# Developers Interface Guide

For Motorized Zoom Systems and BrightLight® Controllers

*Visual C++ and Visual Basic.net*
*Serial and USB Connection*

Navitar Inc., 200 Commerce Drive, Rochester, NY 14623
Phone 800.828.6778 Fax 585.359.4999 www.navitar.com E-mail info@navitar.com

## Table of Contents

1. Introduction ..... 5
1.1. Other Documents to reference ..... 5
1.2. Products Covered ..... 5
2. Physical Connection Environments ..... 5
2.1. RS232 ..... 5
2.2. USB 2 ..... 5
3. Runtime considerations ..... 5
3.1. Dynamic Link Library (DLL) ..... 5
4. Development Environments ..... 6
4.1. Visual C++ ..... 6
4.2. Visual Basic.NET ..... 6
5. Sample code naming convention ..... 6
5.1. Serial Port Connection Layout ..... 7
6. Application Basics ..... 7
6.1. DLL linkage ..... 7
6.1.1. Visual C++ ..... 7
6.1.2. Visual Basic.NET ..... 8
6.2. Connection initialization ..... 8
6.2.1. Serial interface Visual C++ ..... 8
6.2.2. USB interface Visual C++ ..... 9
6.2.3. Serial interface Visual Basic.NET ..... 9
6.2.4. USB interface Visual Basic.NET ..... 10
7. Function call definitions ..... 11
7.1. Serial interface ..... 11
7.1.1. SerConnectionConnect ..... 11
7.1.2. SerConnectionEstablished ..... 11
7.1.3. SerConnectionDisconnect ..... 12

7.1.4. SerConnectionDisconnectAll ........ 12
7.1.5. SerConnectionWrite ........ 12
7.1.6. SerConnectionRead ........ 12
7.2. USB interface ........ 13
7.2.1. USBFindUSBinterfaces ........ 13
7.2.2. USBConnectionConnect ........ 13
7.2.3. USBConnectionConnectSpec ........ 13
7.2.4. USBConnectionEstablished ........ 14
7.2.5. USBConnectionDisconnect ........ 14
7.2.6. USBConnectionDisconnectAll ........ 14
7.2.7. USBConnectionWrite ........ 15
7.2.8. USBConnectionRead ........ 15
8. Motor Controller Register Definitions ........ 15
8.1. Motor Controller Identification Registers ........ 15
8.1.1. REG_SYS_PRODUCTID ........ 15
8.1.2. REG_SYS_VERSIONHW ........ 16
8.1.3. REG_SYS_VERSIONDATE ........ 16
8.1.4. REG_SYS_VERSIONSW ........ 16
8.2. Motor Controller System Setup Registers ........ 16
8.2.1. REG_SETUP_ACCEL ........ 17
8.2.2. REG_SETUP_INITVELOCITY ........ 17
8.2.3. REG_SETUP_MAXVELOCITY ........ 18
8.2.4. REG_SETUP_REVBACKLASH ........ 18
8.2.5. REG_SETUP_FWDBACKLASH ........ 18
8.2.6. REG_SETUP_CONFIG ........ 19
8.2.7. REG_SETUP_WRITE ........ 19
8.3. Motor Controller Operation Registers ........ 20
8.3.1. REG_USER_TARGET ........ 20
8.3.2. REG_USER_INCREMENT ........ 20
8.3.3. REG_USER_CURRENT ........ 20

8.3.4. REG_USER_LIMIT ..... 21

8.3.5. REG_USER_STATUS ..... 21

8.3.6. REG_SETUP_LIMIT ..... 22

9. BrightLight Register Definitions ..... 22

9.1. BrightLight Identification Registers ..... 22

9.1.1. REG_SYS_PRODUCTID ..... 22

9.1.2. REG_SYS_VERSIONHW ..... 23

9.1.3. REG_SYS_VERSIONDATE ..... 23

9.1.4. REG_SYS_VERSIONSW ..... 23

9.2. BrightLight Operation Registers ..... 23

9.2.1. REG_PWM_ABSOLUTE ..... 23

9.2.2. REG_PWM_INCREMENT ..... 24

# 1. Introduction

## 1.1. Other Documents to Reference

Motor Controller User’s Manual- Windows.doc

## 1.2. Products Covered

- Navitar 2 Phase Stepper Controller
- Navitar 5 Phase Stepper Controller
- Navitar DC Encoded Controller
- Navitar Digital BrightLight® Controllers

