

# Compass B


## User's Guide

Revision 2.0



*Passion for innovation*

# Trademark

Innovati®, , and BASIC Commander® are registered trademarks of Innovati, Inc.

InnoBASIC™ and, cmdBUS™ are trademarks of Innovati, Inc.

Copyright © 2013 by Innovati, Inc. All Rights Reserved.

Due to continual product improvements, Innovati reserves the right to make modifications to its products without prior notice. Innovati does not recommend the use of its products for application that may present a risk to human life due to malfunction or otherwise.

No part of this publication may be reproduced or transmitted in any form or by any means without the expressed written permission of Innovati, Inc.

# Disclaimer

Full responsibility for any applications using Innovati products rests firmly with the user and as such Innovati will not be held responsible for any damages that may occur when using Innovati products. This includes damage to equipment or property, personal damage to life or health, damage caused by loss of profits, goodwill or otherwise. Innovati products should not be used for any life saving applications as Innovati's products are designed for experimental or prototyping purposes only. Innovati is not responsible for any safety, communication or other related regulations. It is advised that children under the age of 14 should only conduct experiments under parental or adult supervision.

# Errata

We hope that our users will find this user's guide a useful, easy to use and interesting publication, as our efforts to do this have been considerable. Additionally, a substantial amount of effort has been put into this instruction manual to ensure accuracy and complete and error free content, however it is almost inevitable that certain errors may have remained undetected. If you find any errors in the instruction manual, contact us via email [service@innovati.com.tw](mailto:service@innovati.com.tw). For the most up-to-date information, please visit our web site at <http://www.innovati.com.tw>.

# Table of Contents

- Overview ..... 4
- Applications ..... 4
- Features ..... 4
- Connection ..... 4
- Specifications ..... 5
- Calibration ..... 5
- Commands and Events ..... 6
- Example Program ..... 9
- Appendix A --- Module ID Setting ..... 10
- Appendix B --- I2C Format Command Table ..... 11
- Appendix C --- C language Format ..... 14

## Overview

Innovati's Compass B module is an easy-to-use, high precision electronic compass. Through the cmdBUS™ and BASIC Commander®, you can get the azimuth and the magnetic field intensity. In addition, the user calibration feature is provided to eliminate magnetic field measurement error generated by magnetic components in surrounding environment.

## Applications

- Designs to obtain azimuth and magnetic intensity electronically.
- Projects with deviation measurement feature to enable vehicle to move autonomously.
- Applications related to magnetic field intensity measurement.

## Features

- Azimuth measurement with 1 to 2 degree heading accuracy.
- 3-axis magnetic field intensity measurement.
- Enable event when the heading direction is beyond the deviation range.
- Six refresh rates, up to 50 times per second, for automatic azimuth measurement.
- Software or manual calibration by button is available.
- Detectable magnetic intensity up to  $\pm 8$  Gauss with 2 milli-Gauss field resolution.
- I2C Command syntax available for generic microcontrollers.
- Operating temperature: 0 °C ~ 70°C
- Storage temperature: -40°C~125°C

## Connection

To access Compass B through BASIC Commander®, set the DIP switch to the desired module ID setting (see Appendix A), and connect the cmdBUS™ cable between the module and the BASIC Commander®.

