TI-Innovator™ Technology with Python Guidebook

Learn more about TI Technology through the online help at education.ti.com/eguide.

Important Information

Except as otherwise expressly stated in the License that accompanies a program, Texas Instruments makes no warranty, either express or implied, including but not limited to any implied warranties of merchantability and fitness for a particular purpose, regarding any programs or book materials and makes such materials available solely on an "as-is" basis. In no event shall Texas Instruments be liable to anyone for special, collateral, incidental, or consequential damages in connection with or arising out of the purchase or use of these materials, and the sole and exclusive liability of Texas Instruments, regardless of the form of action, shall not exceed the amount set forth in the license for the program. Moreover, Texas Instruments shall not be liable for any claim of any kind whatsoever against the use of these materials by any other party.

Learning More with the TI-Innovator™ Technology with Python eGuide

Parts of this document refer you to the TI-Innovator™ Technology with Python eGuide for more details. The eGuide is a Web-based source of TI-Innovator™ information, including:

- Programming with the TI CE Family of Graphing Calculators and TI-Nspire™ Technology, including sample programs.
- Available TI-RGB Array and its commands.
- Available TI-Innovator™ Rover and its commands.
- Link to update the TI-Innovator™ Sketch software.
- Free classroom activities for TI-Innovator™ Hub.

Actual products may vary slightly from provided images.

Contents

Python Interfaces **1**

TI-Innovator™ Hub with Python Commands **2**

Built-in **9**

Built-in Devices 9
Color 10
color.rgb() 10
color.blink() 11
color.off() 11
Light 12
light.on() 12
light.off() 12
light.blink() 13
Sound 14
sound.tone() 14
sound.note() 15
Brightness sensor 16
brightness.measurement() 16
brightness.range() 17

Input **18**

Input Devices 18
DHT (Digital Humidity & Temperature) 18
var = dht("port") 18
var.temp_measurement() 19
var.humidity_measurement() 19
var.t_h_measurements() 20
var=dht("port","opt") 20
var.temp_measurement() 21
var.humidity_measurement() 21
var.t_h_measurements() 22
Ranger 23
var=ranger("port") 23
var.measurement() 23
LightLevel (Light Sensor) 25
var=light_level("port") 25
var.measurement() 25
var.range(min,max) 26
Moisture 27
var=moisture("port") 27

var.measurement() 27
var.range(min,max) 28
Magnetic 29
var=magnetic("port") 29
var=magnetic_close() 30
var.measurement() 30
var.trigger(level) 31
Vernier (used with TI-SensorLink) 32
var=vernier("port","type") 32
var.measurement() 33
var.calibrate(a,b) 34
var.calibrate(a,b,c,r) 34
Analog In 35
var=analog_in("port") 35
var.measurement() 35
var.range(min,max) 36
Digital In/Out 37
var=digital("port") 37
var.measurement() 38
var.set(val) 38
var.off() 39
var.on() 39
Potentiometer 39
var=potentiometer("port") 40
var.measurement() 40
var.range(min,max) 41
Thermistor 42
var=thermistor("port") 42
var.measurement() 43
var.calibrate(c1,c2,c3,r) 43
Loudness 44
var=loudness("port") 44
var.measurement() 44
var.range(min,max) 45
Color Input 46
var=color_input("bb_port") 47
var.color_number() 47
var.red() 48
var.green() 48
var.blue() 49
var.gray() 49
BB Port (Breadboard Port) 50
var=bb_port("mask") 50

var.read_port() 51
var.read_port(mask) 52
var.write_port(value) 52
var.write_port(value, mask) 52
Hub Time 54
var=hub_time() 54
var.measurement() 54
var.reset_time() 55
TI-RGB Array 56
var=rgb_array() 57
var.set(led_position, red, green, blue) 58
var.set_all(red, green, blue) 58
var.all_off() 58
var.pattern(value) 59
var.measurement() 59

Output 60

Output Devices 60
LED 61
var=led("port") 61
var.off() 61
var.on() 62
var.blink(freq,time) 62
RGB 64
var=rgb("port") 64
var.rgb(r,g,b) 64
var.blink(freq,time) 65
var.off() 66
Speaker 67
var=speaker("port") 67
var.tone(freq,time) 68
var.note(“note”,time) 68
Power 70
var=power("port") 70
var.set(value) 71
var.off() 72
var.on() 72
Continuous Servo 73
var=continuous_servo("OUT 3") 73
var.set_cw(speed,time) 74
var.set_ccw(speed,time) 74
var.stop() 75
Analog Out 76

var=analog_out ("port") 76
var.set(value) 76
var.off() 77
var.on() 78
Vibration Motor 79
var=vibration_motor("OUT 3") 79
var.set(value) 80
var.off() 80
var.on() 81
Relay 82
var=relay("OUT 3") 82
var.off() 83
var.on() 83
Servo 85
var=servo("OUT 3") 85
var.set_position(pos) 86
var.zero() 86
Squarewave 88
var=squarewave("port") 88
var.set(freq,duty,time) 89
var.off() 89
Digital in|out 91
var=digital("port") 91
var.measurement() 92
var.set(value) 93
var.off() 93
var.on() 93

Python ti_rover Module 95

[Fns...]>Modul: ti_rover module 95
Drive Menu 98
CE menus 98
TI-Nspire™ CX II menus 98
Drive Commands 99
rv.forward 99
rv.forward_time 100
rv.backward 101
rv.backward_time 102
rv.left 103
rv.right 104
rv.stop 105
rv.resume() 105
rv.stay 107

rv.to_xy 107
rv.to_polar 108
rv.to_angle 108
Inputs Menu 109
CE menus 109
TI-Nspire™ CX II menus 109
Sensor Commands 109
rv.ranger_measurement() 109
rv.color_measurement(): 110
rv.red_measurement() 111
rv.green_measurement() 111
rv.blue_measurement(): 112
rv.gray_measurement(): 112
rv.encoders_gyro_measurement() 113
rv.gyro_measurement 113
Outputs Menu 114
CE menus 114
TI-Nspire™ CX II menus 114
Output Commands 114
rv.color_rgb() 114
rv.color_blink() 115
rv.color_off() 115
rv.motor_left() 116
rv.motor_right() 116
rv.motors() 117
Path Menu 118
CE menus 118
TI-Nspire™ CX II menus 118
Path Commands 119
rv.waypoint_xythdrn() 119
rv.waypoint_prev() 120
rv.waypoint_eta() 120
rv.path_done() 121
rv.pathlist_x() 121
rv.pathlist_y() 122
rv.pathlist_time() 123
rv.pathlist_heading() 123
rv.pathlist_distance() 124
rv.pathlist_revs() 124
rv.pathlist_cmdnum() 125
rv.waypoint_x() 126
rv.waypoint_y() 126
rv.waypoint_time() 127

rv.waypoint_heading () 127
rv.waypoint_distance() 128
rv.waypoint_revs() 128
Settings Menu 129
CE menus 129
TI-Nspire™ CX II menus 129
Settings 130
units/s 130
ms/s 130
revs/s 131
units 131
m 132
revs 132
degrees 133
radians 133
gradians 134
clockwise 134
counter-clockwise 135
Commands Menu 136
CE menus 136
TI-Nspire™ CX II menus 136
Commands 137
sleep(seconds) 137
disp_at(row,col,"text") 138
disp_clr() 139
disp_wait() 140
disp_cursor() 141
while not escape(): clear 142
rv.wait_until_done() 143
while not path_done() 144
rv.position() 144
rv.grid_origin() 145
rv.grid_m_unit() 145
rv.path_clear() 146
rv.zero_gyro() 146

TI-SensorLink Adapter Using VERNIER Commands with Python Programs 147

Stainless Steel Temperature Probe with Python 147
pH Sensor with Python 147
Gas Pressure Sensor with Python 148
Dual-Range Force Sensor with Python 148
vernier() interface 148
CE Family 148

TI-Nspire CX II 150
vernier() 152
.measurement() 152
.calibrate(a,b) 153
.calibrate(a,b,c,r) 154

TI-RGB Array Commands Using Python Programs 155

rgb_array() 156
rgb_array("lamp") 156
CE Family 157
TI-Nspire CX II 157
.set(n,r,g,b) 158
.set_all(r,g,b) 158
all_off() 159
.pattern() 159
.measurement() 160

Appendix 161

Python ti_hub Module 162

[Fns...]>Modul: ti_hub module 162
connect() 164
disconnect() 165
set() 166
read() 167
calibrate() 168
range() 169
version() 170
begin() 171
about() 172
isti() 173
what() 174
who() 175
last_error() 176
sleep() 177

Floating Point Values and Digits after Decimal Point 178

Floating Point Return Values 178

Python Interfaces

These pages describe the calculator based Python modules that provide support for the TI-Innovator™ ecosystem on the following models:

- ti_hub_module
- ti_rover module

Python is available on these TI Graphing Calculator products:

- TI-83 Premium CE *Edition Python*
- TI-84 Plus CE Python
- TI-84 Plus CE-T Python Edition

Python is available on these TI-Nspire™ products:

Handhelds

- TI-Nspire™ CX II
- TI-Nspire™ CX II CAS
- TI-Nspire™ CX II-T
- TI-Nspire™ CX II-T CAS
- TI-Nspire™ CX II-C
- TI-Nspire™ CX II-C CAS

Desktop Software

- TI-Nspire™ CX Premium Teacher Software
- TI-Nspire™ CX CAS Premium Teacher Software
- TI-Nspire™ CX Student Software
- TI-Nspire™ CX CAS Student Software

TI-Innovator™ Hub with Python Commands

Use the menus for the TI Hub module (“ti_hub”) to create or edit a program. They can save you time building commands and help you with correct command spelling and syntax.

When you see "**Python Code Sample**" in a command table, this "**Python Code Sample**" may be copied and pasted *as is* to send to your graphing calculator to use in your calculations.

Example:

Python Code Sample:	`light.on()`
	`light.off()`
	`sound.tone(440,4)`
	`b = brightness.measurement()`
	`print(b)`

Note: To build a command from the Hub menu, you need to know:

- The unique name of the component that you are addressing, such as "SOUND" for the on-board speaker.
- The command parameters that apply to the component, such as sound frequency and duration. Some parameters are optional, and you might need to know the value range of a parameter.

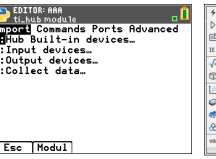
NOTE: The Python commands are identical on the CE calculators and TI-Nspire™ CX II. However, on the CE products, each of the sensor and external devices has its own module.

Screenshots are provided for reference. **Note:** Actual menus may vary slightly from provided images.

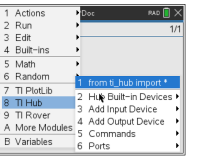
TI-HUB Menus

- Hub Built-in devices

CE Calculators



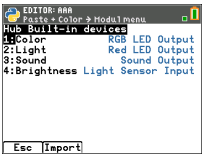
TI-Nspire™ CX



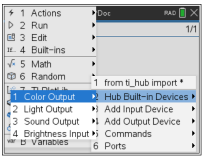
Built-in Menus

- Color
- Light
- Sound
- Brightness

CE Calculators



TI-Nspire™ CX

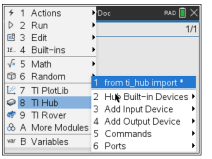


Input Menus

- Hub Input devices

CE Calculators

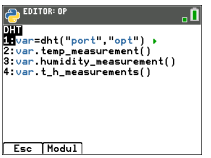
TI-Nspire™ CX



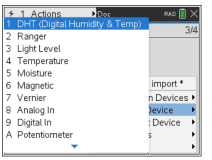
Input DHT

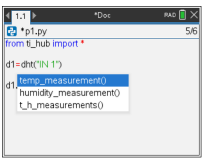
- var = dht("port", "opt")
- var.temp_measurement()
- var.humidity_measurement()
- var.t_h_measurement()

CE Calculators



TI-Nspire™ CX





Input Ranger

- var=ranger("port")
- var.measurement()

Input LightLevel

- var=ranger("port")
- var.measurement()
- var.range(min,max)

Input Temp

- var=temperature("port")

- var.measurement()

Input Moisture

- var=moisture("port")
- var.measurement()
- var.range(min,max)

Input Magnetic

- var=magnetic("port")
- var=magnetic_close()
- var.measurement()
- var.trigger(level)

Input Vernier

- var=vernier("port","type")
- var.measurement()
- var.calibrate(a,b)
- var.calibrate(a,b,c,r)

Input Analog In

- var=analogin("port")
- var.measurement()
- var.range(min,max)

Input Digital In/Out

- var=digital("port")
- var.measurement()
- var.set(val)
- var.off()
- var.on()

Input Potentiometer

- var=potentiometer("port")
- var.measurement()

- var.range(min,max)

Input Thermistor

- var=thermistor("port")
- var.measurement()
- var.calibrate(c1,c2,c3,r)

Input Loudness

- var=loudness("port")
- var.measurement()
- var.range(min,max)

Input Color Input

- var=color_input("bb_port")
- var.color_number()
- var.red()
- var.green()
- var.blue()
- var.gray()

Input BB Port (Breadboard Port)

- var=bb_port("mask")
- var.read_port()
- var.read_port(mask)
- var.write_port(value)
- var.write_port(value, mask)

Input Hub Time

- var=hub_time()
- var.measurement()
- var.reset_time()

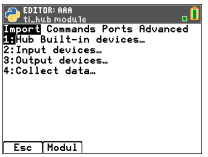
Input TI-RGB Array

- var=rgb_array()

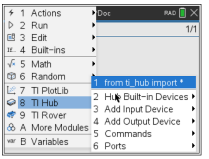
- var.set(led_position, red, green, blue
- var.set_all(red, green, blue
- var.all_off()
- var.pattern(value)
- var.measurement()

Output Menus

CE Calculators



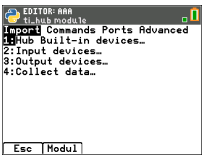
TI-Nspire™ CX



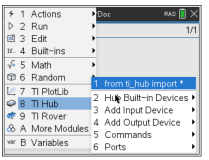
- Hub Output devices

Output LED

CE Calculators



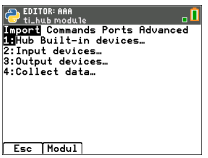
TI-Nspire™ CX



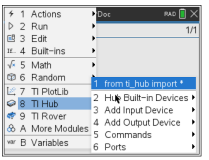
- var=led("port")
- var.off()
- var.on()
- var.blink(freq,time)

Output RGB

CE Calculators



TI-Nspire™ CX

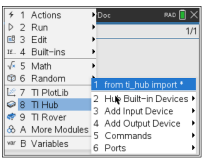


- var=rgb("port")
- var.rgb(r,g,b)
- var.blink(freq,time)
- var.off()

Output Speaker

CE Calculators

TI-Nspire™ CX



- var=speaker("port")
- var.tone(freq,time)
- var.note(“note”,time))

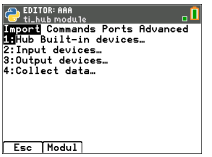
Output Power

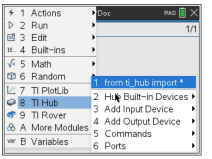
CE Calculators

TI-Nspire™ CX

- var=power("port")
- var.set(value)

- var.off()
- var.on()

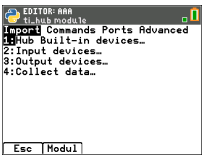


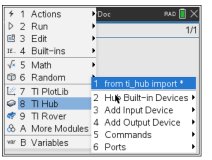


Output Continuous Servo

CE Calculators **TI-Nspire™ CX**

- var=continuous_servo("OUT 3")
- var.set_cw(speed,time)
- var.set_ccw(speed,time)
- var.stop()

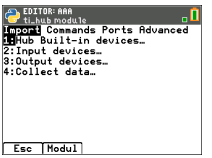


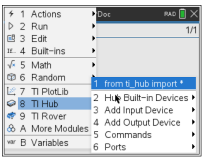


Output Analog Out

CE Calculators **TI-Nspire™ CX**

- var=analog_out ("port")
- var.set(value)
- var.off()
- var.on()