# 2. Physical Connection Environments

## 2.1. RS232

The covered products can be connected to a PC via an RS232 interface. The supplied **navserAPI.dll** configures the serial port with the necessary parameters and provides the connection.

## 2.2. USB 2

The covered products can be connected to a PC via a USB 2 interface. The supplied **navusbAPI.dll** provides the connection.

# 3. Runtime Considerations

## 3.1. Dynamic Link Library (DLL)

The low level communication interface for both the RS232 and USB environments is handled with a **Dynamic Link Library (DLL)**. The proper DLL must reside in the runtime path. The recommended location is to place the DLL in the same directory as the compiled executable.

## 4. Development Environments

### 4.1. Visual C++

Microsoft Visual C++ version 6.0 with service pack 5 or higher.

### 4.2. Visual Basic.NET

Microsoft Visual Basic.NET 2003 or higher.

## 5. Sample Code Naming Convention

There are 16 sample programs. The Visual Basic programs are located under the directory samples VB. The Visual C++ programs are located under the directory VC. Under each directory are 4 serial interface samples and 4 USB samples. Within each group of 4 samples there are the following samples:

1) BrightLight controller only
2) Motor controller only
3) 1 Motor controller and 1 BrightLight controller
4) 2 Motors controller and 2 BrightLight controller

The directory and project names are specified as follows:

| Environment | Interface | Number of Motor controllers | Number of BrightLight Controllers | Folder name |
|---|---|---|---|---|
| VB | USB | 0 | 1 | VB/ubm0b1 |
| VB | USB | 1 | 0 | VB/ubm1b0 |
| VB | USB | 1 | 1 | VB/ubm1b1 |
| VB | USB | 2 | 2 | VB/ubm2b2 |
| VB | serial | 0 | 1 | VB/sbm0b1 |
| VB | serial | 1 | 0 | VB/sbm1b0 |
| VB | serial | 1 | 1 | VB/sbm1b1 |
| VB | serial | 2 | 2 | VB/sbm2b1 |
| VC | USB | 0 | 1 | VC/ubm0b1 |
| VC | USB | 1 | 0 | VC/ubm1b0 |
| VC | USB | 1 | 1 | VC/ubm1b1 |
| VC | USB | 2 | 2 | VC/ubm2b2 |
| VC | serial | 0 | 1 | VC/sbm0b1 |
| VC | serial | 1 | 0 | VC/sbm1b0 |
| VC | serial | 1 | 1 | VC/sbm1b1 |
| VC | serial | 2 | 2 | VC/sbm2b2 |

### 5.1. Serial Port Connection Layout

The serial port samples use the following COM numbers:

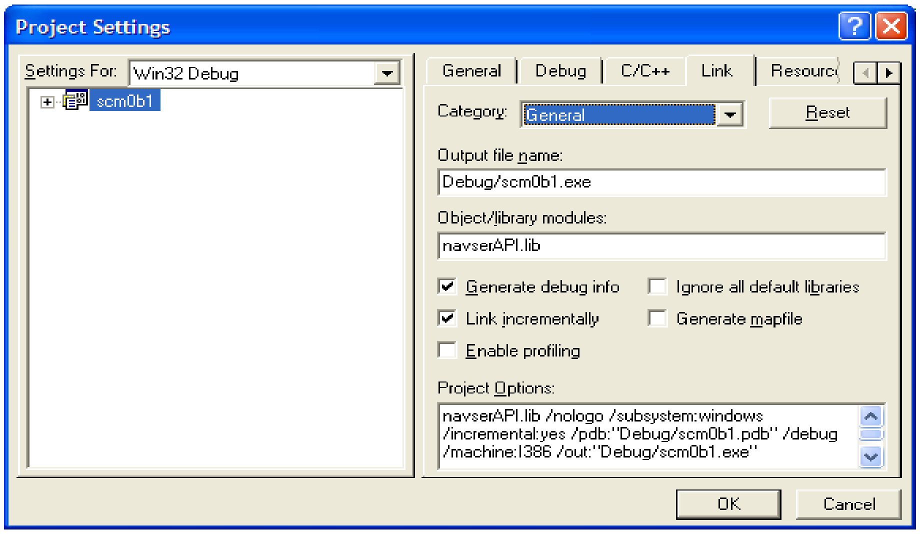
| Number of Motor Controller s | Number of BrightLight Controller s | COM1 | COM2 | COM3 | COM4 |
|---|---|---|---|---|---|
| 0 | 1 | BrightLight | | | |
| 1 | 0 | Motor | | | |
| 1 | 1 | Motor | BrightLight | | |
| 2 | 2 | Motor | Motor | BrightLight | BrightLight |

## 6. Application Basics

### 6.1. DLL linkage

#### 6.1.1. Visual C++

The application should include the header file navserAPI.h or navusbAPI.h. Under Project/Settings Link tab the library file (**navserAPI.lib** or **navusbAPI.lib**) must be included.



Category: General

Output file name:
Debug/scm0b1.exe

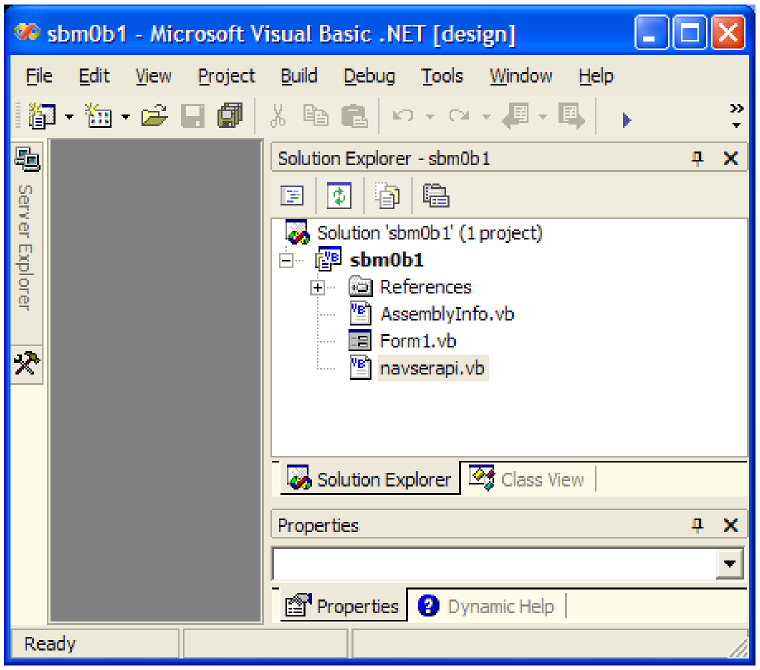
Object/library modules:
navserAPI.lib

[✓] Generate debug info
[ ] Ignore all default libraries
[✓] Link incrementally
[ ] Generate mapfile
[ ] Enable profiling

Project Options:
```
navserAPI.lib /nologo /subsystem:windows
/incremental:yes /pdb:"Debug/scm0b1.pdb" /debug
/machine:I386 /out:"Debug/scm0b1.exe"
```

### 6.1.2. Visual Basic.NET

The DLL interface description (**navserapi.vb** or **navusbapi.vb**) must be included in the project file.


sbm0b1 - Microsoft Visual Basic .NET [design]
File Edit View Project Build Debug Tools Window Help
Server Explorer
Solution Explorer - sbm0b1
Solution 'sbm0b1' (1 project)
sbm0b1
References
AssemblyInfo.vb
Form1.vb
navserapi.vb
Solution Explorer
Class View
Properties
Properties
Dynamic Help
Ready

## 6.2. Connection Initialization

### 6.2.1. Serial interface Visual C++

In order to establish a connection to a device the following code is used:

```
serHandle1 = SerConnectionConnect(COM1,DEF_MOTOR_CONTROLLER);
serHandle2 = SerConnectionConnect(COM2,DEF_BRIGHTLIGHT);
```

The returned serHandle will be used in future calls to the device.