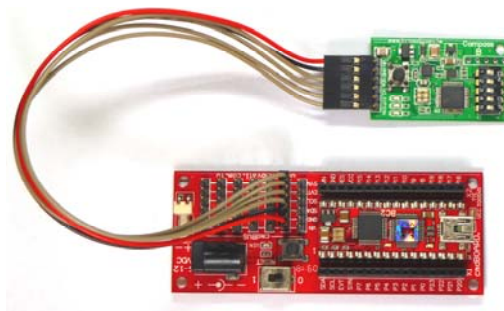


Figure 1: Connection with the BASIC Commander®

# Specifications

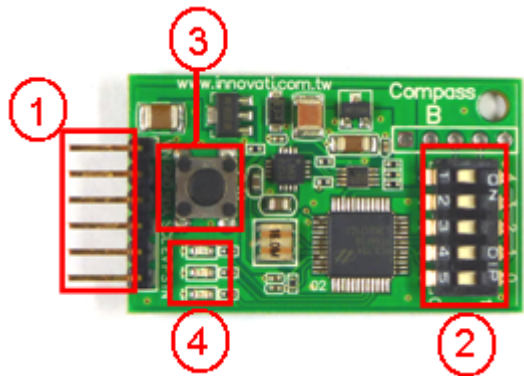


Figure 2: Pin assignment and switch description

| Item | Description  |
|------|--|
| 1    | CmdBUS™ pins: To access the Compass B module through the BASIC Commander®, connect these pins to the corresponding pins on the BASIC Commander®. Pay attention to pin assignments while connecting. Incorrect connection may damage the modules permanently.   |
| 2    | Module ID Switch: Each module needs to be assigned its own ID not conflicting with other modules on the same CmdBUS™. This assigned ID will be used in the program so the BASIC Commander® can communicate with module correctly. Refer to Appendix A for DIP switch setting.  |
| 3    | Manual Calibration button: Follow the instruction in Calibration section. The calibration LED will blink during calibration. When done, press the button again to exit the calibration mode and the calibration LED will turn off. To reset the settings, press and hold the calibration button for more than 7 seconds, all the indicator LEDs blink twice and default calibration settings will be restored. |
| 4    | Indicator LEDs from top to bottom are:<br>Calibration LED: blinks when the module is in calibration mode;<br>Event LED: blinks when the module is transmitting events;<br>Status LED: blinks when the module is communicating with BASIC Commander®.   |

## Calibration

To set the module for calibration within the program, use the Calibrate() command. For more information of this command, see the command table. To set the module manually, press and hold the Calibration button for two seconds to enter the calibration mode.

Once the module is in calibration mode, keep the module horizontal and rotate it clockwise or counterclockwise, and perpendicular to the z-axis, as shown in the figure below. Note that the calibration LED is blinking during calibration. Please do not rotate the module too fast so that

the module can determine the limits in all axes, and do not rotate the module more than 360 degrees.

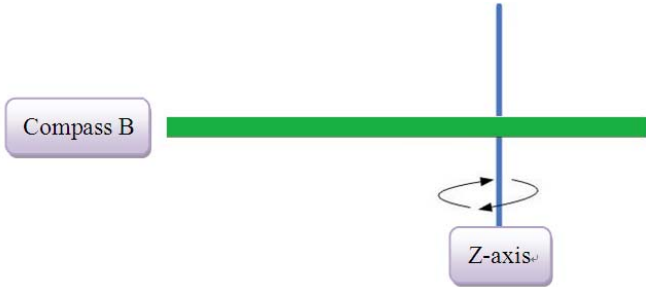


Figure 3: Calibration

Place the module horizontally during operation to get accurate measurements. The axis definition is illustrated as below.

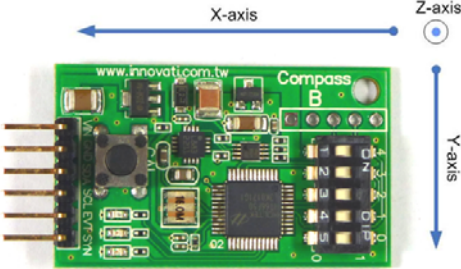


Figure 4: Axis definition

## Commands and Events

The following tables list all the unique commands and events provided with the Compass B module. Note that essential words in the commands will be written in **bold** type and *italics* in bold type. The bold type word must be written exactly as shown, whereas the italic bold type words must be replaced with the user values. Note that the innoBASIC language is case-insensitive.