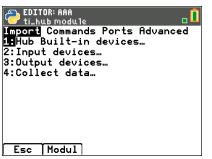


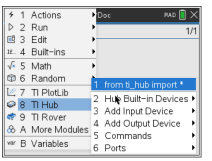


Output Vibration Motor

CE Calculators **TI-Nspire™ CX**

- var=vibration_motor("OUT 3")
- var.set(value)
- var.off()
- var.on()

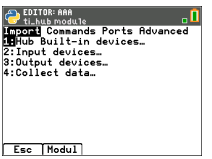


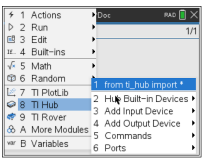


Output Relay

CE Calculators **TI-Nspire™ CX**

- var=relay("OUT 3")
- var.off()
- var.on()

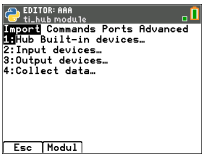


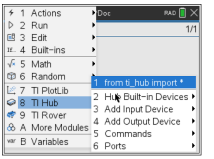


Output Servo

CE Calculators **TI-Nspire™ CX**

- var=servo("OUT 3")
- var.set_position(pos)
- var.zero()



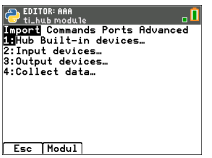


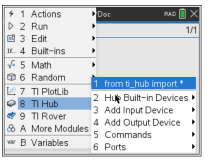
Output Squarewave

CE Calculators

TI-Nspire™ CX

- var=squarewave("port")
- var.set(freq,duty,time)
- var.off()



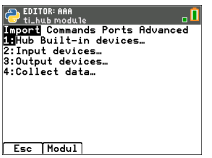


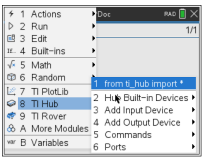
Output Digital in|out

CE Calculators

TI-Nspire™ CX

- var=digital("port")
- var.measurement()
- var.set(value)
- var.off()
- var.on()





Built-in

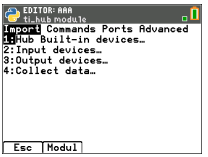
Built-in Devices

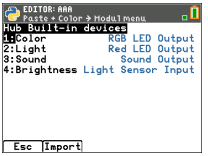
The TI-Innovator™ Hub has 4 built-in devices – an LED, an RGB LED, a speaker and a brightness sensor.

Additional devices can be attached to the Hub through the IN and OUT ports.

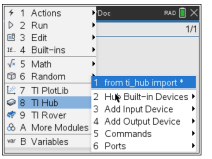
Menu

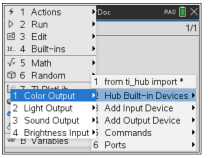
CE Calculators

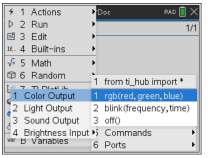


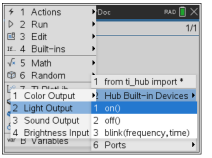


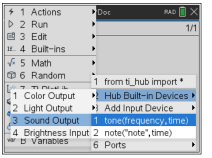
TI-Nspire™ CX II

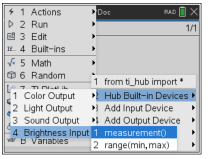








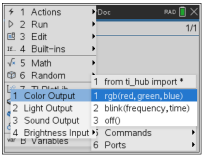




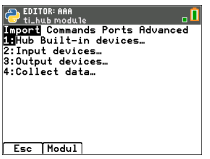
Color

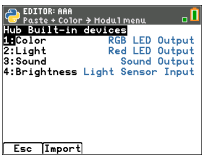
This object controls the on-board RGB LED of the TI-Innovator™ Hub.

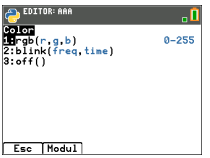
TI-Nspire CX II: `from ti_hub import *`



CE products: `import color`







color.rgb()

Command:	color.rgb()
Command Syntax:	**color.rgb(red, green, blue)**
Range:	red = [0, 255] green = [0, 255] blue = [0, 255]
Describe:	Controls on-board **RGB LED**. Example: `color.rgb(255,0,255)` – sets the color of the RGB LED to purple (red and blue) **Note:** Setting all three to 255 will make the RGB LED turn bright white. All 0s will turn it OFF
Result:	Sets the on-board RGB LED to the color specified by the values.

Command:	color.rgb()
Type or Addressable Component:	Control

color.blink()

Command:	color.blink()
Command Syntax:	**color.blink(frequency, time)**
Range:	frequency = [0.1, 20] time = [0.1, 100]

Describe:	Set blink frequency and duration for on-board RGB LED.
	Example:
	`color.rgb(255,0,255) # Set LED to purple`
	`color.blink(4, 5) # Make it blink 4 times a second for 5 seconds`
	Note: The color needs to be specified before the blink function is called.

Result:	NA
Type or Addressable Component:	Control

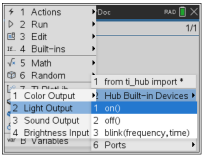
color.off()

Command:	color.off
Command Syntax:	**color.off()**
Range:	
Describe:	Turns off on-board RGB LED. This is equivalent to “color.rgb(0,0,0)”
Result:	The LED turns OFF.
Type or Addressable Component:	Control

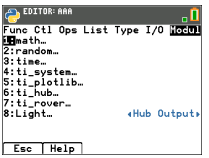
Light

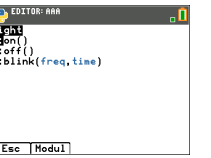
This object controls the on-board LED of the TI-Innovator™ Hub.

TI-Nspire CX II: `from ti_hub import *`



CE products: `import light`





light.on()

Command:	light.on()
Command Syntax:	**light.on()**
Range:	
Describe:	Turns on the on-board digital RED LED.
Result:	Turns on LIGHT.
Type or Addressable Component:	Control

light.off()

Command:	light.off()
Command Syntax:	**light.off**
Range:	
Describe:	Turns off the on-board digital RED LED.

Command:	light.off()
Result:	Turns off LIGHT.
Type or Addressable Component:	Control

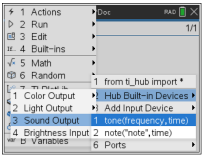
light.blink()

Command:	light.blink()
Command Syntax:	**light.blink(frequency, seconds)**
Range:	Frequency: 0.1 – 20 Hz Time: 0.1 – 100 s
Describe:	Set blink frequency and duration for on-board digital LED. Example: `light.blink(2, 5) # Blink the LED 2 times a second for 5 seconds`
Result:	
Type or Addressable Component:	Control

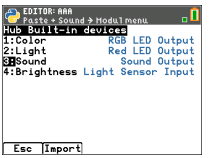
Sound

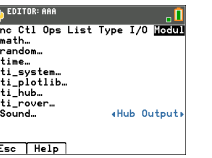
This object controls the on-board speaker of the TI-Innovator™ Hub.

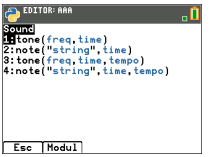
TI-Nspire CX II: `from ti_hub import *`



CE products: `import sound`







sound.tone()

Command:	sound.tone()
Command Syntax:	**sound.tone(frequency, [time])**
Range:	frequency = [0, 8000] time = [0.1 100]
Describe:	**SOUND** is the on-board speaker and can generate a sound with a specified frequency. If time not specified, sound will play for 1 second default.
Result:	Play tone through on-board speaker.
Type or Addressable Component:	Control

sound.note()

Command:	sound.note()
Command Syntax:	**sound.note("note", [time])**
Range:	note: This is a string that specifies the note to be played. Note names are: "C", "CS", "D", "DS", "E", "F", "FS", "G", "GS", "A", "AS", and "B". And the octave numbers range from 1 to 9 (inclusive)) time = [0.1 100]
Describe:	**SOUND** is the on-board speaker and can generate a sound with a specified note. If time not specified, sound will play for 1 second default. Example: sound.note("C5", 2)
Result:	Play note through on-board speaker.
Type or Addressable Component:	Control

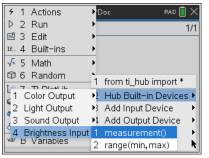
Brightness sensor

This object is the interface to the brightness sensor on the TI-Innovator™ Hub.

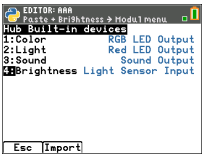
It allows the program to read the value from the brightness sensor.

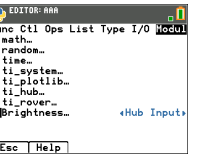
.

TI-Nspire CX II: `from ti_hub import *`



CE products: `import brightns`





brightness.measurement()

Command:	**TI-Nspire CX II: brightness.measurement()** **CE products: brightns.measurement()**
Command Syntax:	**brightness.measurement()**
Range:	0 - 100

Describe:	Returns the current internal reading from the on-board ambient light sensor.
	Example TI-Nspire CX II program:
	`from ti_hub import *`
	`from time import *`
	`while get_key() != "esc":`
	`  b = brightness.measurement()`

Command:	TI-Nspire CX II: brightness.measurement() CE products: brightns.measurement()
	••print(b) ••sleep(1)
Result:	Read on-board light sensor level.
Type or Addressable Component:	Control

brightness.range()

Command:	TI-Nspire CX II: brightness.range() CE products: brightns.range() **Advanced user**
Command Syntax:	**brightness.range([min, max])**
Range:	
Describe:	Changes/Sets the mapping of ADC input values from the ADC 0-16383 range to a user-selected range. The resulting sensor reading is mapped to this and a floating point result is returned. By default, the on-board BRIGHTNESS sensor is ranged to a 0-100 range.
Result:	Set mapping for on-board brightness/light sensor.
Type or Addressable Component:	Sensor

Input

Input Devices

The Python interface to the sensors requires the creation of an object that represents the sensor. This includes specifying the port on the TI-Innovator™ Hub to which the sensor is attached.

The sensor readings can then be read through the ".measurement()" function for each sensor.

Here are the input devices and their associated functions.

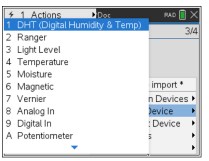
DHT (Digital Humidity & Temperature)

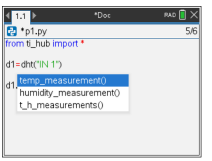
DHT (Digital Humidity & Temp) supports the interface of a digital humidity and temperature sensor.

Add Input Device

Item	Description
DHT	Returns a list consisting of the current temperature, humidity, type of sensor, and last cached read status.

TI-Nspire CX II: `from ti_hub import *`





The function to create the object is pasted from the menu.

var = dht("port")

Command:	var = dht("port")
Command Syntax:	**var = dht("port")**
Range:	
Describe:	The available functions for the objects are displayed by adding a period "." after the variable name for the sensor object.
Result:	

Command:	var = dht("port")
Type or Addressable Component:	Control

var.temp_measurement()

Command:	var.temp_measurement()
Command Syntax:	var.temp_measurement()

Python Code Sample:	TI Nspire CX II:
	`from ti_hub import *`
	`d1 = dht("IN 1")`
	`t = d1.temp_measurement()`
	`print("Temp is: ", t)`

	CE products:
	`from dht import *`
	`d1 = dht("IN 1", "DHT11")`
	`t = d1.temp_measurement()`
	`print("Temp is: ", t)`

Range:	
Describe:	Function to return the temperature value in degrees Celsius.
Result:	NA
Type or Addressable Component:	Control

var.humidity_measurement()

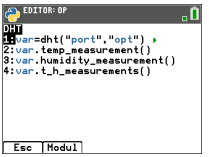
Command:	var.humidity_measurement()
Command Syntax:	var.humidity_measurement()
Range:	
Describe:	Function to return the relative humidity value.

Command:	var.humidity_measurement()
Result:	
Type or Addressable Component:	Control

var.t_h_measurements()

Command:	var.t_h_measurements()
Command Syntax:	var.t_h_measurements()
Range:	
Describe:	Function to return the temperature value in degrees Celsius and the relative humidity value. Returns two floating point values.
Result:	
Type or Addressable Component:	Control

CE products: `from dht import *`



var=dht("port","opt")

Command:	var=dht("port","opt")
Command Syntax:	var=dht("port","opt")
Range:	
Describe:	This function creates an object named by the variable “var” for the DHT sensor.

Command:	var=dht("port","opt")
	"port" – the port to which the DHT sensor is attached "opt" – option that allows the use of two different types of DHT sensors – DHT11 (default) and DHT22
Result:	
Type or Addressable Component:	Control

var.temp_measurement()

Command:	var.temp_measurement()
Command Syntax:	var.temp_measurement()

Python Code Sample:	**TI Nspire CX II:**
	`from ti_hub import *`
	`d1 = dht("IN 1")`
	`t = d1.temp_measurement()`
	`print("Temp is: ", t)`

	CE products:
	`from dht import *`
	`d1 = dht("IN 1", "DHT11")`
	`t = d1.temp_measurement()`
	`print("Temp is: ", t)`

Range:	
Describe:	Function to return the temperature value in degrees Celsius.
Result:	NA
Type or Addressable Component:	Control

var.humidity_measurement()

Command:	var.humidity_measurement()
Command	var.humidity_measurement()

Command:	var.humidity_measurement()
Syntax:	
Range:	
Describe:	Function to return the relative humidity value.
Result:	
Type or Addressable Component:	Control

var.t_h_measurements()

Command:	var.t_h_measurements()
Command Syntax:	**var.t_h_measurements()**
Range:	
Describe:	Function to return the temperature value in degrees Celsius and the relative humidity value. Returns two floating point values.
Result:	
Type or Addressable Component:	Control

Ranger

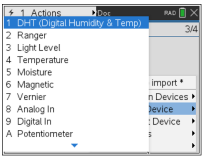
RANGER is the Ultrasonic distance sensor support module.

Add Input Device

Item	Description
Ranger	Returns the current distance measurement from the specified ultrasonic ranger.

CE products: `from ranger import *`

TI-Nspire CX II: `from ti_hub import *`



The function to create the object is pasted from the menu.

var=ranger("port")

Command:	var=ranger("port")
Command Syntax:	**var=ranger("port")**
Range:	
Describe:	Create an object for the ranger sensor attached to the “port”.
Result:	
Type or Addressable Component:	Control

var.measurement()

Command:	var.measurement()
Command Syntax:	**var.measurement()**
Python	**TI Nspire CX II:**

Command:	var.measurement()

Code Sample:	`from ti_hub import *`
	`r1 = dht("IN 2")`
	`d = r1.measurement()`
	`print("Distance is: ", d)`

	CE products:
	`from ranger import *`
	`r1 = dht("IN 2")`
	`d = r1.measurement()`
	`print("Distance is: ", d)`

Range:	
Describe:	Returns the distance measured by the sensor in meters.
Result:	
Type or Addressable Component:	Control

LightLevel (Light Sensor)

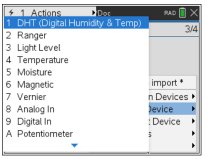
LIGHTLEVEL supports reading external light level sensors.

Add Input Device

Item	Description
Light Level	Returns the brightness level from the external light level (brightness) sensor.