See the definition of SerConnnectionConnect for more details.

Also examine the sample code near the end of the **OnInitDialog** function.

### 6.2.2. USB interface Visual C++

In order to establish a connection to a device the following line of code is used:

```
USBFindUSBinterfaces();

usbHandle1 = USBConnectionConnect(1,DEF_MOTOR_CONTROLLER);
usbHandle2 = USBConnectionConnect(1,DEF_BRIGHTLIGHT);
```

The returned usbHandle will be used in future calls to the device.

See the definition of USBConnnectionConnect for more details.

The call to USBFindUSBinterfaces must be called first. Then all the desired devices can be opened. The 'handles should be check for validity. If any handles are invalid, such as requested device not being connected or powered up the situation should be corrected and the sequence started again with the USBFindUSBinterfaces call.

If the application is running two different kinds of motor controllers the following is a more defined method of connecting to the correct motors:

```
USBFindUSBinterfaces();

usbHandle1 =
USBConnectionConnectSpec(1,DEF_MOTOR_CONTROLLER,DEF_MICROMO2PHASE);

usbHandle2 =
USBConnectionConnectSpec(1,DEF_MOTOR_CONTROLLER,DEF_VEXTA5PHASE);

usbHandle3 = USBConnectionConnect(1,DEF_BRIGHTLIGHT);
```

Examine the sample code near the end of the **OnInitDialog** function.

### 6.2.3. Serial interface Visual Basic.NET

In order to establish a connection to a device the following line of code is used:

```
serHandle1 = SerConnectionConnect(COM1,DEF_MOTOR_CONTROLLER)
serHandle2 = SerConnectionConnect(COM2,DEF_BRIGHTLIGHT)
```

The returned serHandle will be used in future calls to the device.

See the definition of SerConnnectionConnect for more details.

Also examine the sample code near the beginning of the:

```
#Region " Windows Form Designer generated code "
```
after the `InitializeComponent()` call.

#### 6.2.4. USB interface Visual Basic.NET

In order to establish a connection to a device the following line of code is used:

```
USBFindUSBinterfaces()

usbHandle1 = USBConnectionConnect(1,DEF_MOTOR_CONTROLLER)
usbHandle2 = USBConnectionConnect(1,DEF_BRIGHTLIGHT)
```

The returned usbHandle will be used in future calls to the device.

See the definition of USBConnnectionConnect for more details.

The call to USBFindUSBinterfaces must be called first. Then all the desired devices can be opened. The handles should be check for validity. If any handles are invalid, such as requested device not being connected or powered up the situation should be corrected and the sequence started again with the USBFindUSBinterfaces call.

```
USBFindUSBinterfaces()

' connect to the 1st MicroMo motor found
usbHandle1 =
USBConnectionConnectSpec(1, DEF_MOTOR_CONTROLLER, DEF_MICROMO2PHASE)

' connect to the 1st VEXTA motor controller found
usbHandle2 =
USBConnectionConnectSpec(1, DEF_MOTOR_CONTROLLER, DEF_VEXTA5PHASE)

' connect to the 1st BrightLight found
usbHandle3 = USBConnectionConnect(1, DEF_BRIGHTLIGHT)
```

Also examine the sample code near the beginning of the:

```
#Region " Windows Form Designer generated code "
```

after the `InitializeComponent()` call.