Before executing the command of Compass B module, declare the corresponding parameters and the module ID at the beginning of the program, for example:

```
Peripheral ModuleName As CompassB @ ModuleID
```

In addition to the featured cmdBUS™ commands, I2C commands are also available for users who would like to use our modules in their systems. Check the Appendix B for details.

| Command Format   | Description                                      |
|--|--|
| <b>Magnetic Field Intensity and Direction Measurement Commands</b> |  |
| <b>GetXField</b> ( <i>Fx</i> )                                     | Read the difference between the central magnetic |

|   |   |
|---|---|
| <b>GetYField(<i>Fy</i>)</b>                                 | field intensity and the axial magnetic field intensity of the x-axis, y-axis and z-axis and stored in <i>Fx</i> , <i>Fy</i> and <i>Fz</i> , respectively. The return value ranges from -32768 ~ 32767.  |
| <b>GetZField(<i>Fz</i>)</b>                                 |   |
| <b>GetAngle(<i>Ang</i>)</b>                                 | Read the azimuth angle of the magnetic North with respect to the axes assigned in the <b>SetDimension()</b> command in unit of degrees. The angle is stored in <i>Ang</i> ranging from 0~359.   |
| <b>GetAngle3D(<i>Ang1</i>, <i>Ang2</i>)</b>                 | Read the azimuth angle of the projection of magnetic North on x-y plane with respect to the x-axis in unit of degrees. The angle is stored in <i>Ang1</i> ranging from 0~359. The included angle between the magnetic North and the z-axis is stored in <i>Ang2</i> ranging from 0~179.                 |
| <b>Setting of Deviation Angles and Measurement Commands</b> |   |
| <b>SetCurrentTargetAngle()</b>                              | Save the currently measured angle in non-volatile memory.   |
| <b>SetTargetAngle (<i>Ang</i>)</b>                          | Save the value of target <i>Ang</i> ranging from 0 to 359 in non-volatile memory.   |
| <b>GetTargetAngle (<i>Ang</i>)</b>                          | Read the value ranging from 0 to 359 from non-volatile memory to <i>Ang</i> .   |
| <b>GetDevAngle(<i>Ang</i>)</b>                              | Read the included angles of the current direction with respect to the preset base direction in unit of degree and saves the value in <i>Ang</i> . The value ranges from 0~180 counterclockwise with respect to the base direction, and ranges from 0~-179 clockwise with respect to the base direction. |
| <b>SetDevAngleLimit(<i>Ang</i>)</b>                         | Set the limit of deviation angle in unit of degrees. <i>Ang</i> ranges from 0~179 and its default value is 5.   |
| <b>GetDevAngleLimit(<i>Ang</i>)</b>                         | Return the current deviation angle limit in unit of degrees and saves the value in <i>Ang</i> ranging from 0~179.   |
| <b>EnableDevAngleLimitEvent()</b>                           | Enable the event to trigger when the deviation angle exceed the limit.  |
| <b>DisableDevAngleLimitEvent()</b>                          | Disable the Deviation Angle Limit event.  |
| <b><i>Status</i> = GetDevAngleLimitStatus()</b>             | Check if the current directional angle exceeds the deviation angle limit. When the current direction angle exceeds the limit, this function returns 1 in <i>Status</i> , otherwise 0.   |
| <b>Measurement Refresh and Calibration Commands</b>         |   |