CE products: `from lightlvl import *`

TI-Nspire CX II: `from ti_hub import *`



The function to create the object is pasted from the menu.

var=light_level("port")

Command:	var=light_level("port")
Command Syntax:	**var=light_level("port")**
Range:	
Describe:	Create an object for the light level sensor attached to the “port”.
Result:	
Type or Addressable Component:	Control

var.measurement()

Command:	var.measurement()
Command Syntax:	**var.measurement()**
Python	**TI Nspire CX II:**

Command:	var.measurement()

Code Sample:	`from ti_hub import *`
	`l1 = light_level("IN 1")`
	`b = l1.measurement()`
	`print("Light level is: ", b)`

	CE products:
	`from lightlvl import *`
	`l1 = light_level("IN 1")`
	`b = l1.measurement()`
	`print("Light level is: ", b)`

Range:	
Describe:	Returns the relative brightness between 0 and 100 measured by the sensor.
Result:	
Type or Addressable Component:	Control

var.range(min,max)

Command:	var.range(min,max)
Command Syntax:	**var.range(min,max)**
Range:	
Describe:	Sets the range of returned values for the sensor. This needs to be called before the “var.measurement()” function for it to take effect.
Result:	
Type or Addressable Component:	Control

Moisture

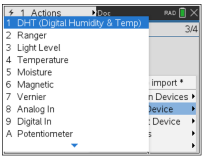
MOISTURE is the soil moisture sensor support module.

Add Input Device

Item	Description
Moisture	Returns the moisture sensor reading.

CE products: `from moisture import *`

TI-Nspire CX II: `from ti_hub import *`



The function to create the object is pasted from the menu.

var=moisture("port")

Command:	var=moisture("port")
Command Syntax:	**var=moisture("port")**
Range:	
Describe:	Create an object for the moisture sensor attached to the “port”.
Result:	
Type or Addressable Component:	Control

var.measurement()

Command:	var.measurement()
Command Syntax:	**var.measurement()**
Python Code	**TI Nspire CX II:** `from ti_hub import *`

Command:	var.measurement()

Sample:	`m1 = moisture("IN 1")`
	`m = m1.measurement()`
	`print("Moisture is: ", m)`

	CE products:
	`from moisture import *`
	`m1 = moisture("IN 1")`
	`m = m1.measurement()`
	`print("Moisture is: ", m)`

Range:	
Describe:	Returns the moisture value measured by the sensor.
Result:	
Type or Addressable Component:	Control

var.range(min,max)

Command:	var.range(min,max)
Command Syntax:	**var.range(min,max)**
Range:	Default range: 0 - 16383
Describe:	Sets the range of returned values for the sensor. This needs to be called before the “var.measurement()” function for it to take effect.
Result:	
Type or Addressable Component:	Control

Magnetic

The magnetic sensor can be used to detect when a magnet is close to the sensor.

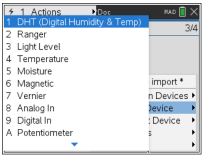
MAGNETIC is the Magnetic Hall Effect sensor support module.

Add Input Device

Item	Description
Magnetic	Detects the presence of a magnetic field. The threshold value to determine the presence of the field is set through the trigger() function. The default value of the threshold is 150.

CE products: `from magnetic import *`

TI-Nspire CX II: `from ti_hub import *`



The function to create the object is pasted from the menu.

var=magnetic("port")

Command:	var=magnetic("port")
Command Syntax:	**var=magnetic("port")**
Range:	
Describe:	Create an object for the magnetic sensor attached to the “port”.
Result:	
Type or Addressable Component:	Control

var=magnetic_close()

Command:	var=magnetic_close()
Command Syntax:	var=magnetic_close()
Range:	
Describe:	Returns a 0 or 1 if a magnet is detected close to the sensor. 0 = magnet detected 1 = no magnet detected The trigger value can be configured through the “var.trigger()” function. Default value of the trigger is 150
Result:	
Type or Addressable Component:	Control

var.measurement()

Command:	var.measurement()
Command Syntax:	var.measurement()

Python Code Sample:	TI Nspire CX II:
	`from ti_hub import *`
	`m1 = magnetic("IN 1")`
	`mg = m1.measurement()`
	`print("Magnetic strength is: ", mg)`
	`mc = m1.magnet_close()`
	`if (mc == 0):`
	`print("Magnet detected")`
	`else`
	`print("No magnet detected")`

	CE products:
	`from magnetic import *`
	`m1 = magnetic("IN 1")`
	`mg = m1.measurement()`
	`print("Magnetic strength is: ", mg)`
	`mc = m1.magnet_close()`

Command:	var.measurement()

	`if (mc == 0):`
	`print("Magnet detected")`
	`else`
	`print("No magnet detected")`

Range:	Range: 0 – 16383
Describe:	Returns a value representing the strength of the magnetic field near the sensor.
Result:	
Type or Addressable Component:	Control

var.trigger(level)

Command:	var.trigger(level)
Command Syntax:	**var.trigger(level)**
Range:	The range for 'level' is 0 – 16383 The default value for level is 150
Describe:	Use to set the trigger level for the "var.magnet_close()" function.
Result:	
Type or Addressable Component:	Control

Vernier (used with TI-SensorLink)

The “vernier” object is used with the TI Sensor Link adapter to collected data from Vernier sensors.

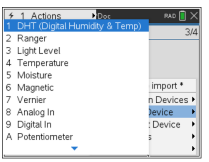
VERNIER support interfaces to allow setting up and reading Vernier analog sensors via a TI-SensorLink.

Add Input Device

Item	Description
Vernier	Reads the value from the Vernier analog sensor specified in the command. The command supports the following Vernier sensors: • **temperature** - Stainless Steel Temperature sensor. • **lightlevel** - TI Light level sensor. • **pressure** - Original gas pressure sensor • **pressure** - Newer gas pressure sensor. • **pH** - pH sensor. • **force10** - ±10 N setting, Dual Force Sensor. • **force50** - ±50 N setting, Dual Force Sensor. • **accelerometer** - Low-G Accelerometer. • **generic** - Allows setting of other sensors not supported directly above, and use of the calibrate() API above to set equation coefficients.

CE products: `from vernier import *`

TI-Nspire CX II:`from ti_hub import *`



The function to create the object is pasted from the menu.

var=vernier("port","type")

Command:	var=vernier("port","type")
Command Syntax:	**var=vernier("port","type")**
Range:	
Describe:	This function creates the “vernier” object to represent the sensor

Command:	var=vernier("port","type")
	specified by “type” connected to “port”. The supported Vernier sensors are: • Stainless Steel Temperature Probe (“temperature”) • pH Sensor (“pH”) • Gas Pressure Sensor (“pressure” and “pressure2”) • Dual-Range Force Sensor (“force10” and “force50”) • Light Sensor (“lightlevel”) • Low-g Accelerometer (“accelerometer”) For all of these sensors, the “var.measurement()” function returns the calibrated sensor value. A generic Vernier sensor is also supported. This can be used to get sensor data from a Vernier analog sensor that’s not in the list above. The sensor object will need to be calibrated with the right coefficients to get the appropriate value. In the absence of explicit calibration, the “var.measurement()” function will return a raw voltage value between 0 and 16383.
Result:	
Type or Addressable Component:	Control

var.measurement()

Command:	var.measurement()
Command Syntax:	**var.measurement()**

Python Code Sample:	**TI Nspire CX II:**
	`from ti_hub import *`
	`v1 = vernier("IN 2", "pH")`
	`pHval = v1.measurement()`
	`print("pH is: ", pHval)`

	CE products:
	`from vernier import *`
	`v1 = vernier("IN 2", "pH")`
	`pHval = v1.measurement()`
	`print("pH is: ", pHval)`

Range:	

Command:	var.measurement()
Describe:	Returns the sensor data.
Result:	
Type or Addressable Component:	Control

var.calibrate(a,b)

Command:	var.calibrate(a,b)
Command Syntax:	var.calibrate(a,b)
Range:	
Describe:	This function will be needed for a “generic” Vernier sensor that’s not in the list above. It calibrates linear ax+b sensors.
Result:	
Type or Addressable Component:	Control

var.calibrate(a,b,c,r)

Command:	var.calibrate(a,b,c,r)
Command Syntax:	var.calibrate(a,b,c,r)
Range:	
Describe:	This function is needed to calibrate thermistor style Steinhart sensors.
Result:	
Type or Addressable Component:	Control

Analog In

This object can be used to interface with analog sensors that are not explicitly supported

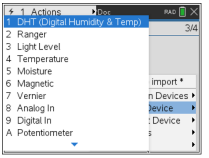
ANALOG.IN supports the use of analog input generic devices.

Add Input Device

Item	Description
Analog In	Supports the use of analog input generic devices.

CE products: `from analogin import *`

TI-Nspire CX II:`from ti_hub import *`



The function to create the object is pasted from the menu.

var=analog_in("port")

Command:	var=analog_in("port")
Command Syntax:	**var=analog_in("port")**
Range:	
Describe:	Creates an object for the analog input sensor attached to the "port"
Result:	
Type or Addressable Component:	Control

var.measurement()

Command:	var.measurement()
Command Syntax:	**var.measurement()**

Command:	var.measurement()

Python Code Sample:	**TI Nspire CX II:**
	`from ti_hub import *`
	`an1 = analog_in("BB 5")`
	`val = an1.measurement()`
	`print("Analog In value: ", val)`

	CE products:
	`from analogin import *`
	`an1 = analog_in("BB 5")`
	`val = an1.measurement()`
	`print("Analog In value: ", val)`

Range:	
Describe:	Returns the sensor data value. Default range is 0 – 16383.
Result:	
Type or Addressable Component:	Control

var.range(min,max)

Command:	var.range(min,max)
Command Syntax:	**var.range(min,max)**
Range:	
Describe:	Sets the range of returned values for the sensor. This needs to be called before the “var.measurement()” function for it to take effect.
Result:	
Type or Addressable Component:	Control

Digital In/Out

This object can be used to interface with digital devices that are not explicitly supported. The same object works with digital inputs as well as digital outputs.

DIGITAL provides interfaces for controlling a digital input / output pin.

Add Input Device

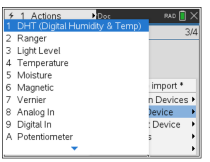
Item	Description
Digital In	Returns the current state of the digital pin connected to the DIGITAL object, or the cached state of the digital output value last SET to the object.

Add Output Device

Item	Description
Digital Out	Interfaces for controlling a digital output. • **set(val):** Sets the digital output to the value specified by "val" (0 or 1). • **on():** Sets the state of the digital output to high (1). • **off():** Sets the state of the digital output to low (0).

CE products: `from digital import *`

TI-Nspire CX II: `from ti_hub import *`



The function to create the object is pasted from the menu.

var=digital("port")

Command:	var=digital("port")
Command Syntax:	**var=digital("port")**
Range:	
Describe:	This object can be used to interface with digital devices that are not explicitly supported.

Command:	var=digital("port")
Result:	
Type or Addressable Component:	Control

var.measurement()

Command:	var.measurement()
Command Syntax:	**var.measurement()**

Python Code Sample:	**TI Nspire CX II:**
	`from ti_hub import *`
	`d1 = digital("IN 1")`
	`val = d1.measurement()`
	`print("Value is: ", val)`

	CE products:
	`from digital import *`
	`d1 = digital("IN 1")`
	`val = d1.measurement()`
	`print("Value is: ", val)`

Range:	
Describe:	Reads the digital sensor and returns 0.0 or 1.0.
Result:	
Type or Addressable Component:	Control

var.set(val)

Command:	var.set(val)
Command Syntax:	**var.set(val)**
Range:	

Command:	var.set(val)
Describe:	Set the digital device to 0 or 1.
Result:	
Type or Addressable Component:	Control

var.off()

Command:	var.off()
Command Syntax:	**var.off()**
Range:	
Describe:	Equivalent to "var.set(0)".
Result:	
Type or Addressable Component:	Control

var.on()

Command:	var.on()
Command Syntax:	**var.on()**
Range:	
Describe:	Equivalent to "var.set(1)".
Result:	
Type or Addressable Component:	Control

Potentiometer

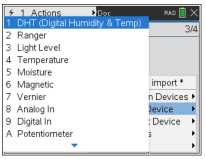
POTENTIOMETER is Generic variable resistance/potentiometer support interfaces.

Add Input Device

Item	Description
Potentiometer	Supports a potentiometer sensor. The range of the sensor can be changed by the range() function.

CE products: `from potentio import *`

TI-Nspire CX II: `from ti_hub import *`



The function to create the object is pasted from the menu.

var=potentiometer("port")

Command:	var=potentiometer("port")
Command Syntax:	**var=potentiometer("port")**
Range:	
Describe:	Creates an object for the potentiometer sensor.
Result:	
Type or Addressable Component:	Control

var.measurement()

Command:	var.measurement()
Command Syntax:	**var.measurement()**

Python Code Sample:	**TI Nspire CX II:**
	`from ti_hub import *`
	`p1 = potentiometer("IN 1")`
	`r = p1.measurement()`

Command:	var.measurement()
	`print("Value is: ", m)`

	CE products:
	`from potentio import *`
	`p1 = potentiometer("IN 1")`
	`r = p1.measurement()`
	`print("Value is: ", r)`

Range:	
Describe:	Returns the measured value of the potentiometer.
Result:	
Type or Addressable Component:	Control

var.range(min,max)

Command:	var.range(min,max)
Command Syntax:	**var.range(min,max)**
Range:	
Describe:	Sets the range of returned values for the potentiometer. This needs to be called before the "var.measurement()" function for it to take effect.
Result:	
Type or Addressable Component:	Control

Thermistor

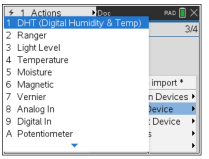
THERMISTOR is the support for reading thermistor sensors.

Add Input Device

Item	Description
Thermistor	Reads thermistor sensors. The default coefficients are designed to match the thermistor included in the Breadboard Pack of the TI-Innovator™ Hub, when used with a 10KΩ fixed resistor. A new set of calibration coefficients and reference resistance for the thermistor can be configured using the calibrate() function.

CE products: `from thermist import *`

TI-Nspire CX II: `from ti_hub import *`



The function to create the object is pasted from the menu.

var=thermistor("port")

Command:	var=thermistor("port")
Command Syntax:	**var=thermistor("port")**
Range:	
Describe:	Creates an object for the thermistor sensor.
Result:	
Type or Addressable Component:	Control

var.measurement()

Command:	var.measurement()
Command Syntax:	var.measurement()

Python Code Sample:	TI Nspire CX II:
	`from ti_hub import *`
	`t1 = thermistor("IN 1")`
	`val = t1.measurement()`
	`print("Thermistor value: ", val)`

	CE products:
	`from thermist import *`
	`t1 = thermistor("IN 1")`
	`val = t1.measurement()`
	`print("Thermistor value: ", val)`

Range:	
Describe:	Returns the measured value of the thermistor.
Result:	
Type or Addressable Component:	Control

var.calibrate(c1,c2,c3,r)

Command:	var.calibrate(c1,c2,c3,r)
Command Syntax:	var.calibrate(c1,c2,c3,r)
Range:	
Describe:	Calibrates the thermistor with the specified values.
Result:	
Type or Addressable Component:	Control

Loudness

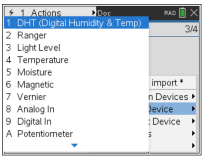
LOUNDESS input sensor support routines for sound loudness sensors.

Add Input Device

Item	Description
Loudness	Supports sound loudness sensors.

CE products: `from loudness import *`

TI-Nspire CX II: `from ti_hub import *`



The function to create the object is pasted from the menu.

var=loudness("port")

Command:	var=loudness("port")
Command Syntax:	**var=loudness("port")**
Range:	
Describe:	Creates an object for the thermistor sensor.
Result:	
Type or Addressable Component:	Control

var.measurement()

Command:	var.measurement()
Command Syntax:	**var.measurement()**
Python Code	**TI Nspire CX II:** `from ti_hub import *`

Command:	var.measurement()

Sample:	`l1 = loudness("IN 1")`
	`ld_val = l1.measurement()`
	`print("Loudness value is: ", ld_val)`

	CE products:
	`from loudness import *`
	`l1 = loudness("IN 1")`
	`ld_val = l1.measurement()`
	`print("Loudness value is: ", ld_val)`

Range:	
Describe:	Returns the measured value of the loudness sensor.
Result:	
Type or Addressable Component:	Control

var.range(min,max)

Command:	var.range(min,max)
Command Syntax:	**var.range(min,max)**
Range:	
Describe:	Sets the range of returned values for the loudness sensor. This needs to be called before the "var.measurement()" function for it to take effect.
Result:	
Type or Addressable Component:	Control

Color Input

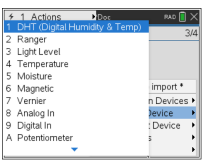
COLORINPUT provides interfaces to an I2C-connected Color Input sensor.