# 7. Function Call Definitions

## 7.1. Serial Interface

### 7.1.1. SerConnectionConnect

int SerConnectionConnect(int **port**,unsigned short **ProductID**);

| | |
|---|---|
| **port** | Specifies which COM port to connect to. |
| **ProductID** | Specifies which type of controller to connect to. |
| | DEF_BRIGHTLIGHT |
| | DEF_MOTOR_CONTROLLER |
| returns | An **id** to be used as a handle to future calls. |

This routine attempts to open a connection to a specified device on the specified port.

If 0 is returned it indicates the call failed. Causes could be that the port could not be opened or that the specified controller was not found on the port.

### 7.1.2. SerConnectionEstablished

int SerConnectionEstablished(int **id**,unsigned short **ProductID**);

| | |
|---|---|
| **id** | Handle returned by SerConnectionConnect. |
| **ProductID** | Specifies which type of controller to check. |
| | DEF_BRIGHTLIGHT |
| | DEF_MOTOR_CONTROLLER |
| returns | Connection status. |

This routine asks if a controller of a specified type is connected to the specified port. This returns TRUE if the specified controller is connected, otherwise it returns FALSE.

### 7.1.3. SerConnectionDisconnect

int SerConnectionDisconnect(int **id**);

| | |
|---|---|
| **id** | Handle returned by SerConnectionConnect. |

This routine disconnects the device on the specified handle.

### 7.1.4. SerConnectionDisconnectAll

int SerConnectionDisconnectAll(void);

This routine disconnects all devices opened with SerConnectionConnect.

### 7.1.5.SerConnectionWrite

SerConnectionWrite(int **id**,unsigned char reg,long ***v**);

| | |
|---|---|
| **id** | Handle returned by SerConnectionConnect. |
| **reg** | Register number. |
| ***v** | Pointer to the value to be written to the register. |

This call writes the value pointed to by **v** to the register **reg**.

See the section on Register Definitions for more details.

### 7.1.6.SerConnectionRead

int SerConnectionRead(int **id**,unsigned char **reg**,long ***v**);

| | |
|---|---|
| **id** | Handle returned by SerConnectionConnect. |
| **reg** | Register number. |
| ***v** | Pointer to the value to be read from the register. |

This routine reads a value to the location pointed to by **v** from register **reg**.

See the section on Register Definitions for more details.

## 7.2. USB Interface

### 7.2.1.USBFindUSBinterfaces

int USBFindUSBinterfaces(void);

This routine parses the USB chain looking for Navitar devices.

### 7.2.2.USBConnectionConnect

int USBConnectionConnect(int **device**,unsigned short **ProductID**);

| | |
|---|---|
| **device** | Specifies which of the specified product on the USB chain to connect to. |
| **ProductID** | Specifies which type of controller to connect to. |
| | DEF_BRIGHTLIGHT |
| | DEF_MOTOR_CONTROLLER |
| returns | An **id** to be used as a handle to future calls. |

This routine attempts to open a connection to a specified device on the USB chain.

If 0x100 is returned it indicates the call failed. Causes could be that specified controller was not found on the port.

**Example:**

**To connect to the first motor controller on the USB chain:**

```
handle = USBConnectionConnect(1,DEF_MOTOR_CONTROLLER);
```

### 7.2.3.USBConnectionConnectSpec

int USBConnectionConnectSpec(int **device**,unsigned short **ProductID**,unsigned short **ProductType**);

| | |
|---|---|
| **device** | Specifies which of the specified product on the USB chain to connect to. |
| **ProductID** | Specifies which type of controller to connect to. |
| | DEF_BRIGHTLIGHT |
| | DEF_MOTOR_CONTROLLER |
| returns | An **id** to be used as a handle to future calls. |

This routine attempts to open a connection to a specified device on the specified port.

If 0x100 is returned it indicates the call failed. Causes could be that specified controller was not found on the port.