|   |  |
|---|--|
| <b>SetRefreshFreq(<i>Rate</i>)</b>        | <p>Set refresh rate of the azimuth angle measurement by the value of <b><i>Rate</i></b>. The default value is 0.</p> <p>Six refresh rates are available:</p> <p>0 → Refresh every 20 ms (50Hz)</p> <p>1 → Refresh every 50 ms (20Hz)</p> <p>2 → Refresh every 100 ms (10Hz)</p> <p>3 → Refresh every 250 ms (4Hz)</p> <p>4 → Refresh every 500 ms (2Hz)</p> <p>5 → Refresh every 1000 ms (1Hz)</p> |
| <b>GetRefreshFreq(<i>Rate</i>)</b>        | <p>Return refresh rate of the azimuth angle measurement. The return value of <b><i>Rate</i></b> ranges from 0 to 5. See command <b>SetRefreshFreq()</b> for details.</p>   |
| <b><i>Status</i> = GetRefreshStatus()</b> | <p>Check the refresh status. When the azimuth angle measurement is refreshed, it returns 1 in <b><i>Status</i></b>. After checking the status, it returns 0 in <b><i>Status</i></b>.</p>   |
| <b>SetDimension(<i>Dimen</i>)</b>         | <p>Assign the 2D plane by setting <b><i>Dimen</i></b> with value 0, 1 or 2. The default value is 0.</p> <p>0 → x-y plane, 0 degree on x-axis and 90 on y-axis.</p> <p>1 → x-z plane, 0 degree on x-axis and 90 on z-axis.</p> <p>2 → y-z plane, 0 degree on y-axis and 90 on z-axis.</p>   |
| <b>GetDimension(<i>Dimen</i>)</b>         | <p>Read the 2D plane setting value ranging 0~2 and store in <b><i>Dimen</i></b>. See <b>SetDimension()</b> command for details.</p>  |
| <b>EnableRefreshEvent()</b>               | <p>Enable the event to trigger when the azimuth measurement is refreshed</p>   |
| <b>DisableRefreshEvent()</b>              | <p>Disable the azimuth measurement refresh event.</p>  |
| <b>ABConvert(<i>Ang</i>, <i>Bin</i>)</b>  | <p>Convert the <b><i>Ang</i></b> in degrees to <b><i>Bin</i></b> in binary radians (Brads). A full circle of 360 degrees is equal to 256 Brads. More than one circle conversion is accepted. <b><i>Ang</i></b> ranges from 0~359 and the return value of <b><i>Bin</i></b> from 0~255.</p>   |
| <b>BAConvert(<i>Bin</i>, <i>Ang</i>)</b>  | <p>Convert the <b><i>Bin</i></b> in binary radians (Brads) to <b><i>Ang</i></b> in degrees. A full circle of 360 degrees is equal to 256 Brads. <b><i>Bin</i></b> ranges from 0~255 and the return value of <b><i>Ang</i></b> from 0~359.</p>  |
| <b>Calibration(<i>Time</i>)</b>           | <p>Set the calibration duration with the value of <b><i>Time</i></b>. Five different calibration duration are available:</p> <p>0 → keep calibrating until the button is pressed again.</p> <p>1 → Calibrate for 10 seconds.</p>   |



|                                 |   |
|---------------------------------|---|
|                                 | 2 → Calibrate for 20 seconds.<br>3 → Calibrate for 30 seconds.<br>4 → Calibrate for 60 seconds. |
| <b>RestoreDefaultCalValue()</b> | Restore to the default calibration values.  |

### Events Provided by the Module

| Event                       | Description   |
|-----------------------------|---|
| <b>FieldRefreshEvent()</b>  | After <b>EnableRefreshEvent()</b> is executed, this event is activated when the module refreshes the current angle measurement. The refresh time varies by the setting of <b>SetRefreshFreq()</b> .   |
| <b>DevAngleLimitEvent()</b> | After <b>EnableDevAngleLimitEvent()</b> is executed, this event is activated when the deviation of the current directional angle is beyond the setting of <b>SetDevAngleLimit()</b> . The base direction is set by <b>SetTargetAngle()</b> command. |
| <b>CalEndEvent()</b>        | This event is activated automatically when Compass B calibration is completed. Related Enable/Disable Event commands are not required.  |

































### Example Program

```
Peripheral myCompass as CompassB @ 0 'set module ID to 0
Dim iFX As Integer 'variable for x-axis magnetic field intensity
Dim iFY As Integer 'variable for y-axis magnetic field intensity
Dim wAngle As Word 'variable for angle measurement
Sub Main() 'main program
    myCompass.SetRefreshFreq(4) 'set the refresh rate
    myCompass.SetDimension(0) 'set x-y plane
    myCompass.EnableRefreshEvent() 'enable event
    Do:Loop 'infinite loop
End Sub

Event myCompass.FieldRefreshEvent() 'measurement refresh event
    myCompass.GetAngle(wAngle) 'get the azimuth
    myCompass.GetXField(iFX) 'get x-axis intensity
    myCompass.GetYField(iFY) 'get y-axis intensity
    Debug CSRXY(1, 5), "Azimuth = ", %DEC3 wAngle, CR
    Debug CSRXY(1, 6), "X-axis Intensity = ", %DEC6 iFX, CR
    Debug CSRXY(1, 7), "Y-axis Intensity = ", %DEC6 iFY, CR
End Event
```