Add Input Device

Item	Description
Color Input	Provides interfaces to an I2C-connected Color Input sensor. The bb_port pin is used in addition to the I2C port to control the LED on the color sensor. • **color_number():** Returns a value from 1 to 9 that represents the color the sensor is detecting. The numbers represent the colors per the following mapping: 1: Red 2: Green 3: Blue 4: Cyan 5: Magenta 6: Yellow 7: Black 8: White 9: Gray • **red():** Returns a value from 0 to 255 that represents the intensity of the RED color level being detected. • **green():** Returns a value from 0 to 255 that represents the intensity of the GREEN color level being detected. • **blue():** Returns a value from 0 to 255 that represents the intensity of the BLUE color level being detected. • **gray():** Returns a value from 0 to 255 that represents the gray level being detected, where 0 is black and 255 is white.

CE products: `from colorinp import *`

TI-Nspire CX II: `from ti_hub import *`



The function to create the object is pasted from the menu.

var=color_input("bb_port")

Command:	**var=color_input("bb_port")**
Command Syntax:	**var=color_input("bb_port")**

Python Code Sample:	**TI Nspire CX II:**
	`from ti_hub import *`
	`c1 = color_input("BB 2")`
	`value = c1.color_number()`
	`if (value == 1):`
	`··print("Red!")`
	`else:`
	`··print("Not red")`

	CE products:
	`from colorinp import *`
	`c1 = color_input("BB 2")`
	`value = c1.color_number()`
	`if (value == 1):`
	`··print("Red!")`
	`else:`
	`··print("Not red")`

Range:	
Describe:	Creates an object for the color_input sensor. The sensor is attached to the I2C port and the "bb_port" is the pin on the breadboard port for the pin to control the LED in the sensor.
Result:	
Type or Addressable Component:	Control

var.color_number()

Command:	**var.color_number()**
Command Syntax:	**var.color_number()**

Command:	var.color_number()
Range:	
Describe:	Returns the a value between 1 – 9 from the color sensor. 1 – red 2 – green 3 – blue 4 – cyan 5 – magenta 6 - yello 7 – black 8 - white 9 – gray
Result:	
Type or Addressable Component:	Control

var.red()

Command:	var.red()
Command Syntax:	**var.red()**
Range:	
Describe:	Returns a value between 0 and 255 representing the “red” component of the color under the sensor.
Result:	
Type or Addressable Component:	Control

var.green()

Command:	var.green()
Command Syntax:	**var.green()**

Command:	var.green()
Range:	
Describe:	Returns a value between 0 and 255 representing the “green” component of the color under the sensor.
Result:	
Type or Addressable Component:	Control

var.blue()

Command:	var.blue()
Command Syntax:	var.blue()
Range:	
Describe:	Returns a value between 0 and 255 representing the “blue” component of the color under the sensor.
Result:	
Type or Addressable Component:	Control

var.gray()

Command:	var.gray()
Command Syntax:	var.gray()
Range:	
Describe:	Returns a value between 0 and 255 representing the “gray component of the color under the sensor.
Result:	
Type or Addressable Component:	Control

BB Port (Breadboard Port)

BBPORT provides support for using all 10 BB port pins as a combined digital input/output port.

Add Input Device

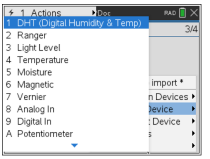
Item	Description
BBPort	Provides support for using all 10 BB port pins as a combined digital input/output port. The initialization functions have an optional "mask" parameter that allows the use of the subset of the 10 pins. • **read_port():** Reads the current values on the input pins of the BB port. • **write_port(value):** Sets the output pin values to the specified value, where value is between 0 and 1023. Note that the value is also adjusted against the mask value in the var=bbport(mask) operation, if a mask was provided.

Add Output Device

Item	Description
BB Port	Provides functions for programming the TI-RGB Array. See the details above.

CE products: `from bbport import *`

TI-Nspire CX II: `from ti_hub import *`



The function to create the object is pasted from the menu.

var=bb_port("mask")

Command:	var=bb_port("mask")
Command Syntax:	**var=bb_port("mask")**
Range:	
Describe:	Creates an object for the BreadBoard port (bb_port).

Command:	var=bb_port("mask")
	"mask" is an optional parameter with values between 0 and 1023 and is used to selectively connect specific pins. The mask value may be specified in decimal, binary, or hexadecimal format. For example, 1023 or 0X3FF selects all 10 pins and is the default internal mask value used by the bb_port() object if a "mask" is not specified.
Result:	
Type or Addressable Component:	Control

var.read_port()

Command:	var.read_port()
Command Syntax:	**var.read_port()**

Python Code Sample:	**TI Nspire CX II:**
	`from ti_hub import *`
	`bb = bbport()`
	`readval = bb.read_port()`
	`print("BB Port value is: ", readval)`
	`bb.write_port(0x55)`

	CE products:
	`from bbport import *`
	`bb = bbport()`
	`readval = bb.read_port()`
	`print("BB Port value is: ", readval)`
	`bb.write_port(0x55)`

Range:	
Describe:	Reads the current values on the input pins of the BB port.
Result:	
Type or Addressable Component:	Control

var.read_port(mask)

Command:	var.read_port(mask)
Command Syntax:	**var.read_port(mask)**
Range:	
Describe:	Reads the current values on the input pins of the BB port. "mask" allows for specifying which pins to read as inputs. Default if not provided = the connected pins mask specified during object initialization. This mask is also verified for valid range and is logically AND'd internally against the connected pin mask.
Result:	
Type or Addressable Component:	Control

var.write_port(value)

Command:	var.write_port(value)
Command Syntax:	**var.write_port(value)**
Range:	
Describe:	Sets the output pin values to the specified value, where value is between 0 and 1023. Note that the value is also adjusted against the mask value in the var=bbport(mask) operation, if a mask was provided.
Result:	
Type or Addressable Component:	Control

var.write_port(value, mask)

Command:	var.write_port(value, mask)
Command Syntax:	**var.write_port(value, mask)**

Command:	var.write_port(value, mask)
Range:	
Describe:	Sets the output pin values to the specified value, where value is between 0 and 1023. Note that the value is also adjusted against the mask value in the var=bbport(mask) operation, if a mask was provided. Optional mask allows for specifying which subset of pins to set as outputs. Default if not provided = the connected pins mask specified during object initialization. This mask is also verified for valid range and is logically AND'd internally against the connected pin mask.
Result:	
Type or Addressable Component:	Control

Hub Time

This object can be used to interface with analog sensors that are not explicitly supported

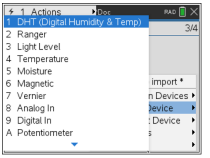
TIMER interface to the TI-Innovator™ TIMER object.

Add Input Device

Item	Description
Hub Time	Provides access to the internal millisecond timer.

CE products: `from analogin import *`

TI-Nspire CX II: `from ti_hub import *`



The function to create the object is pasted from the menu.

var=hub_time()

Command:	var=hub_time()
Command Syntax:	**var=hub_time()**
Range:	
Describe:	Creates an object for the timer on the TI-Innovator™ Hub.
Result:	
Type or Addressable Component:	Control

var.measurement()

Command:	var.measurement()
Command Syntax:	**var.measurement()**

Command:	var.measurement()

Python Code Sample:	**TI Nspire CX II:**
	`from ti_hub import *`
	`from time import *`
	`t1=hub_time()`
	`t1.reset_time()`
	`print("Before: ", t1.measurement())`
	`sleep(.5)`
	`print("After: ", t1.measurement())`

	CE products:
	`from timer import *`
	`from time import *`
	`t1=hub_time()`
	`t1.reset_time()`
	`print("Before: ", t1.measurement())`
	`sleep(.5)`
	`print("After: ", t1.measurement())`

Range:	
Describe:	Returns the current value of the timer in milliseconds.
Result:	
Type or Addressable Component:	Control

var.reset_time()

Command:	var.reset_time()
Command Syntax:	**var.reset_time()**
Range:	
Describe:	Resets the time to zero.
Result:	
Type or Addressable Component:	Control

TI-RGB Array

TI-RGB Array control interfaces for programming the TI-RGB Array with a variety of easy to difficult operations.

Add Input Device

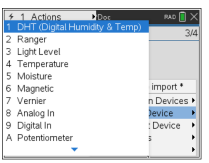
Item	Description
TI-RGB Array	Provides functions for programming the TI-RGB Array. The initialization function accepts an optional "LAMP" parameter to enable a high-brightness mode for the TI-RGB Array that requires an external power supply. • **set(led_position, r,g,b):** Sets a specific led_position (0-15) to the specified r,g,b value, where r,g,b are values from 0 to 255. • **set_all(r,g,b):** Sets all RGB LEDs in the array to the same r,g,b value. • **all_off():** Turns off all RGBs in the array. • **measurement():** Returns the approximate current draw that the RGB array is using from the TI-Innovator™ in milliAmps. • **pattern(pattern):** Using the value of the argument as a binary value in the range 0 to 65535, turns on pixels where a 1 value in the representation would be. LEDs are turned on as RED with pwm level value of 255.

Add Output Device

Item	Description
TI-RGB Array	Provides functions for programming the TI-RGB Array.

CE products: `from rgb_arr import *`

TI-Nspire CX II: `from ti_hub import *`



The function to create the object is pasted from the menu.

var=rgb_array()

Command:	var=rgb_array()
Command Syntax:	**var=rgb_array()**

Python Code Sample:	**TI Nspire CX II:**
	`from ti_hub import *`
	`r1 = rgb_array()`
	`r1.set_all(255,0,0)`
	`c_red = r1.measurement()`
	`r1.set_all(255,255,255)`
	`c_white = r1.measurement()`
	`print("Red LEDs current: ", c_red, "White LEDs current: ", c_white)`

	CE products:
	`from rgb_arr import *`
	`r1 = rgb_array()`
	`r1.set_all(255,0,0)`
	`c_red = r1.measurement()`
	`r1.set_all(255,255,255)`
	`c_white = r1.measurement()`
	`print("Red LEDs current: ", c_red, "White LEDs current: ", c_white)`

Range:	
Describe:	Creates an object for the TI-RGB Array. The initialization includes verification that the TI-RGB Array is connected to the TI-Innovator™ Hub. Note that TI-RGB Array is both an input and an output device. It is included in both menus. An optional parameter (“lamp”) requires the presence of an external battery to increase the brightness of the LEDs.
Result:	
Type or Addressable Component:	Control

var.set(led_position, red, green, blue)

Command:	var.set(led_position, red, green, blue)
Command Syntax:	var.set(led_position, red, green, blue)
Range:	
Describe:	Sets a specific led_position (0-15) to the specified r, g, b value, where r,g,b are values from 0 to 255.
Result:	
Type or Addressable Component:	Control

var.set_all(red, green, blue)

Command:	var.set_all(red, green, blue)
Command Syntax:	var.set_all(red, green, blue)
Range:	
Describe:	Sets all RGB LEDs in the array to the same r,g,b value.
Result:	
Type or Addressable Component:	Control

var.all_off()

Command:	var.all_off()
Command Syntax:	var.all_off()
Range:	
Describe:	Turns off all RGB LEDs in the array.
Result:	
Type or	Control

Command:	var.all_off()
Addressable Component:	

var.pattern(value)

Command:	var.pattern(value)
Command Syntax:	**var.pattern(value)**
Range:	
Describe:	Using the value of the argument as a binary value in the range 0 to 65535, turns on pixels where a 1 value in the representation would be. LEDs are turned on as RED. Example: a value of 201 which is 0b 1100 1001 in binary would turn ON LEDs 0, 3, 6 7
Result:	
Type or Addressable Component:	Control

var.measurement()

Command:	var.measurement()
Command Syntax:	**var.measurement()**
Range:	
Describe:	Returns the approximate current draw that the RGB array is using from the TI-Innovator in milliAmps.
Result:	
Type or Addressable Component:	Control

Output

Output Devices

Add Output Device

This menu has a list of the output devices supported by the ti_hub module. All the menu items will paste the name of the object and expect a variable and a port used with the device.

LED

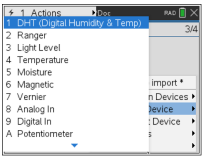
LED interfaces for controlling externally connected LEDs.

Add Output Device

Item	Description
LED	Functions for controlling externally connected LEDs.

CE products: `from led import *`

TI-Nspire CX II: `from ti_hub import *`



The function to create the object is pasted from the menu.

var=led("port")

Command:	var=led("port")
Command Syntax:	**var=led("port")**
Range:	
Describe:	Creates an object for an LED. Port can be OUT 1, OUT 2 or OUT 3
Result:	
Type or Addressable Component:	Control

var.off()

Command:	var.off()
Command Syntax:	**var.off()**

Command:	var.off()
Range:	
Describe:	Turns off the LED.
Result:	
Type or Addressable Component:	Control

var.on()

Command:	var.on()
Command Syntax:	var.on()
Range:	
Describe:	Turns on the LED.
Result:	
Type or Addressable Component:	Control

var.blink(freq,time)

Command:	var.blink(freq,time)
Command Syntax:	var.blink(freq,time)

Python Code Sample:	TI Nspire CX II:
	`from ti_hub import *`
	`from time import *`
	
	`l1=led("OUT 1")`
	`for i in range(5):`
	`··l1.on()`
	`··sleep(1)`
	`··l1.off()`

Command:	var.blink(freq,time)
	`l1.blink(5,2)`

	CE products:
	`from led import *`
	`from time import *`
	
	`l1=led("OUT 1")`
	`for i in range(5):`
	`··l1.on()`
	`··sleep(1)`
	`··l1.off()`
	
	`l1.blink(5,2)`

Range:	
Describe:	Blinks the LED at the specified frequency for the specified time. frequency: 0.1 – 20 Hz time: 0.1 – 100 seconds
Result:	
Type or Addressable Component:	Control

RGB

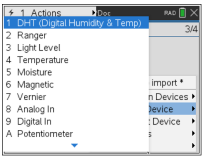
RGB support for controlling external RGB LEDs.

Add Output Device

Item	Description
RGB	Support for controlling external RGB LEDs.

CE products: `from rgb import *`

TI-Nspire CX II: `from ti_hub import *`



The function to create the object is pasted from the menu.

var=rgb("port")

Command:	var=rgb("port")
Command Syntax:	**var=rgb("port")**
Range:	
Describe:	Creates an object for an RGB LED. Port can be OUT 1, OUT 2 or OUT 3 **CE module:** var=rgb("r", "g", "b") – allows connecting an RGB LED to three different breadboard pins or to the same OUT port.
Result:	
Type or Addressable Component:	Control

var.rgb(r,g,b)

Command:	var.rgb(r,g,b)
Command	**var.rgb(r,g,b)**

Command:	var.rgb(r,g,b)
Syntax:	
Range:	
Describe:	Sets the color of the RGB LED to the color specified by the 'r' (red), 'g'(green), 'b'(blue) values. r, g, b all are between 0 and 255.
Result:	
Type or Addressable Component:	Control

var.blink(freq,time)

Command:	var.blink(freq,time)
Command Syntax:	**var.blink(freq,time)**

Python Code Sample:	**TI Nspire CX II:**
	`from ti_hub import *`
	`rgb1=rgb("OUT 2")`
	`rgb1.rgb(255,0,255)`
	`rgb1.blink(4,4)`

	CE products:
	`from rgb import *`
	`rgb1=rgb("OUT 2", "OUT 2", "OUT 2")`
	`rgb1.rgb(255,0,255)`
	`rgb1.blink(4,4)`

Range:	
Describe:	Blinks the RGB LED at the specified frequency for the specified time. frequency: 0.1 – 20 Hz time: 0.1 – 100 seconds Note tha.t var.rgb() will need to be called first to set the color of the RGB LED
Result:	
Type or Addressable Component:	Control

var.off()

Command:	var.off()
Command Syntax:	var.off()
Range:	
Describe:	Turns off the RGB LED.
Result:	
Type or Addressable Component:	Control

Speaker

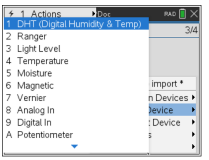
SPEAKER is the module for supporting an external speaker with the TI-Innovator™ Hub.