**Example:**

**To connect to the first motor controller on the USB chain:**

```
handle = USBConnectionConnect(1,DEF_MOTOR_CONTROLLER,DEF_VEXTA5PHASE);
```

### 7.2.4. USBConnectionEstablished

int USBConnectionEstablished(int **device**,unsigned short **ProductID**);

| | |
|---|---|
| **device** | Handle returned by USBConnectionConnect. |
| **ProductID** | Specifies which type of controller to check. |
| | DEF_BRIGHTLIGHT |
| | DEF_MOTOR_CONTROLLER |
| returns | connection status |

This routine asks if a controller of a specified type is connected to the specified port.

This returns TRUE if the specified controller is connected, otherwise it returns FALSE.

### 7.2.5. USBConnectionDisconnect

int USBConnectionDisconnect(int **id**);

| | |
|---|---|
| **id** | Handle returned by USBConnectionConnect |

This routine disconnects the device on the specified handle.

### 7.2.6. USBConnectionDisconnectAll

int USBConnectionDisconnectAll(void);

This routine disconnects all devices opened with USBConnectionConnect.

### 7.2.7. USBConnectionWrite

USBConnectionWrite(int **device**,unsigned char **reg**,long ***v**);

| | |
|---|---|
| **device** | Handle returned by USBConnectionConnect |
| **reg** | Register number |
| ***v** | Pointer to the value to be written to the register |

This call writes the value pointed to by **v** to the register **reg**.

See the section on Register Definitions for more details.

### 7.2.8. USBConnectionRead

int USBConnectionRead(int **device**,unsigned char **reg**,long ***v**);

| | |
|---|---|
| **device** | Handle returned by USBConnectionConnect |
| **reg** | Register number |
| ***v** | Pointer to the value to be read from the register |

This routine reads a value to the location pointed to by **v** from register **reg**.

See the section on Register Definitions for more details.

# 8. Motor Controller Register Definitions

The section entitled “Motor Controller Operation Registers” is probably the most interesting.

## 8.1. Motor Controller Identification Registers

### 8.1.1. REG_SYS_PRODUCTID

| | |
|---|---|
| REG_SYS_PRODUCTID(0x01) | read only |

return value:

either

| | |
|---|---|
| DEF_BRIGHTLIGHT | 0x4000 |
| DEF_MOTOR_CONTROLLER | 0x4001 |

#### 8.1.2.REG_SYS_VERSIONHW

REG_SYS_VERSIONHW(0x02) read only

return value:

32 bit hardware version identifier.

#### 8.1.3.REG_SYS_VERSIONDATE

REG_SYS_VERSIONDATE(0x03) read only

return value:

32 bit build date identifier.

0x00,YY,MM,DD

example:

0x00050402 would be 2005, April, 2

#### 8.1.4.REG_SYS_VERSIONSW

REG_SYS_VERSIONSW(0x04) read only

return value:

32 bit software version identifier.

## 8.2. Motor Controller System Setup Registers

These system registers are intended to be system setup registers. Changing these may result in a system that does not perform properly. The default values for these registers are listed in the document: Motor Controller User's Manual-Windows.

### 8.2.1.REG_SETUP_ACCEL

| | |
|---|---|
| REG_SETUP_ACCEL_1(0x15) | sets motor 1 acceleration |
| REG_SETUP_ACCEL_2(0x25) | sets motor 2 acceleration |

For Two and Five phase motors, these 2 registers set the acceleration value. If this value is set too high it is possible that the motor may stall.

For DC Encoded motors, these 2 registers set the velocity gain component. The predominant effect of this is controlling the deceleration of the motor.

**WARNING: Changing these values from the factory defaults may result in a system that does not perform correctly.**

See the document **Motor Controller User's Manual-Windows.doc** for the proper factory defaults.

### 8.2.2.REG_SETUP_INITVELOCITY

| | |
|---|---|
| REG_SETUP_INITVELOCITY_1(0x16) | sets initial motor 1 velocity |
| REG_SETUP_INITVELOCITY_2(0x26) | sets initial motor 2 velocity |

For Two and Five phase motors, these 2 registers set the initial step rate. If this value is set too high it is possible that the motor may stall on startup.

For DC Encoded motors, these 2 registers set the error gain parameter.