## Appendix A --- Module ID Setting

Each module needs to be assigned its own ID not conflicting with other modules on the same CmdBUS™. This assigned ID will be used in the program so the BASIC Commander® can communicate with module correctly. The table below shows the DIP switch setting for desired module ID.

|   |   |    |   |    |  |    |   |
|---|---|----|---|----|--|----|---|
| 0 |    | 8  |    | 16 |    | 24 |    |
| 1 |    | 9  |    | 17 |    | 25 |    |
| 2 |    | 10 |    | 18 |    | 26 |    |
| 3 |    | 11 |    | 19 |    | 27 |    |
| 4 |    | 12 |    | 20 |    | 28 |    |
| 5 |   | 13 |   | 21 |   | 29 |   |
| 6 |  | 14 |  | 22 |  | 30 |  |
| 7 |  | 15 |  | 23 |  | 31 |  |

## Appendix B --- I2C Format Command Table

In addition to the featured BASIC Commander<sup>®</sup> command format, I2C format is also listed here for users who would like to use Innovati smart peripheral modules in their systems. The I2C command convention is described as below:

MID: Module ID ranging from 0 to 31 indicated by the DIP switch setting on the smart peripheral module.

CID: 8-bit Command ID as shown in each command.

CS1: 8-bit Checksum defined as  $255 - (MID) \times 2 - CID$

CS2: Optional second 8-bit Checksum defined as  $255 - \text{sum of bytes in between CS1 and CS2}$ .

CS3: Optional third 8-bit Checksum defined as  $255 - MID - \text{sum of bytes in between MID and CS3}$ .

DMY: 8-bit dummy byte with value 0.

If data is wider than one byte, Little Endian is employed. All the data should be expressed in hexadecimal format. The following is an example to demonstrate how to constitute an I2C command sequence. Let's assume the Module ID is set to 2 and the command ID is 153 and the WORD-size payload data is 511. The I2C command sequence is:

MID, CID, CS1, DataL, DataH, CS2, DMY

where,

MID = 2 (or known as &H02)

CID = 153 (or known as &H99)

CS1 =  $255 - (2 \times 2) - 153 = 98$  (or known as &H62)

DataL = 255 (or known as &HFF)

DataH = 1 (or known as &H1)

CS2 =  $255 - \text{DataL} - \text{DataH} = 255$  (or known as &HFF)

DMY = 0 (or known as &H0)

The following is an example to demonstrate how to read the returned data. Let's assume a Word data type value &H1FF is returned, The I2C command sequence is:

MID, ResultL, ResultH, CS3

where,

MID = 2 (or known as &H02)

ResultL = 255 (or known as &HFF, returned by module)

ResultH = 1 (or known as &H1, returned by module)

CS3 = 255 – ResultL – ResultH = 255 (or known as &HFF, returned by module)

Note that the EVENT functions are available only under CmdBUS™ scheme collaborating with BASIC Commander® and not available for generic microcontrollers. Therefore, event-related I2C commands are not listed in the table below.