Add Output Device

Item	Description
Speaker	Functions for supporting an external speaker with the TI-Innovator™ Hub. The functions are the same as the ones for "sound" above.

CE products: `from speaker import *`

TI-Nspire CX II: `from ti_hub import *`



The function to create the object is pasted from the menu.

var=speaker("port")

Command:	var=speaker("port")
Command Syntax:	**var=speaker("port")**
Range:	
Describe:	Creates an object for an external speaker. Port can be OUT 1, OUT 2, OUT 3 or any of the BB port pins. The external speaker has the same API as the built-in speaker.
Result:	
Type or Addressable Component:	Control

var.tone(freq,time)

Command:	var.tone(freq,time)
Command Syntax:	var.tone(freq,time)
Range:	
Describe:	Plays a tone specified by 'freq' (0-8000 Hz) for the 'time' (0.1 – 100s).
Result:	
Type or Addressable Component:	Control

var.note("note",time)

Command:	var.note("note",time)
Command Syntax:	var.note("note",time)

Python Code Sample:	**TI Nspire CX II:**
	`from ti_hub import *`
	`from time import *`
	
	`sp1=speaker("OUT 1")`
	
	`sp1.tone(440,2)`
	`sleep(2)`
	`sp1.note("A4",4)`

	CE products:
	`from speaker import *`
	`from time import *`
	
	`sp1=speaker("OUT 1")`
	
	`sp1.tone(440,2)`
	`sleep(2)`
	`sp1.note("A4",4)`

Range:	

Command:	var.note(“note”,time)
Describe:	Plays a note specified by ‘note’ for the ‘time’ (0.1 – 100s). Notes can be: "C", "CS", "D", "DS", "E", "F", "FS", "G", "GS", "A", "AS", and "B". And the octave numbers range from 1 to 9 (inclusive)).
Result:	
Type or Addressable Component:	Control

Power

POWER is Power / FET control interfaces for controlling external power with the TI-Innovator™ Hub.

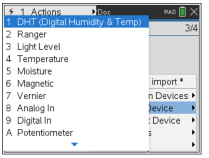
Add Output Device

This menu has a list of the output devices supported by the ti_hub module. All the menu items will paste the name of the object and expect a variable and a port used with the device.

Item	Description
Power	Functions for controlling external power with the TI-Innovator™ Hub. • **set(value):** Sets the Power level to the specified value, between 0 and 100. • **on():** Sets the Power level to 100. • **off():** Sets the Power level to 0.

CE products: `from power import *`

TI-Nspire CX II: `from ti_hub import *`



The function to create the object is pasted from the menu.

var=power("port")

Command:	var=power("port")
Command Syntax:	**var=power("port")**
Range:	
Describe:	Creates an object for a Power / FET control interfaces for controlling external power with the TI-Innovator.
Result:	
Type or Addressable Component:	Control

var.set(value)

Command:	var.set(value)
Command Syntax:	**var.set(value)**

Python Code Sample:	**TI Nspire CX II:**
	`from ti_hub import *`
	`from time import *`
	
	`p1=power("OUT 3")`
	
	`# Set 50% power`
	`p1.set(50)`
	`sleep(2)`
	`# Turn off`
	`p1.off()`

	CE products:
	`from power import *`
	`from time import *`
	
	`p1=power("OUT 3")`
	
	`# Set 50% power`
	`p1.set(50)`
	`sleep(2)`
	`# Turn off`
	`p1.off()`

Range:	
Describe:	Sets the state of the output to “value” (between 0 and 100).
Result:	
Type or Addressable Component:	Control

var.off()

Command:	var.off()
Command Syntax:	var.off()
Range:	
Describe:	Sets the power level to zero.
Result:	
Type or Addressable Component:	Control

var.on()

Command:	var.on()
Command Syntax:	var.on()
Range:	
Describe:	Sets the power level to 100.
Result:	
Type or Addressable Component:	Control

Continuous Servo

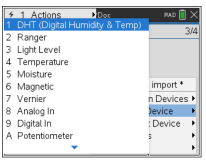
CONTINUOUS SERVO provides the interfaces for controlling a continuous servo motor.

Add Output Device

Item	Description
Continuous Servo	Functions for controlling continuous servo motors. • **set_cw(speed,time):** The servo will spin in the clockwise direction at the specified speed (0-255) and for the specific duration in seconds. • **set_ccw(speed,time):** The servo will spin in the counter-clockwise direction at the specified speed (0-255) and for the specific duration in seconds. • **stop():** Stops the continuous servo.

CE products: `from conservo import *`

TI-Nspire CX II: `from ti_hub import *`



The function to create the object is pasted from the menu.

var=continuous_servo("OUT 3")

Command:	var=continuous_servo("OUT 3")
Command Syntax:	**var=continuous_servo("OUT 3")**

Python Code Sample:	**TI Nspire CX II:**
	`from ti_hub import *`
	`from time import *`
	
	`cs1=continuous_servo("OUT 3")`
	
	`cs1.set_cw(100,5)`
	`sleep(5)`
	`cs1.set_ccw(100,5)`

	CE products:

Command:	var=continuous_servo("OUT 3")

	`from conservo import *`
	`from time import *`
	
	`cs1=continuous_servo("OUT 3")`
	
	`cs1.set_cw(100,5)`
	`sleep(5)`
	`cs1.set_ccw(100,5)`

Range:	
Describe:	Creates an object for a continuous servo motor. The port is limited to OUT 3 because a servo motor needs 5V to operate that is only available on OUT 3.
Result:	
Type or Addressable Component:	Control

var.set_cw(speed,time)

Command:	var.set_cw(speed,time)
Command Syntax:	**var.set_cw(speed,time)**
Range:	
Describe:	Turn the motor clockwise at the specified "speed" (0 – 100) for the specified time in seconds.
Result:	
Type or Addressable Component:	Control

var.set_ccw(speed,time)

Command:	var.set_ccw(speed,time)
Command Syntax:	**var.set_ccw(speed,time)**

Command:	var.set_ccw(speed,time)
Range:	
Describe:	Turn the motor counter-clockwise at the specified “speed” (0 – 100) for the specified time in seconds.
Result:	
Type or Addressable Component:	Control

var.stop()

Command:	var.stop()
Command Syntax:	**var.stop()**
Range:	
Describe:	Stops the motor.
Result:	
Type or Addressable Component:	Control

Analog Out

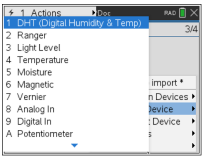
ANALOG.OUT supports a software generated pulse-width modulation output with variable duty cycle.

Add Output Device

Item	Description
Analog Out	Functions for the use of analog output generic devices.

CE products: `from analogout import *`

TI-Nspire CX II: `from ti_hub import *`



The function to create the object is pasted from the menu.

var=analog_out ("port")

Command:	var=analog_out ("port")
Command Syntax:	**var=analog_out ("port")**
Range:	
Describe:	Creates an object for an analog output device connected to the "port".
Result:	
Type or Addressable Component:	Control

var.set(value)

Command:	var.set(value)
Command Syntax:	**var.set(value)**

Command:	var.set(value)

Python Code Sample:	**TI Nspire CX II:**
	`from ti_hub import *`
	
	`a1=analog_out("BB 5")`
	`a1.set(127)`

	CE products:
	`from analgout import *`
	
	`a1=analog_out("BB 5")`
	`a1.set(127)`

Range:	
Describe:	Sets the output PWM (pulse width modulated) value for the analog_out object specified. Value is from 0 to 255.
Result:	
Type or Addressable Component:	Control

var.off()

Command:	var.off()
Command Syntax:	**var.off()**
Range:	
Describe:	Sets the output PWM value to zero, driving the output pin to digital low level. Equivalent to "var.set(0)".
Result:	
Type or Addressable Component:	Control

var.on()

Command:	var.on()
Command Syntax:	var.on()
Range:	
Describe:	Sets the output PWM value to 255, which results in a digital high signal being continuously output. Equivalent to "var.set(255)".
Result:	
Type or Addressable Component:	Control

Vibration Motor

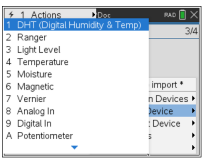
VIBRATION MOTOR control interfaces for running vibration motors.

Add Output Device

Item	Description
Vibration Motor	Functions for controlling vibration motors. • **set(val):** Sets the vibration motor intensity to "val" (0-255). • **off():** Turns the vibration motor off. • **on():** Turns the vibration motor on at the highest level.

CE products: `from vibmotor import *`

TI-Nspire CX II: `from ti_hub import *`



The function to create the object is pasted from the menu.

var=vibration_motor("OUT 3")

Command:	var=vibration_motor("OUT 3")
Command Syntax:	**var=vibration_motor("OUT 3")**
Range:	
Describe:	Creates an object for a vibration motor. The port is limited to OUT 3 because a servo motor needs 5V to operate that is only available on OUT 3.
Result:	
Type or Addressable Component:	Control

var.set(value)

Command:	var.set(value)
Command Syntax:	var.set(value)

Python Code Sample:	TI Nspire CX II:
	`from ti_hub import *`
	`from time import *`
	
	`vm1=vibration_motor("OUT 3")`
	`vm1.set(127)`
	`sleep(5)`
	`vm1.off()`

	CE products:
	`from vibmotor import *`
	`from time import *`
	
	`vm1=vibration_motor("OUT 3")`
	`vm1.set(127)`
	`sleep(5)`
	`vm1.off()`

Range:	
Describe:	Turns on the vibration motor to the power specified by “value”. Value is from 0 to 255.
Result:	
Type or Addressable Component:	Control

var.off()

Command:	var.off()
Command Syntax:	var.off()
Range:	
Describe:	Turns off the vibration motor.

Command:	var.off()
Result:	
Type or Addressable Component:	Control

var.on()

Command:	var.on()
Command Syntax:	**var.on()**
Range:	
Describe:	Turns on the vibration motor to the highest level.
Result:	
Type or Addressable Component:	Control

Relay

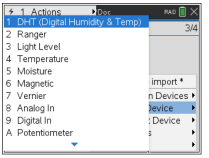
RELAY control interfaces for dealing with relays.

Add Output Device

Item	Description
Relay	Controls interfaces for controlling relays. • **on():** Sets the relay to the ON state. • **off():** Sets the relay to the OFF state.

CE products: `from relay import *`

TI-Nspire CX II: `from ti_hub import *`



The function to create the object is pasted from the menu.

var=relay("OUT 3")

Command:	var=relay("OUT 3")
Command Syntax:	**var=relay("OUT 3")**

Python Code Sample:	**TI Nspire CX II:**
	`from ti_hub import *`
	`from time import *`
	
	`r1=relay("OUT 3")`
	
	`r1.on()`
	`sleep(5)`
	`r1.off()`

	CE products:
	`from relay import *`
	`from time import *`
	
	`r1=relay("OUT 3")`

Command:	var=relay("OUT 3")

	`r1.on()`
	`sleep(5)`
	`r1.off()`

Range:	
Describe:	Creates an object for a relay The port is limited to OUT 3 because a relay needs 5V to operate that is only available on OUT 3.
Result:	
Type or Addressable Component:	Control

var.off()

Command:	var.off()
Command Syntax:	**var.off()**
Range:	
Describe:	Turns off the relay.
Result:	
Type or Addressable Component:	Control

var.on()

Command:	var.on()
Command Syntax:	**var.on()**
Range:	
Describe:	Turns on the relay.

Command:	var.on()
Result:	
Type or Addressable Component:	Control

Servo

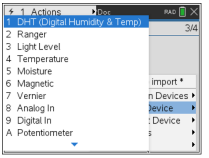
SERVO the Sweep Servo motor control interfaces.

Add Output Device

Item	Description
Servo	Functions for controlling servo motors. • **set_position(pos):** Sets the sweep servo position within a range of -90 to +90. • **zero():** Sets the sweep servo to the zero position.

CE products: `from servo import *`

TI-Nspire CX II: `from ti_hub import *`



The function to create the object is pasted from the menu.

var=servo("OUT 3")

Command:	var=servo("OUT 3")
Command Syntax:	**var=servo("OUT 3")**

Python Code Sample:	**TI Nspire CX II:**
	`from ti_hub import *`
	`from time import *`
	`servo1=servo("OUT 3")`
	
	`servo1.zero()`
	`sleep(2)`
	`servo1.set_position(45)`

	CE products:
	`from servo import *`
	`from time import *`
	`servo1=servo("OUT 3")`

Command:	var=servo("OUT 3")

	`servo1.zero()`
	`sleep(2)`
	`servo1.set_position(45)`

Range:	
Describe:	Creates an object for a sweep servo motor. The port is limited to OUT 3 because a servo motor needs 5V to operate that is only available on OUT 3.
Result:	
Type or Addressable Component:	Control

var.set_position(pos)

Command:	var.set_position(pos)
Command Syntax:	**var.set_position(pos)**
Range:	
Describe:	Sets the sweep servo position within a range of -90 to +90, with the pos rounded to the nearest tenth.
Result:	
Type or Addressable Component:	Control

var.zero()

Command:	var.zero()
Command Syntax:	**var.zero()**
Range:	
Describe:	Sets the sweep servo to the zero position.

Command:	var.zero()
Result:	
Type or Addressable Component:	Control

Squarewave

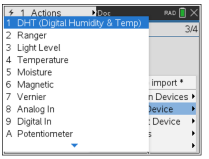
SQUAREWAVE is the software square wave generation interface with duty cycle variation control.

Add Output Device

Item	Description
Squarewave	Functions for generating a square wave. • **set(frequency,duty,time):** Sets the output squarewave with a default duty cycle of 50% (if duty is not specified) and an output frequency specified by "frequence". The frequency may be from 1 to 500 Hz. The duty cycle, if specified, may be from 0 to 100%. • **off():** Turns the squarewave off.

CE products: `from squarewv import *`

TI-Nspire CX II: `from ti_hub import *`



The function to create the object is pasted from the menu.

var=squarewave("port")

Command:	var=squarewave("port")
Command Syntax:	**var=squarewave("port")**

Python Code Sample:	**TI Nspire CX II:**
	`from ti_hub import *`
	
	`sq1=squarewave("BB 7")`
	
	`# Generate squarewave of`
	`# 200Hz at 25% duty cycle`
	`# for 5 seconds`
	`sq1.set(200,25,5)`

	CE products: `from squarewv import *`

Command:	var=squarewave("port")

	`sq1=squarewave("BB 7")`
	
	`# Generate squarewave of`
	`# 200Hz at 25% duty cycle`
	`# for 5 seconds`
	`sq1.set(200,25,5)`

Range:	
Describe:	Creates an object for a squarewave generator. The port is limited to OUT 3 because a servo motor needs 5V to operate that is only available on OUT 3.
Result:	
Type or Addressable Component:	Control

var.set(freq,duty,time)

Command:	var.set(freq,duty,time)
Command Syntax:	**var.set(freq,duty,time)**
Range:	
Describe:	Sets the output squarewave with a default duty cycle of 50% (if duty is not specified) and an output frequency specified by “freq”. “freq” may be from 1 to 500 Hz. duty cycle, if specified, may be from 0 to 100%.
Result:	
Type or Addressable Component:	Control

var.off()

Command:	var.off()
Command	**var.off()**

Command:	var.off()
Syntax:	
Range:	
Describe:	Turns off the squarewave being generated and sets output line on TI-Innovator hub to a digital low state.
Result:	
Type or Addressable Component:	Control

Digital in|out

DIGITAL provides interfaces for controlling a digital input / output pin.