**WARNING: Changing these values from the factory defaults may result in a system that does not perform correctly.**

See the document **Motor Controller User's Manual-Windows** for the proper factory defaults.

### 8.2.3.REG_SETUP_MAXVELOCITY

| | |
|---|---|
| REG_SETUP_MAXVELOCITY_1(0x17) | sets maximum motor 1 velocity |
| REG_SETUP_MAXVELOCITY_2(0x27) | sets maximum motor 2 velocity |

For Two and Five phase motors, these 2 registers set the maximum step rate. If this value is set too high the motor may skip steps.

For DC Encoded motors, these 2 registers set the maximum torque.

**WARNING: Changing these values from the factory defaults may result in a system that does not perform correctly.**

See the document **Motor Controller User's Manual-Windows** for the proper factory defaults.

### 8.2.4.REG_SETUP_REVBACKLASH

REG_SETUP_REVBACKLASH_1(0x18) sets motor 1 reverse backlash value

REG_SETUP_REVBACKLASH_2(0x28) sets motor 2 reverse backlash value

These 2 registers set the number of position count to use for backlash when traveling in the reverse direction. This value will have no effect if the motor is setup to reverse seek thru home.

### 8.2.5.REG_SETUP_FWDBACKLASH

REG_SETUP_FWDBACKLASH_1(0x19) sets motor 1 forward backlash value

REG_SETUP_FWDBACKLASH_2(0x29) sets motor 2 forward backlash value

These 2 registers set the number of position count to use for backlash when traveling in the forward direction. This value is normally set to zero.

### 8.2.6.REG_SETUP_CONFIG

| | |
|---|---|
| REG_SETUP_CONFIG_1(0x1B) | sets motor 1 sensor configuration |
| REG_SETUP_CONFIG_2(0x2B) | sets motor 2 sensor configuration |

These 2 registers setup the sensor configuration and homing operation.

| | |
|---|---|
| Bit 0: | 0 – near sensor is home |
| | 1 – far sensor is home |
| Bit 1: | 0 – reverse seek direct |
| | 1 – reverse seek thru home |
| Bit 2: | 0 – Axis 2 enabled (only for AXIS 2) |
| | 1 – Axis 2 disabled (only for AXIS 2) |

The default is for the near sensor to be home and to reverse seek thru home.

### 8.2.7.REG_SETUP_WRITE

| | |
|---|---|
| REG_SETUP_WRITE(0x0E) | commits system register changes |

Writing a 0x1 to this register commits any and all changes made to the setup registers to flash memory. This allows the changes to survive beyond a power down in the controller.

**Example:**

```
// set near sensor to home and turn off reverse seek thru home
v = 0;
ConnectionWrite(REG_SETUP_CONFIG_1,&v)
v = 0;
ConnectionWrite(REG_SETUP_CONFIG_2,&v)


// commits the changes to flash memory
// so it survives the next controller power cycle.
v = 0x00000001;
ConnectionWrite(REG_SETUP_WRITE,&v);
```

## 8.3. Motor Controller Operation Registers

These user registers are intended for normal operation.

### 8.3.1. REG_USER_TARGET

Writing to either of these two registers commands the motor to seek to the specified value.

| | |
|---|---|
| REG_USER_TARGET_1(0x10) | motor 1 seek command |
| REG_USER_TARGET_2(0x20) | motor 2 seek command |

**Example: See the appropriate code sample.**

### 8.3.2. REG_USER_INCREMENT

Writing to either of these two registers commands the motor to advance to retreat by the specified value. This command is intended for use to make minor adjusts to the position and do not use the backlash parameters or the seek thru home capabilities.

| | |
|---|---|
| REG_USER_INCREMENT_1(0x11) | motor 1 delta seek from current |
| REG_USER_INCREMENT_2(0x21) | motor 2 delta seek from current |

**Example: See the appropriate code sample.**

### 8.3.3. REG_USER_CURRENT

These two registers are used to query the current motor position.

| | |
|---|---|
| REG_USER_CURRENT_1(0x12) | motor 1 current position |
| REG_USER_CURRENT_2(0x22) | motor 2 current position |

**Example: See the appropriate code sample.**

### 8.3.4.REG_USER_LIMIT

Writes to these registers command the motors to do one of the following three operations:

| | |
|---|---|
| REG_USER_LIMIT_1(0x13) | motor 1 limit seek command |
| REG_USER_LIMIT_2(0x23) | motor 2 limit seek command |

The value written drives the motor to one of the limits:

0 – drive motor to home.