| <b>BASIC Commander® Format</b>                  | <b>I2C Format</b>   |
|---|---|
| <b>GetXField(<i>Fx</i>)</b>                     | <b>MID, 88, CS1, DMY</b><br><b>MID, <i>Fx_L</i>, <i>Fx_H</i>, CS3</b>                     |
| <b>GetYField(<i>FyY</i>)</b>                    | <b>MID, 89, CS1, DMY</b><br><b>MID, <i>Fy_L</i>, <i>Fy_H</i>, CS3</b>                     |
| <b>GetZField(<i>Fz</i>)</b>                     | <b>MID, 130, CS1, DMY</b><br><b>MID, <i>Fz_L</i>, <i>Fz_H</i>, CS3</b>                    |
| <b>GetAngle(<i>Ang</i>)</b>                     | <b>MID, 91, CS1, DMY</b><br><b>MID, <i>Ang_L</i>, <i>Ang_H</i>, CS3</b>                   |
| <b>GetAngle3D(<i>Ang1</i>, <i>Ang2</i>)</b>     | <b>MID, 136, CS1, DMY</b><br><b>MID, <i>Ang1_L</i>, <i>Ang1_H</i>, <i>Ang2</i>, CS3</b>   |
| <b>SetTargetAngle (<i>Ang</i>)</b>              | <b>MID, 134, CS1, <i>Ang_L</i>, <i>Ang_H</i>, CS2, DMY</b>                                |
| <b>GetTargetAngle (<i>Ang</i>)</b>              | <b>MID, 135, CS1, DMY</b><br><b>MID, <i>Ang_L</i>, <i>Ang_H</i>, CS3</b>                  |
| <b>GetDevAngle(<i>Ang</i>)</b>                  | <b>MID, 96, CS1, DMY</b><br><b>MID, <i>Ang_L</i>, <i>Ang_H</i>, CS3</b>                   |
| <b>SetDevAngleLimit(<i>Ang</i>)</b>             | <b>MID, 97, CS1, <i>Ang</i>, CS2, DMY</b>   |
| <b>GetDevAngleLimit(<i>Ang</i>)</b>             | <b>MID, 98, CS1, DMY</b><br><b>MID, <i>Ang</i>, CS3</b>                                   |
| <b><i>Status</i> = GetDevAngleLimitStatus()</b> | <b>MID, 103, CS1, DMY</b><br><b>MID, <i>Status</i>, CS3</b>                               |
| <b>SetRefreshFreq(<i>Rate</i>)</b>              | <b>MID, 104, CS1, <i>Rate</i>, CS2, DMY</b>   |
| <b>GetRefreshFreq(<i>Rate</i>)</b>              | <b>MID, 105, CS1, DMY</b><br><b>MID, <i>Rate</i>, CS3</b>                                 |
| <b><i>Status</i> = GetRefreshStatus()</b>       | <b>MID, 106, CS1, DMY</b><br><b>MID, <i>Status</i>, CS3</b>                               |
| <b>SetDimension(<i>Dimen</i>)</b>               | <b>MID, 131, CS1, <i>Dimen</i>, CS2</b>   |
| <b>GetDimension(<i>Dimen</i>)</b>               | <b>MID, 132, CS1, DMY</b><br><b>MID, <i>Dimen</i>, CS3</b>                                |
| <b>ABConvert(<i>Ang</i>, <i>Bin</i>)</b>        | <b>MID, 109, CS1, <i>Ang_L</i>, <i>Ang_H</i>, CS2, DMY</b><br><b>MID, <i>Bin</i>, CS3</b> |

|                                   |  |
|-----------------------------------|--|
| <b>BAConvert(<i>Bin, Ang</i>)</b> | <b>MID, 110, CS1, <i>Bin</i>, CS2, DMY</b><br><b>MID, <i>Ang_L, Ang_H</i>, CS3</b> |
| <b>Calibration(<i>Time</i>)</b>   | <b>MID, 121, CS1, <i>Time</i>, CS2, DMY</b>  |
| <b>RestoreDefaultCalValue()</b>   | <b>MID, 125, CS1, DMY</b>  |

Note: I2C is a Registered Trademark of Philips Semiconductors.

## Appendix C --- C language Format for Ozone™ Board

In addition to the featured BASIC Commander® command format, C language format is also available for Innovati's 8-bit Ozone™ controller board.