Add Input Device

Item	Description
Digital In	Returns the current state of the digital pin connected to the DIGITAL object, or the cached state of the digital output value last SET to the object.

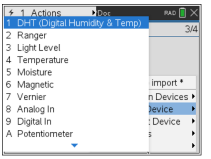
Add Output Device

Item	Description
Digital Out	Interfaces for controlling a digital output. • **set(val):** Sets the digital output to the value specified by "val" (0 or 1). • **on():** Sets the state of the digital output to high (1). • **off():** Sets the state of the digital output to low (0).

The **Digital** (input | output) module is implemented as a Python class and has methods defined as:

CE products: `from digital import *`

TI-Nspire CX II: `from ti_hub import *`



The function to create the object is pasted from the menu.

var=digital("port")

Command:	var=digital("port")
Command Syntax:	**var=digital("port")**
Range:	
Describe:	Creates an object for a digital device. This can be an input or an output device.

Command:	var=digital("port")
Result:	
Type or Addressable Component:	Control

var.measurement()

Command:	var.measurement()
Command Syntax:	**var.measurement()**

Python Code Sample:	**TI Nspire CX II:**
	`from ti_hub import *`
	`from time import *`
	
	`d1=digital("BB 1")`
	`d1.on()`
	`sleep(5)`
	`d1.off()`

	CE products:
	`from digital import *`
	`from time import *`
	
	`d1=digital("BB 1")`
	`d1.on()`
	`sleep(5)`
	`d1.off()`

Range:	
Describe:	Reads the state of the digital input and returns a value of 1 or 0 to the caller.
Result:	
Type or Addressable Component:	Control

var.set(value)

Command:	var.set(value)
Command Syntax:	var.set(value)
Range:	
Describe:	Sets the state of the digital output to “value” (0 or 1).
Result:	
Type or Addressable Component:	Control

var.off()

Command:	var.off()
Command Syntax:	var.off()
Range:	
Describe:	Sets the state of the digital output to low (0).
Result:	
Type or Addressable Component:	Control

var.on()

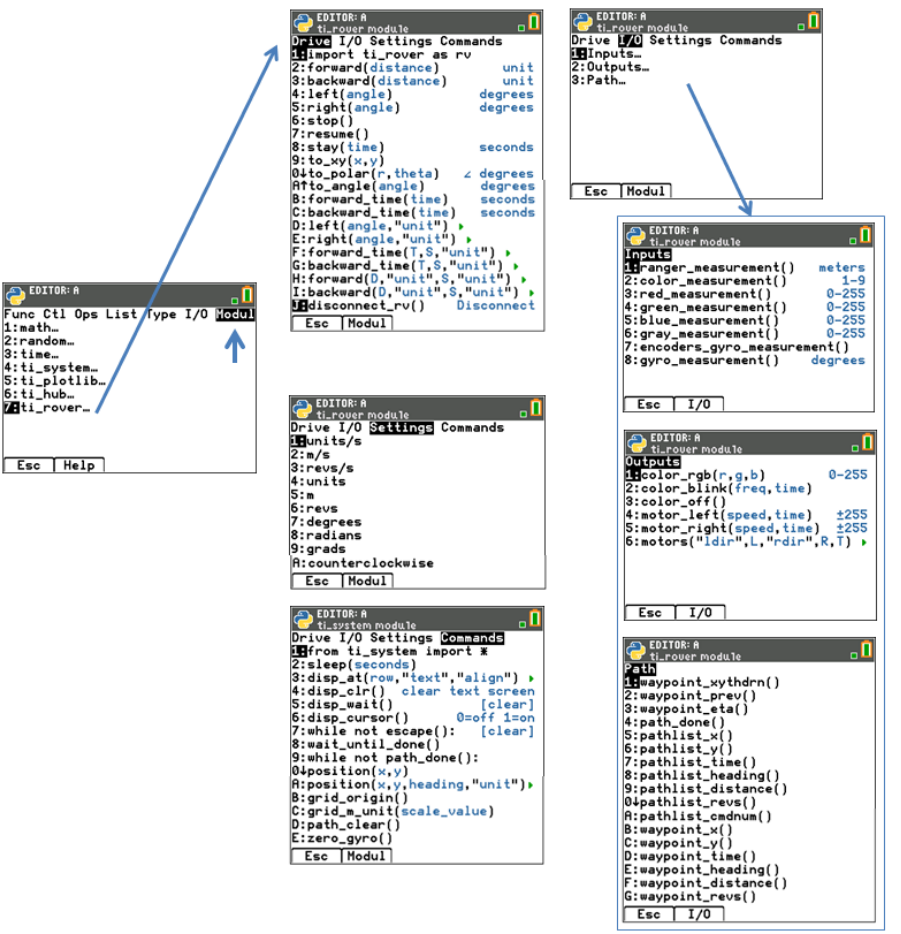
Command:	var.on()
Command Syntax:	var.on()
Range:	
Describe:	Sets the state of the digital output to high (1).
Result:	
Type or Addressable Component:	Control

Python ti_rover Module

As it appears in the CE Graphing Calculators

[Fns…]>Modul: ti_rover module

ti_rover methods are not listed in Catalog and thus, not listed in the Reference Guide. Please use the screen information in the menus for arguments and argument default or allowed value details. More information on Python programming for TI-Innovator™ Hub and TI-Innovator™ Rover will be available at education.ti.com.



Notes:

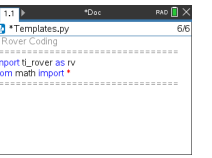
In TI-Python programming, you do not need to include methods to connect and disconnect TI-Innovator™ Rover. The TI-Innovator™ Rover Python methods handle connect and disconnect with no additional methods. This is a bit different than programming TI-Innovator™ Rover in TI-Basic.

rv.stop() executes as a pause and then resume continues with the Rover movements in the queue. If another movement command is executed after

rv.stop(), then movement queue is cleared. This again is a bit different than programming TI-Innovator™ Rover in TI-Basic.

CE Calculators

TI-Nspire™ CX



import ti_rover

Note: When creating a new program that uses this module, it is recommended to use the **Rover Coding** program type. This will ensure that all the relevant modules are imported.

Item	Description
import ti_rover as rv	Imports all methods (functions) from the ti_rover module in the "rv" namespace. As a result, all function names pasted from the menus will be preceded by "rv.".

Drive Menu

I/O Input Menu

I/O Output Menu

I/O Path Menu

Settings

Commands Menu

The ti_rover module is implemented as a Python library and is built as a frozen Python module embedded in the Python firmware distribution.

The import of the ti_rover module into a script performs both the TI-Innovator™Hub presence check and a TI-Rover presence check. If the TI-Innovator™Hub is not present, an exception of 'TI-Innovator is not present.' is generated. If the Hub is present, but the TI-Rover is not detected as present during the CONNECT RV process, then an exception of 'TI-Rover is not present.' is generated (most common cause of this is the I2C cable of the TI-Rover is not properly connected.).

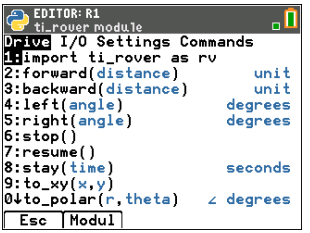
In normal use cases, the ti_rover module is imported in a script as "import ti_rover as rv".

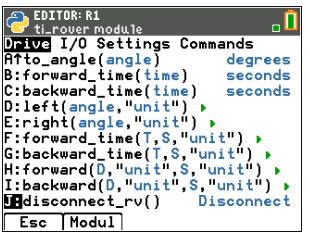
The library implements the functions defined in the tables below.

Drive Menu

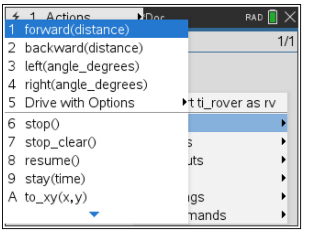
The CE menus and TI-Nspire™ CX II menus vary. Screen shots of each are provided.

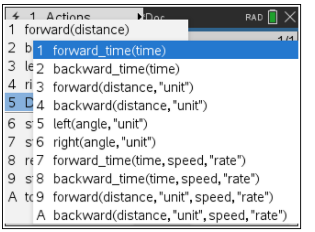
CE menus

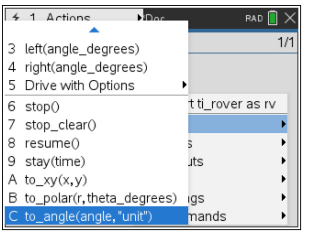




TI-Nspire™ CX II menus







Drive Commands

rv.forward

Command:	**rv.forward()**
Command Syntax:	**rv.forward([distance])** **rv.forward(distance, [rate])** **rv.forward(distance, rate, speed, [rate])**

Python Code Samples:	`rv.forward(0.5)`
	`rv.forward(0.5, "m")`
	`rv.forward(0.5, "m", 0.22, "m/s")`
	--- 	
	`rv.forward()`
	`rv.forward([distance])`
	`rv.forward([distance], ["m"|"units"|"revs"])`
	`rv.forward([distance], ["m"|"units"|"revs"], s.ss)`
	`rv.forward([distance], ["m"|"units"|"revs"], s.ss, ["m/s"|"units/s"|"revs/s"])`

Range:	N/A
Describe:	RV moves forward a given distance (default 0.75 m). Default distance if specified is in UNIT (grid units). Optional M=meters, UNIT=grid-unit, REV=wheel-revolution. Default speed is 0.20 m/sec, max is 0.23 m/sec, min is 0.14 m/sec. Speed may be given and specified in meters/second, unit/second, revolutions/second.
Result:	Action to make the RV move in a forward direction
Type or Addressable Component:	Control **Note:** This Rover control command is sent and executed in a queue.

rv.forward_time

Command:	.forward_time()
Command Syntax:	**rv.forward_time([time])** **rv.forward_time (time, speed, [rate])**

Python Code Samples:	`rv.forward_time(10)`
	`rv.forward_time(10, 0.22, "m/s")`
	--- 	
	`rv.forward_time([TIME])`
	`rv.forward_time([TIME], [SPEED])`
	`rv.forward_time([TIME], [SPEED], ["m/s"|"units/s"|"revs/s"])`

Range:	N/A
Describe:	RV moves forward a given distance (default 0.75 m). Default distance if specified is in UNIT (grid units). Optional M=meters, UNIT=grid-unit, REV=wheel-revolution. Default speed is 0.20 m/sec, max is 0.23 m/sec, min is 0.14 m/sec. Speed may be given and specified in meters/second, unit/second, revolutions/second. Default speed unit is meters/second.
Result:	Action to make the RV move in a forward direction
Type or Addressable Component:	Control **Note:** This Rover control command is sent and executed in a queue.

rv.backward

Command:	**.backward()**
Command Syntax:	**rv.backward ([distance])** **rv.backward (distance, [rate])** **rv.backward(distance, rate, speed, [rate])**

Python Code Samples:	`rv.backward(0.5)`
	`rv.backward(0.5, "m")`
	`rv.backward(0.5, "m", 0.22, "m/s")`
	--- 	
	`rv.backward()`
	`rv.backward([distance])`
	`rv.backward([distance], ["m"|"units"|"revs"])`
	`rv.backward([distance], ["m"|"units"|"revs"], s.ss)`
	`rv.backward([distance], ["m"|"units"|"revs"], s.ss, ["m/s"|"units/s"|"revs/s"])`

Range:	N/A
Describe:	RV moves backward a given distance (default 0.75 m). Default distance if specified is in UNIT (grid units). Optional M=meters, UNIT=grid-unit, REV=wheel-revolution. Default speed is 0.20 m/sec, max is 0.23 m/sec, min is 0.14 m/sec. Speed may be given and specified in meters/second, unit/second, revolutions/second. Default speed unit is meters/second.
Result:	
Type or Addressable Component:	Control **Note:** This Rover control command is sent and executed in a queue.

rv.backward_time

Command:	.backward_time()
Command Syntax:	**rv.backward_time([time])** **rv.backward_time (time, speed, [rate])**

Python Code Samples:	`rv.backward_time(10)`
	`rv.backward_time(10, 0.22, "m/s")`
	--- 	
	`rv.backward_time([TIME])`
	`rv.backward_time([TIME], [SPEED])`
	`rv.backward_time([TIME], [SPEED], ["m/s"|"units/s"|"revs/s"])`

Range:	N/A
Describe:	RV moves backward a given distance (default 0.75 m). Default distance if specified is in UNIT (grid units). Optional M=meters, UNIT=grid-unit, REV=wheel-revolution. Default speed is 0.20 m/sec, max is 0.23 m/sec, min is 0.14 m/sec. Speed may be given and specified in meters/second, unit/second, revolutions/second. Default speed unit is meters/second. .
Result:	Action to make the RV move in a backward direction.
Type or Addressable Component:	Control **Note:** This Rover control command is sent and executed in a queue.

rv.left

Command:	.left()
Command Syntax:	**rv.left([angle])** **rv.left(angle, [unit])**

Python Code Samples:	`rv.left(90)`
	--- 	
	`rv.left(ddd, ["degrees"])`
	`rv.left(rrr, ["radians"])`
	`rv.left(ggg, ["grads"])`

Range:	N/A
Describe:	Default turn is 90 degrees unless DEGREES, RADIANS, or GRADIANS keyword is present, and then the value is converted internally to degrees format from the specified units. Value given is ranged to a value between 0.0 and 360.0 degrees. The turn will be executed as a SPIN motion.
Result:	Turn Rover to the LEFT.
Type or Addressable Component:	Control **Note:** This Rover control command is sent and executed in a queue.

rv.right

Command:	.right()
Command Syntax:	**rv.right([angle])** **rv.right(angle, [unit])**

Python Code Samples:	`rv.right(90)`
	--- 	
	`rv.right(ddd, ["degrees"])`
	`rv.right(rrr, ["radians"])`
	`rv.right(ggg, ["grads"])`

Range:	N/A
Describe:	Default turn is 90 degrees unless "degrees", "radians", or "grads" keyword is present, and then the value is converted internally to degrees format from the specified units. Value given is ranged to a value between 0.0 and 360.0 degrees. The turn will be executed as a SPIN motion. .
Result:	Turn Rover to the RIGHT.
Type or Addressable Component:	Control **Note:** This Rover control command is sent and executed in a queue.

rv.stop

Command:	.stop()
Command Syntax:	**rv.stop()** **rv.stop_clear()**
Python Code Samples:	`rv.stop()`
Range:	N/A
Describe:	The **RV** will stop any current movement immediately. That movement can be resumed from where it left off with a **RESUME** operation. Any movement commands will cause the queue to flush immediately, and begin the just-posted new movement operation.
Result:	Stop processing Rover commands from the command queue, and leave pending operations in the queue (immediate action). Queue can be resumed by **resume()**.The **RV** will stop any current movement immediately. That movement can be resumed from where it left off with a **resume()** operation. Any movement commands will cause the queue to flush immediately, and begin the just-posted new movement operation. Stop processing Rover commands from the command queue, and flush any pending operations left in the queue (immediate action).
Type or Addressable Component:	Control **Note:** This Rover control command is sent and executed in a queue.

rv.resume()

Command:	.resume()
Command Syntax:	**rv.resume()**
Python Code Samples:	`rv.resume()`
Range:	N/A
Describe:	Enable processing of Rover commands from the command queue (immediate action), or resume (see **rv.stay()** operation.
Result:	Resume operation

Command:	.resume()
Type or Addressable Component:	Control **Note:** This Rover control command is sent and executed in a queue.

rv.stay

Command:	.stay()
Command Syntax:	**rv.stay([time])**
Python Code Samples:	`rv.stay(5)`
Range:	N/A
Describe:	Tells RV to "stay" in place for an optionally specified amount of time in seconds. Default is 30.0 seconds.
Result:	RV stays in position
Type or Addressable Component:	Control **Note:** This Rover control command is sent and executed in a queue.

rv.to_xy

Command:	.to_xy()
Command Syntax:	**rv.to_xy([x,y])**
Python Code Samples:	`rv.to_xy(1, 1)`
Range:	-327 to +327 for X and Y coordinates
Describe:	This command controls the movement of Rover on a virtual grid. Default location at start of program execution is (0,0) with Rover facing the positive x-axis. The x and y coordinates match the current grid size (default: 0.1 M/grid unit). Grid size can be changed through **grid_m_unit()** command The speed parameter is optional.
Result:	Moves Rover from current grid location to the specified grid location
Type or Addressable Component:	Control **Note:** This Rover control command is sent and executed in a queue.

rv.to_polar

Command:	.to_polar()
Command Syntax:	**rv.to_polar([radius, theta])**
Python Code Samples:	`rv.to_polar(5, 30) - r = 5 units, theta = 30 degrees`
Range:	Theta-coordinate: -360 to +360 degrees R-coordinate: -327 to +327
Describe:	Moves the RV from its current position to the specified polar position relative to that position. The RV's X/Y position will be updated to reflect the new position. The "r" coordinate matches the current grid size (default: 0.1 M/grid unit). Default location at start of program execution is (0, 0) with Rover facing the positive x-axis. Default unit of theta is Degrees. The speed parameter is optional.
Result:	Moves Rover from current grid location to the specified grid location.
Type or Addressable Component:	Control **Note:** This Rover control command is sent and executed in a queue.

