true on success, false on failure.

`BOOL IoClrOutBit(byte addr, int bitnum)`
Clears the value of a PIC-IO module output bit (bitnum = 0 - 11) to 0. This has no effect if the bit is defined as an input. Returns true on success, false on failure.

BOOL IoBitDirIn(byte addr, int bitnum)
Sets the direction of a PIC-IO module I/O bit (bitnum = 0 - 11) to be an input bit. Returns true on success, false on failure.

BOOL IoBitDirOut(byte addr, int bitnum)
Sets the direction of a PIC-IO module I/O bit (bitnum = 0 - 11) to be an output bit. Returns true on success, false on failure.

BOOL IoSetPWMVal(byte addr, byte pwm1, byte pwm2)
Sets the PWM values for the two channels of a PIC-IO module. pwm1 and pwm2 should be between 0 and 255. Returns true on success, false on failure.

BOOL IoSetTimerMode(byte addr, byte tmrmode)
Sets the counter/timer mode for a PIC-IO module. 'tmrmode' is the timer control byte and should be set to the bitwise OR of the timer control bits defined in PICIO.H. Returns true on success, false on failure.

BOOL IoSetSynchOutput(byte addr, short int outbits, byte pwm1, byte pwm2)
Specify the output bit values and the PWM values to be output at a later time when NmcSynchOutput() is called.

---

* This function simply retrieves data from the module's data structure stored locally on the PC. To insure that the data is current, any command may be sent to the module just prior to calling this command.

† This function simply retrieves data from the module's data structure stored locally on the PC. To insure that the data is current, an NmcReadStatus (with the appropriate bit in the 'statusitems' byte set) should be issued just prior to calling this function. If NmcDefineStatus has already been used to permanently include the relevant data item in the default status packet, any command sent to the module will update the local data structure.