Note that the EVENT functions are available only under CmdBUS™ scheme collaborating with BASIC Commander®. Therefore, event-related commands are not listed in the table below.

| BASIC Commander® Format                         | C language Format for Ozone™   |
|---|--|
| <b>GetXField(<i>Fx</i>)</b>                     | <b>void GetXField(long <i>Fx</i>)</b>  |
| <b>GetYField(<i>Fy</i>)</b>                     | <b>void GetYField(long <i>Fy</i>)</b>  |
| <b>GetZField(<i>Fz</i>)</b>                     | <b>void GetZField(long <i>Fz</i>)</b>  |
| <b>GetAngle(<i>Ang</i>)</b>                     | <b>void GetAngle(unsigned long <i>Ang</i>)</b>                               |
| <b>GetAngle3D(<i>Ang1</i>, <i>Ang2</i>)</b>     | <b>void GetDevAngle(unsigned long <i>Ang1</i>, unsigned int <i>Ang2</i>)</b> |
| <b>SetTargetAngle(<i>Ang</i>)</b>               | <b>void SetTargetAngle(unsigned long <i>Ang</i>)</b>                         |
| <b>GetTargetAngle(<i>Ang</i>)</b>               | <b>void GetTargetAngle(unsigned long <i>Ang</i>)</b>                         |
| <b>GetDevAngle(<i>Ang</i>)</b>                  | <b>void GetDevAngle(long <i>Ang</i>)</b>                                     |
| <b>SetDevAngleLimit(<i>Ang</i>)</b>             | <b>void SetDevAngle(unsigned int <i>Ang</i>)</b>                             |
| <b>GetDevAngleLimit(<i>Ang</i>)</b>             | <b>void GetDevAngle(unsigned int <i>Ang</i>)</b>                             |
| <b><i>Status</i> = GetDevAngleLimitStatus()</b> | <b>boolean GetDevAngleLimitStatus(void)</b>                                  |
| <b>SetRefreshFreq(<i>Rate</i>)</b>              | <b>void SetRefreshFreq(unsigned int <i>Rate</i>)</b>                         |
| <b>GetRefreshFreq(<i>Rate</i>)</b>              | <b>void GetRefreshFreq(unsigned int <i>Rate</i>)</b>                         |
| <b><i>Status</i> = GetRefreshStatus()</b>       | <b>boolean GetFreshStatus(void)</b>  |
| <b>SetDimension(<i>Dimen</i>)</b>               | <b>void SetDimension(unsigned int <i>Dimen</i>)</b>                          |
| <b>GetDimension(<i>Dimen</i>)</b>               | <b>void SetDimension(unsigned int <i>Dimen</i>)</b>                          |
| <b>ABConvert(<i>Ang</i>, <i>Bin</i>)</b>        | <b>void ABConvert(unsigned long <i>Ang</i>, unsigned int <i>Bin</i>)</b>     |
| <b>BAConvert(<i>Bin</i>, <i>Ang</i>)</b>        | <b>void BAConvert(unsigned int <i>Bin</i>, unsigned long <i>Ang</i>)</b>     |
| <b>Calibration(<i>Time</i>)</b>                 | <b>void Calibration(unsigned int <i>Time</i>)</b>                            |
| <b>RestoreDefaultCalValue()</b>                 | <b>void RestoreDefaultCalValue(void)</b>                                     |

### Sample Program

```
#include <ozone.h>
```

```
CompassB myCompass(0); //set module ID to 0
```

```

unsigned int angle;           //variable for angle measurement
int fx;                       //variable for x-axis magnetic field intensity
int fy;                       //variable for y-axis magnetic field intensity

void setup()
{
    Serial.begin(115200);     //serial communication Baud rate
}
void loop()                   //infinite loop
{
    myCompass.GetAngle(angle); //get the azimuth
    Serial.print("Azimuth = "); //display
    Serial.println(angle);     //
    myCompass.GetXField(fx);   //get x-axis intensity
    Serial.print("X-axis Intensity = "); //display
    Serial.println(fx);       //
    myCompass. GetYField(fy); //get y-axis intensity
    Serial.print("Y-axis Intensity = "); //display
    Serial.println(fy);      //
    delay(200);              //pause 200ms
}

```

## Appendix D --- C language Format for Arminno™

In addition to the featured BASIC Commander® command format, C language format is also available for Innovati's 32-bit Arminno™ controller board.