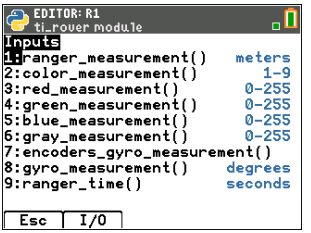
rv.to_angle

Command:	.to_angle()
Command Syntax:	**.to_angle([angle])** **.to_angle(angle, [unit])**
Python Code Samples:	`rv.to_angle(0)` `rv.to_angle(rr.rr, ["degrees"|"radians"|"gradians"])`
Range:	N/A
Describe:	
Result:	Spins the RV to the specified angle from current heading.
Type or Addressable Component:	Control **Note:** This Rover control command is sent and executed in a queue.

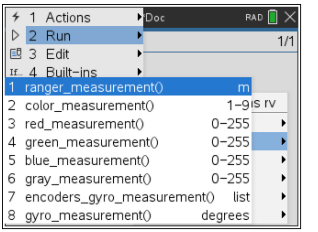
Inputs Menu

The CE menus and TI-Nspire™ CX II menus vary. Screen shots of each are provided.

CE menus



TI-Nspire™ CX II menus



Sensor Commands

rv.ranger_measurement()

Command:	**.ranger_measurment()**
Command Syntax:	**.ranger_measurement()**
Python Code Samples:	`r = rv.ranger_measurement()`
Range:	N/A
Describe:	The front-facing ultrasonic distance sensor. Returns measurements in meters. ~10.00 meters means no obstacle was detected.
Result:	Returns value in Meters.
Type or Addressable Component:	Sensor **Note:** This Rover sensor command is executed immediately.

rv.color_measurement():

Command:	**.color_measurement()**
Command Syntax:	**.color_measurement()**
Python Code Samples:	`c = rv.color_measurement()`
Range:	1 thru 9
Describe:	Bottom-mounted color sensor detects the color of the surface. Can also detect gray-level scale of black (0) to white (255).
Result:	Returns current color sensor information. The return value is in the 1 – 9 range which maps to the colors below: **Color** — **Return value** Red — 1 Green — 2 Blue — 3 Cyan — 4 Magenta — 5 Yellow — 6 Black — 7 White — 8 Gray — 9
Type or Addressable Component:	Sensor **Note:** This Rover sensor command is executed immediately.

rv.red_measurement()

Command:	.red_measurement()
Command Syntax:	**.red_measurement()**
Python Code Samples:	`r = rv.red_measurement()`
Range:	0 - 255
Describe:	Detect intensity of individual red components of surface. The results are in 0-255 range.
Result:	Returns current color sensor "red" value.
Type or Addressable Component:	Sensor **Note:** This Rover sensor command is executed immediately.

rv.green_measurement()

Command:	.green_measurement()
Command Syntax:	**.green_measurement()**
Python Code Samples:	`g = rv.green_measurement()`
Range:	0 - 255
Describe:	Detect intensity of individual green components of surface. The results are in 0-255 range.
Result:	Returns current color sensor "green" value.
Error:	
Type or Addressable Component:	Sensor **Note:** This Rover sensor command is executed immediately.

rv.blue_measurement():

Command:	.blue_measurement()
Command Syntax:	**.blue_measurement()**
Python Code Samples:	`b = rv.blue_measurement()`
Range:	0 - 255
Describe:	Detect intensity of individual blue components of surface. The results are in 0-255 range.
Result:	Returns current color sensor "blue" value.
Error:	
Type or Addressable Component:	Sensor **Note:** This Rover sensor command is executed immediately.

rv.gray_measurement():

Command:	.gray_measurement()
Command Syntax:	**.gray_measurement()**
Python Code Samples:	`gray = rv.gray_measurement()`
Range:	0 - 255
Describe:	Detect intensity of individual gray components of surface. The results are in 0-255 range.
Result:	Returns current color sensor "gray" value.
Error:	
Type or Addressable Component:	Sensor **Note:** This Rover sensor command is executed immediately.

rv.encoders_gyro_measurement()

Command:	.encoders_gyro_measurement()
Command Syntax:	**.encoders_gyro_measurement()**
Python Code Samples:	`L1 = rv.encoders_gyro_measurement()`
Range:	N/A
Describe:	The left and right encoder, coupled with the gyro and operating time information.
Result:	List of values of current left and right encoder, coupled with gyro and operating time information.
Type or Addressable Component:	Control **Note:** This Rover READ command is executed immediately.

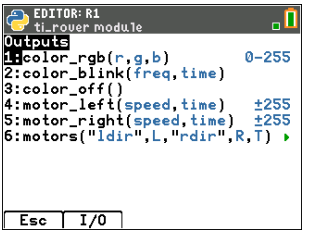
rv.gyro_measurement

Command:	.gyro_measurement()
Command Syntax:	**.gyro_measurement()**
Python Code Samples:	`rv.gyro_measurement()`
Range:	N/A
Describe:	The gyroscope is used to maintain the heading of Rover while it's in motion. It can also be used to measure the change in angle during turns. The gyroscope is ready to use after the CONNECT RV command is processed. The GYRO object shall be usable even when the RV is not in motion.
Result:	Returns current gyro sensor angular deviation from 0.0, reading partially drift-offset compensated.
Type or Addressable Component:	Control **Note:** This Rover READ command is executed immediately.

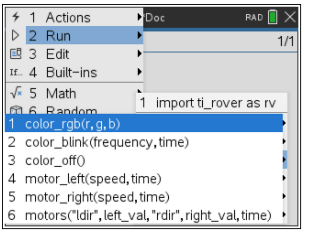
Outputs Menu

The CE menus and TI-Nspire™ CX II menus vary. Screen shots of each are provided.

CE menus



TI-Nspire™ CX II menus



Output Commands

rv.color_rgb()

Command:	**.color_rgb()**
Command Syntax:	**.color_rgb(red, green, blue)**
Python Code Samples:	`rv.color_rgb(255, 255, 255)`
Range:	N/A
Describe:	Set the RGB color to be displayed on the Rover's RGB LED. Same syntax as for all RGB LED operations with COLOR, etc.
Result:	Return the current RGB color, as a three-element list, that is being displayed on the Rover's RGB LED
Type or Addressable Component:	Control **Note:** This Rover control command is executed immediately.

rv.color_blink()

Command:	.color_blink()
Command Syntax:	**.color_blink(frequency, time)**
Python Code Samples:	`rv.color_rgb(255, 255, 10)` `rv.color_blink(2, 10)`
Range:	N/A
Describe:	Used as complement for rv.color_rgb(). Cause a blinking with the current color configuration. **frequency:** Range 0.1-20 **time:** Range 0.1-100
Result:	Blinks LED that is being displayed on the Rover's RGB LED
Type or Addressable Component:	Control **Note:** This Rover control command is executed immediately.

rv.color_off()

Command:	.color_off()
Command Syntax:	**.color_off(frequency, time)**
Python Code Samples:	`rv.color_rgb(255, 255, 10)` `rv.color_off()`
Range:	N/A
Describe:	Turns off RGB LED that is being displayed on the Rover's RGB LED.
Result:	N/A
Type or Addressable Component:	Control **Note:** This Rover control command is executed immediately.

rv.motor_left()

Command:	.motor_left()
Command Syntax:	**.motor_left(power, time)**
Python Code Samples:	`rv.motor_left(100, 2)`
Range:	left_wheel: from -255 to 255 time: from 0.1 to 100
Describe:	Set left motor direct PWM value. **CCW** = forward, **CW** = backward, pwm value negative = forward, positive = backward. **TIME** option available in all modes.
Result:	Left wheel motor and control for direct control (advanced) use.
Type or Addressable Component:	Control **Note:** This Rover control command is sent and executed in a queue.

rv.motor_right()

Command:	.motor_right()
Command Syntax:	**.motor_right(power, time)**
Python Code Samples:	`rv.motor_right(100, 2)`
Range:	right_wheel: from -255 to 255 time: from 0.1 to 100
Describe:	Set right motor direct PWM value. **CCW** = forward, **CW** = backward, pwm value negative = forward, positive = backward. **TIME** option available in all modes.
Result:	Right wheel motor and control for direct control (advanced) use.
Errors:	**Error:** Value out of range -255 to 255. **Error:** Time out of range.
Type or Addressable Component:	Control **Note:** This Rover control command is sent and executed in a queue.

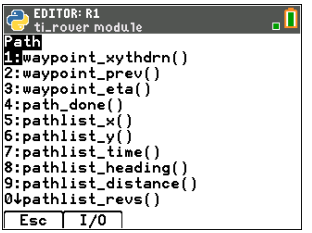
rv.motors()

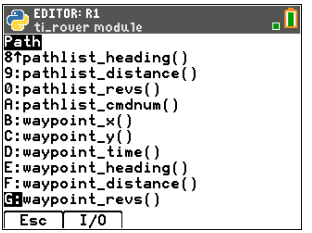
Command:	.motors()
Command Syntax:	**.motors("left_dir", left_wheel, "right_dir", "right_wheel", time)**
Python Code Samples:	`rv.motors("cw"|"ccw", left_wheel, "cw"|"ccw",right_wheel, time)`
Range:	left_wheel: from 0 to 255 right_wheel: from 0 to 255 time: from 0.1 to 100
Describe:	Set left or right or both motor PWM values. Negative values imply **CCW** and Positive values imply **CW**. Left **CW**=backward motion. Left **CCW**=forward motion. Right **CW**=forward motion, Right **CCW**=backward motion. PWM values may be numeric from 0 to255. Value of 0 is stop.
Result:	Both the LEFT and RIGHT motor, managed as a single object for direct control (advanced) use.
Type or Addressable Component:	Control **Note:** This Rover control command is sent and executed in a queue.

Path Menu

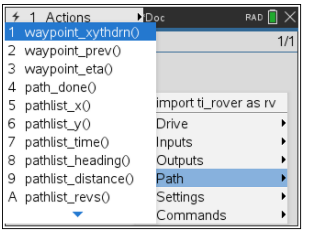
The CE menus and TI-Nspire™ CX II menus vary. Screen shots of each are provided.

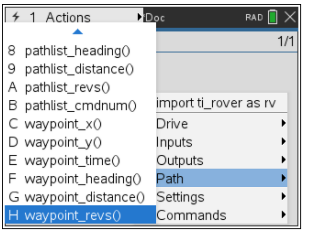
CE menus





TI-Nspire™ CX II menus





rv.waypoint_xythdrn()

Command:	**waypoint_xythdrn()**
Command Syntax:	**waypoint_xythdrn()**
Python Code Samples:	`rv.waypoint_xythdrn()`
Example:	Getting the distance traveled toward the current way-point from the last way-point.
Python Code Samples:	`L1 = rv.waypoint_xythdrn() D = L1[5]`
Range:	N/A
Describe:	Read the x-coord, y-coord, time, heading, distance traveled, number of wheel revolutions, command number of the current waypoint. Returns a list with all these values as elements. **Note:** Revs due to rotation commands such as rv.left() or rv.right() are not measured.
Result:	Return list of current way-point X, Y coordinates, Time, Heading, Distance, Revolutions, and command number.
Type or Addressable Component:	Returns Data

rv.waypoint_prev()

Command:	.waypoint_prev()
Command Syntax:	**.waypoint_prev()**
Python Code Samples:	`rv.waypoint_prev()`
Example:	Getting the distance traveled during the previous way-point.
Python Code Samples:	`L1 = rv.waypoint_prev()` `D = L1[5]`
Range:	N/A
Describe:	Read the x-coord, y-coord, time, heading, distance traveled, number of wheel revolutions, command number of the previous waypoint. Returns a list with all these values as elements.
Result:	Return list of the previous way-point X, Y coordinates, time, heading, distance, revolutions, and command number.
Type or Addressable Component:	Returns Data

rv.waypoint_eta()

Command:	.waypoint_eta()
Command Syntax:	**.waypoint_eta()**
Python Code Samples:	`time = rv.waypoint_eta()`
Range:	N/A
Describe:	Returns the estimated time to drive to a waypoint.
Result:	Returns the estimated time to drive to a waypoint.
Type or Addressable Component:	Returns Data

rv.path_done()

Command:	.path_done()
Command Syntax:	**.path_done()**
Python Code Samples:	`while not rv.path_done():` `  pass`
Range:	N/A
Describe:	Returns a value of 0 or1 depending on whether the Rover is moving (0) or finished with all movement (1).
Result:	Returns 0 or 1 indicating if Rover is moving (1) or if all path commands have been executed (0)
Errors:	
Type or Addressable Component:	Returns Data

rv.pathlist_x()

Command:	.pathlist_x()
Command Syntax:	**.pathlist_x()**
Python Code Samples:	`rv.pathlist_x()`
Example:	Program to plot the RV path on the graph screen.

Python Code Samples:	`rv.to_xy(1, 2)`
	`x = rv.pathlist_x()`
	`y = rv.pathlist_y()`
	
	`print("x = {}".format(x))`
	`print("y = {}".format(y))`

Range:	N/A
Describe:	Returns a list of X values from the beginning to end including the current Waypoint X value.

Command:	.pathlist_x()
Result:	Return list of X coordinates traversed since last **rv.path_clear()** or start of execution.
Type or Addressable Component:	Returns Data

rv.pathlist_y()

Command:	.pathlist_y()
Command Syntax:	**.pathlist_y()**
Python Code Samples:	`rv.pathlist_y()`
Example:	Program to plot the RV path on the graph screen.