1 – drive motor to limit.

2 – abort current motor motion.

**Example: See the appropriate code sample.**

### 8.3.5.REG_USER_STATUS

Reads from these registers query the motor status.

The low 8 bits indicate the motion state. While the motor is in motion the lower 8 bits will return a non zero value.

REG_USER_STATUS_1(0x14)

REG_USER_STATUS_2(0x24)

Return values:

Lower 8 bits (Bit 0 thru 7).

| Hex | Binary | Motor motion state |
|---|---|---|
| 0x0 | 0000 | 0 – idle |
| 0x1 | 0001 | 1 – driving to home |
| 0x2 | 0010 | 2 – coming off home |
| 0x3 | 0011 | 3 – driving to limit |
| 0x4 | 0100 | 4 – seeking forward |
| 0x5 | 0101 | 5 – deaccel in forward direction |
| 0x6 | 0110 | 6 – forward backlash (i.e. rev after overshoot) |
| 0x7 | 0111 | 7 – seeking reverse |
| 0x8 | 1000 | 8 – deaccel in reverse direction |
| 0x9 | 1001 | 9 – reverse backlash (i.e. forward after overshoot) |
| 0xB | 1011 | 11- forward deaccel during an abort |
| 0xC | 1100 | 12- reverse deaccel during an abort |

Bit 8 – motor on home sensor

Bit 9 – motor on limit sensor

When testing for the end of motion, just the lower 8 bits should be compared to zero. If the entire word is compared zero the motion may be completed but we could be stopped on the limit sensor. The return value in this case would be 0x0000000200.

**Example: See the appropriate code sample**

#### 8.3.6.REG_SETUP_LIMIT

Reads from these registers return the values at the number of position counts between HOME and LIMIT.

| | |
|---|---|
| REG_SETUP_LIMIT_1(0x1C) | returns motor 1 limit value |
| REG_SETUP_LIMIT_2(0x2C) | returns motor 2 limit value |

This value is measured during the power up motor calibration sequence.

# 9. BrightLight Register Definitions

## 9.1. BrightLight Identification Registers

### 9.1.1.REG_SYS_PRODUCTID

REG_SYS_PRODUCTID(0x01) read only

Return value:

Either

| | |
|---|---|
| DEF_BRIGHTLIGHT | 0x4000 |
| DEF_MOTOR_CONTROLLER | 0x4001 |

#### 9.1.2. REG_SYS_VERSIONHW

REG_SYS_VERSIONHW(0x02) read only

Return value:

32 bit hardware version identifier.

#### 9.1.3. REG_SYS_VERSIONDATE

REG_SYS_VERSIONDATE(0x03) read only

Return value:

32 bit build date identifier.

0x00,YY,MM,DD

**Example:**

**0x00050402 would be 2005, April, 2**

#### 9.1.4. REG_SYS_VERSIONSW

REG_SYS_VERSIONSW(0x04) read only

Return value:

32 bit software version identifier.

### 9.2. BrightLight Operation Registers

#### 9.2.1. REG_PWM_ABSOLUTE

REG_PWM_ABSOLUTE(0x40) read/write

This value is in 0.1 increments

| | |
|---|---|
| 0 | -> 0% |
| 1000 | -> 100% |

### 9.2.2.REG_PWM_INCREMENT

REG_PWM_INCREMENT(0x41) write only

+/- change to current PWM value

This value is in 0.1 increments

| | |
|---|---|
| 10 | -> 1% |
| -10 | -> -1% |