Note that the EVENT functions are available only under CmdBUS™ scheme collaborating with BASIC Commander®. Therefore, event-related commands are not listed in the table below.

| BASIC Commander® Format                         | C language Format for Arminno™  |
|---|---|
| <b>GetXField(<i>Fx</i>)</b>                     | <b>void GetXField(short <i>Fx</i>)</b>  |
| <b>GetYField(<i>Fy</i>)</b>                     | <b>void GetYField(short <i>Fy</i>)</b>  |
| <b>GetZField(<i>Fz</i>)</b>                     | <b>void GetZField(short <i>Fz</i>)</b>  |
| <b>GetAngle(<i>Ang</i>)</b>                     | <b>void GetAngle(unsigned short <i>Ang</i>)</b>                               |
| <b>GetAngle3D(<i>Ang1</i>, <i>Ang2</i>)</b>     | <b>void GetAngle3D(unsigned short <i>Ang1</i>, unsigned char <i>Ang2</i>)</b> |
| <b>SetTargetAngle(<i>Ang</i>)</b>               | <b>void SetTargetAngle(unsigned short <i>Ang</i>)</b>                         |
| <b>GetTargetAngle(<i>Ang</i>)</b>               | <b>void GetTargetAngle(unsigned short <i>Ang</i>)</b>                         |
| <b>GetDevAngle(<i>Ang</i>)</b>                  | <b>void GetDevAngle(short <i>Ang</i>)</b>                                     |
| <b>SetDevAngleLimit(<i>Ang</i>)</b>             | <b>void SetDevAngle(unsigned char <i>Ang</i>)</b>                             |
| <b>GetDevAngleLimit(<i>Ang</i>)</b>             | <b>void GetDevAngleLimit(unsigned char <i>Ang</i>)</b>                        |
| <b><i>Status</i> = GetDevAngleLimitStatus()</b> | <b>boolean GetDevAngleLimitStatus(void)</b>                                   |
| <b>SetRefreshFreq(<i>Rate</i>)</b>              | <b>void SetRefreshFreq(unsigned char <i>Rate</i>)</b>                         |
| <b>GetRefreshFreq(<i>Rate</i>)</b>              | <b>void SetRefreshFreq(unsigned char <i>Rate</i>)</b>                         |
| <b><i>Status</i> = GetRefreshStatus()</b>       | <b>boolean GetFreshStatus(void)</b>   |
| <b>SetDimension(<i>Dimen</i>)</b>               | <b>void SetDimension(unsigned char <i>Dimen</i>)</b>                          |
| <b>GetDimension(<i>Dimen</i>)</b>               | <b>void GetDimension(unsigned char <i>Dimen</i>)</b>                          |
| <b>ABConvert(<i>Ang</i>, <i>Bin</i>)</b>        | <b>void ABConvert(unsigned short <i>Ang</i>, unsigned char <i>Bin</i>)</b>    |
| <b>BAConvert(<i>Bin</i>, <i>Ang</i>)</b>        | <b>void BAConvert(unsigned char <i>Bin</i>, unsigned short <i>Ang</i>)</b>    |
| <b>Calibration(<i>Time</i>)</b>                 | <b>void Calibration(unsigned char <i>Time</i>)</b>                            |
| <b>RestoreDefaultCalValue()</b>                 | <b>void RestoreDefaultCalValue(void)</b>                                      |

### Sample Program

```
#include "arminno.h"
CompassB myCompass (0); //set module ID to 0
```



```

unsigned short angle;           //variable for angle measurement
short fx;                       //variable for x-axis magnetic field intensity
short fy;                       //variable for y-axis magnetic field intensity

int main(void)
{
    while(1)                    //infinite loop
    {
        myCompass.GetAngle(angle); //variable for angle measurement
        printf("Azimuth = %d\r\n", angle); //display
        myCompass. GetXField(fx); //variable for x-axis magnetic field intensity
        printf("X-axis Intensity = %d\r\n", fx); //display
        myCompass.GetGetYField(fy); //variable for y-axis magnetic field intensity
        printf("Y-axis Intensity = %d\r\n", fy); //display
        Pause(200);             //pause 200ms
    }
}

```