Python Code Samples:	`rv.to_xy(1, 2)`
	`x = rv.pathlist_x()`
	`y = rv.pathlist_y()`
	
	`print("x = {}".format(x))`
	`print("y = {}".format(y))`

Range:	N/A
Describe:	Returns a list of Y values from the beginning to end including the current Waypoint Y value.
Result:	Return list of Y coordinates traversed since last **rv.path_clear()** or start of execution.
Type or Addressable Component:	Returns Data

rv.pathlist_time()

Command:	.pathlist_time()
Command Syntax:	**.pathlist_time()**
Python Code Samples:	`rv.pathlist_time()`
Range:	N/A
Describe:	Returns a list of the time in seconds from the beginning to and including the current Waypoint time value.
Result:	Return list of cumulative travel times for each successive way-point.
Type or Addressable Component:	Returns Data

rv.pathlist_heading()

Command:	.pathlist_heading()
Command Syntax:	**.pathlist_heading()**
Python Code Samples:	`rv.pathlist_heading()`
Range:	N/A
Describe:	Returns a list of the headings from the beginning to and including the current Waypoint heading value.
Result:	Return list of cumulative angular headings taken.
Type or Addressable Component:	Returns Data

rv.pathlist_distance()

Command:	.pathlist_distance()
Command Syntax:	**.pathlist_distance()**
Example:	Getting the cumulative distance traveled since the beginning of a journey by the RV.
Python Code Samples:	`d = rv.pathlist_distance()`
Range:	N/A
Describe:	Returns a list of the distances traveled from the beginning to and including the current Waypoint distance value.
Result:	Return list of cumulative distances traveled.
Type or Addressable Component:	Returns Data

rv.pathlist_revs()

Command:	.pathlist_revs()
Command Syntax:	**.pathlist_revs()**
Python Code Samples:	`rv.pathlist_revs()`
Range:	N/A
Describe:	Returns a list of the number of revolutions traveled from the beginning to and including the current Waypoint revolutions value. **Note:** Revs due to rotation commands such as rv.left() or rv.right() are not measured.
Result:	Return list of wheel revolutions traveled.
Type or Addressable Component:	Returns Data

rv.pathlist_cmdnum()

Command:	**.pathlist_cmdnum()**
Command Syntax:	**.pathlist_cmdnum()**
Python Code Samples:	`actions = rv.pathlist_cmdnum()`
Range:	N/A
Describe:	rv.pathlist.cmdnum() - returns a list of command numbers for the path.
Result:	Return list of commands used to travel to the current way-point entry. 0 - Start of Way-points (if first action is a STAY, then no START is given, but a STAY will be shown instead.) 1 - Travel forward 2 - Travel backward 3 - Left spin motion 4 - Right spin motion 5 - Left turn motion 6 - Right turn motion 7 - Stay (no motion) the time the RV stays at the current position is given in the TIME list. 8 - RV is currently in motion on this way-point traversal
Type or Addressable Component:	Returns Data

rv.waypoint_x()

Command:	.waypoint_x()
Command Syntax:	**.waypoint_x()**
Python Code Samples:	`x = rv.waypoint_x()`
Range:	N/A
Describe:	Returns x coordinate of current waypoint.
Result:	Return current way-point X coordinate.
Type or Addressable Component:	Returns Data

rv.waypoint_y()

Command:	.waypoint_y()
Command Syntax:	**.waypoint_y()**
Python Code Samples:	`y = rv.waypoint_y()`
Range:	N/A
Describe:	Returns y coordinate of current waypoint.
Result:	Return current way-point Y coordinate.
Errors:	
Type or Addressable Component:	Returns Data

rv.waypoint_time()

Command:	.waypoint_time()
Command Syntax:	**.waypoint_time()**
Python Code Samples:	`time = rv.waypoint_time()`
Range:	N/A
Describe:	Returns time spent traveling from previous to current waypoint.
Result:	Return total cumulative way-point travel time value in seconds.
Type or Addressable Component:	Returns Data

rv.waypoint_heading ()

Command:	.waypoint_heading ()
Command Syntax:	**.waypoint_heading ()**
Python Code Samples:	`heading = rv.waypoint_heading()`
Range:	N/A
Describe:	Returns the absolute heading of the current waypoint.
Result:	Return current absolute heading in degrees. (+h = counter-clockwise, -h = clockwise).
Type or Addressable Component:	Returns Data

rv.waypoint_distance()

Command:	.waypoint_distance()
Command Syntax:	**.waypoint_distance()**
Python Code Samples:	`distance = rv.waypoint_distance()`
Range:	N/A
Describe:	Returns the distance traveled between previous and current waypoint.
Result:	Return cumulative total distance traveled in meters.
Type or Addressable Component:	Returns Data

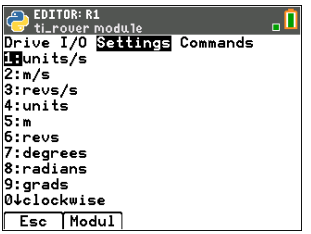
rv.waypoint_revs()

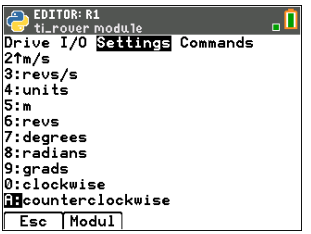
Command:	.waypoint_revs()
Command Syntax:	**.waypoint_revs()**
Python Code Samples:	`revs = rv.waypoint_revs()`
Range:	
Describe:	Returns number of revolutions needed to travel between previous and current waypoint. **Note:** Revs due to rotation commands such as rv.left() or rv.right() are not measured.
Result:	Return total revolutions of the wheels performed to travel the cumulative distance to the current way-point.
Errors:	
Type or Addressable Component:	Returns Data

Settings Menu

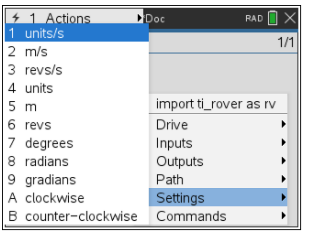
The CE menus and TI-Nspire™ CX II menus vary. Screen shots of each are provided.

CE menus





TI-Nspire™ CX II menus



Settings

units/s

Command:	units/s
Command Syntax:	**units/s**
Method:	rv.units/s
Range:	
Describe:	Option for speed in grid units per second.
Result:	
Errors:	
Type or Addressable Component:	Setting

ms/s

Command:	ms/s
Command Syntax:	**ms/s**
Method:	rv.ms/s
Range:	
Describe:	Option for speed in meters per second.
Result:	
Errors:	
Type or Addressable Component:	Setting

revs/s

Command:	revs/s
Command Syntax:	**revs/s**
Method:	rv.revs/s
Range:	
Describe:	Option for speed in wheel revolutions per second.
Result:	
Errors:	
Type or Addressable Component:	Setting

units

Command:	units
Command Syntax:	**units**
Method:	rv.units
Range:	
Describe:	Option for distance in grid units.
Result:	
Errors:	
Type or Addressable Component:	Setting

m

Command:	m
Command Syntax:	**m**
Method:	rv.m
Range:	
Describe:	Option for distance in meters.
Result:	
Errors:	
Type or Addressable Component:	Setting

revs

Command:	revs
Command Syntax:	**revs**
Method:	rv.revs
Range:	
Describe:	Option for distance in wheel revolutions.
Result:	
Errors:	
Type or Addressable Component:	Setting

degrees

Command:	degrees
Command Syntax:	**degrees**
Method:	rv.degrees
Range:	
Describe:	Option for turning in degrees.
Result:	
Errors:	
Type or Addressable Component:	Setting

radians

Command:	radians
Command Syntax:	**radians**
Method:	rv.radians
Range:	
Describe:	Option for turning in radians.
Result:	
Errors:	
Type or Addressable Component:	Setting

gradians

Command:	gradians
Command Syntax:	**gradians**
Method:	rv.gradians
Range:	
Describe:	Option for turning in gradians.
Result:	
Errors:	
Type or Addressable Component:	Setting

clockwise

Command:	clockwise
Command Syntax:	**clockwise**
Method:	rv.clockwise
Range:	
Describe:	Option for specifying wheel direction.
Result:	
Errors:	
Type or Addressable Component:	Setting

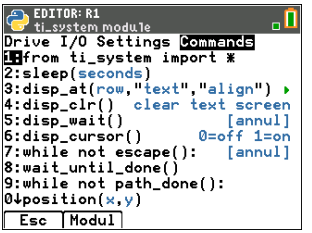
counter-clockwise

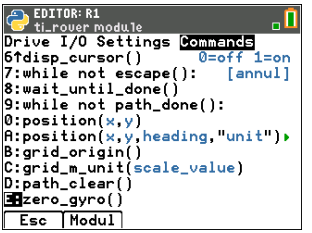
Command:	counter-clockwise
Command Syntax:	**counter-clockwise**
Method:	rv.counter-clockwise
Range:	
Describe:	Option for specifying wheel direction.
Result:	
Errors:	
Type or Addressable Component:	Setting

Commands Menu

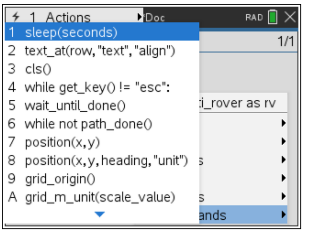
The CE menus and TI-Nspire™ CX II menus vary. Screen shots of each are provided.

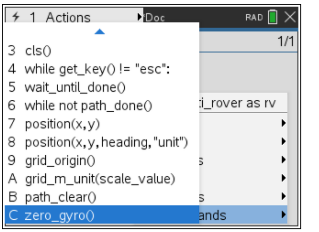
CE menus





TI-Nspire™ CX II menus





Commands

These commands are collection of functions from from ti_system import* modules as well as from the TI Rover module.

sleep(seconds)

Command:	**sleep(seconds)**
Command Syntax:	sleep(seconds)
Module:	ti_system
Describe:	Sleep for a given number of seconds. Seconds argument is a float.

Python Code Sample:	`from time import *`
	`a=monotonic()`
	`sleep(15)`
	`b=monotonic()`
	`print(b-a)`
	
	Run the program TIME
	`>>>15.0`

Result:	Sleep for a given number of seconds. Seconds argument is a float.
Error:	
Type or Addressable Component:	**from ti_system import***

disp_at(row,col,"text")

Command:	**disp_at(row,col,"text")**
Command Syntax:	disp_at(row,col,"text")
Module:	ti_system
Describe:	Display text starting at a row and column position on the plotting area. REPL with cursor >>>| will appear after text if at end of program. Use disp_cursor() to control cursor display.

Python Code Sample:	`from ti_system import *`
	`disp_clr() #clears Shell screen`
	`disp_at(5,6,"hello")`
	`disp_cursor(0)`
	`disp_wait()`

Result:	Display text starting at a row and column position on the plotting area.
Error:	
Type or Addressable Component:	**from ti_system import***

disp_clr()

Command:	**disp_clr()** **clear text screen**
Command Syntax:	disp_clr() clear text screen
Module:	ti_system
Describe:	Clear the screen in the Shell environment. Row 0-11, integer may be used as an optional argument to clear a display row of the Shell environment.

Python Code Sample:	`from ti_system import *`
	`disp_clr() #clears Shell screen`
	`disp_at(5,"hello","left")`
	`disp_cursor(0)`
	`disp_wait()`

Result:	Clear the screen in the Shell environment. Row 0-11, integer may be used as an optional argument to clear a display row of the Shell environment.
Error:	
Type or Addressable Component:	**from ti_system import***

disp_wait()

Command:	**disp_wait()** **[clear]**
Command Syntax:	disp_wait() [clear]
Module:	ti_system
Describe:	Stop the execution of program at this point and display screen content until [clear] is pressed and the screen is cleared.

Python Code Sample:	`from ti_system import *`
	`disp_clr() #clears Shell screen`
	`disp_at(5,"hello","left")`
	`disp_cursor(0)`
	`disp_wait()`

Result:	Stop the execution of program at this point and display screen content until [clear] is pressed and the screen is cleared.
Error:	
Type or Addressable Component:	**from ti_system import***

disp_cursor()

Command:	**disp_cursor()** **0=off 1=on**
Command Syntax:	disp_cursor() 0=off 1=on
Module:	ti_system
Describe:	Control the display of the cursor in the Shell when a program is running.

Python Code Sample:	`from ti_system import *`
	`disp_clr() #clears Shell screen`
	`disp_at(5,"hello","left")`
	`disp_cursor(0)`
	`disp_wait()`

Result:	Control the display of the cursor in the Shell when a program is running.
Error:	
Type or Addressable Component:	**from ti_system import***

while not escape(): clear

Command:	**while not escape(): clear**
Command Syntax:	while not escape(): clear
Module:	ti_system
Describe:	In a while loop running in a program where the program offers to end the loop but keep the script running. Run the commands in the "while" loop until the "esc" key is pressed..
Python Code Sample:	
Result:	
Error:	
Type or Addressable Component:	**from ti_system import***

rv.wait_until_done()

Command:	.wait_until_done()
Command Syntax:	**.wait_until_done()**

Python Code Sample:	`rv.wait_until_done()`
	
	Sample program
	for i in range(4):
	••`rv.forward(.5,"M")`
	••`rv.left()`
	••`rv.wait_until_done()`
	••`rv.color_rgb(255,0,0)`
	••`sleep(1)`
	••`rv.color_off()`
	
	Note: This program works well for the task of turning on the Rover's LED to red for 1 second and then off after each left turn in the square.

Range:	N/A
Describe:	Program waits until TI-Rover stops moving.
Result:	Program waits until TI-Rover stops moving.
Type or Addressable Component:	Setting

while not path_done()

Command:	while not path_done()
Command Syntax:	**while not path_done()**
Method:	rv.while not path_done()
Range:	
Describe:	Runs the commands in the "while" loop until the Rover is finished with all movement.
Result:	The path_done() function returns a value of 0 or 1 depending on whether the Rover is moving (0) or finished with all movement (1).
Errors:	
Type or Addressable Component:	

rv.position()

Command:	.position()
Command Syntax:	**.position(x,y)** **.position(x,y,heading,"unit")**
Python Code Samples:	`rv.position(xxx, yyy [, hhh, "degrees"|"radians"|"grads"])`
Range:	N/A
Describe:	Sets the coordinate position and optionally the heading of the Rover on the virtual grid.
Result:	Rover configuration is updated.
Type or Addressable Component:	Setting

rv.grid_origin()

Command:	.grid_origin()
Command Syntax:	**.grid_origin()**
Python Code Samples:	`rv.grid_origin()`
Range:	N/A
Describe:	Sets RV as being at current grid origin point of (0,0). The "heading" is set to 0.0 resulting in the current position of the RV now set to pointing down a virtual x-axis toward positive x values.
Result:	
Type or Addressable Component:	Setting

rv.grid_m_unit()

Command:	.grid_m_unit()
Command Syntax:	**.grid_m_unit(scale_value)**
Python Code Samples:	`rv.grid_m_unit(nnn)`
Range:	N/A
Describe:	Set the size of a “grid unit” on the virtual grid. Default is 0.1 meters per unit grid. A scale_value of 0.2 means 5 units per meter (20 cm per unit grid). A scale_value of 0.05 means 20 units per meter (5 cm per unit grid). Set the size of a "grid unit" on the virtual grid. Default is 10 units per meter (100 mm / 10 cm per unit grid). A value of 5 means 5 units per meter or 200 mm / 20 cm per unit grid). A value of 20 means 20 units per meter, or 50 mm / 5 cm per.
Result:	
Type or Addressable Component:	Setting

rv.path_clear()

Command:	.path_clear()
Command Syntax:	**.path_clear()**
Python Code Samples:	`rv.path_clear()`
Range:	N/A
Describe:	Clears any pre-existing path / waypoint information. Recommended before doing a sequence of movement operations where waypoint / path-list information.
Result:	
Type or Addressable Component:	Setting

rv.zero_gyro()

Command:	.zero_gyro()
Command Syntax:	**.zero_gyro()**
Python Code Samples:	`rv.zero_gyro()`
Range:	N/A
Describe:	Resets the Rover gyro to 0.0 angle and clears the left and right wheel encoder counts.
Result:	
Error:	
Type or Addressable Component:	Setting

TI-SensorLink Adapter Using VERNIER Commands with Python Programs

Stainless Steel Temperature Probe with Python

Python Command	
Sketch Object	VERNIER
Command Syntax	

Python Code Sample:

Desired Action	Python Code Sample
Read the temperature from the attached Vernier sensor	`v1=vernier ("IN1","temperature")` `t1 = v1.measurement()`

pH Sensor with Python

Python Command	
Sketch Object	VERNIER
Command Syntax	

Python Code Sample:

Desired Action	Python Code Sample
Read the pH from the attached Vernier sensor	`v1 =vernier("IN1","pH")` `ph = v1.measurement()`

Gas Pressure Sensor with Python

Python Command	
Sketch Object	VERNIER
Command Syntax	
Python Code Sample:	

Desired Action	Python Code Sample
Read the gas pressure from the attached Vernier sensor	`v1=vernier ("IN1","pressure")` `p = v1.measurement()`

Dual-Range Force Sensor with Python

Python Command	
Sketch Object	VERNIER
Command Syntax	
Python Code Sample:	

Desired Action	Python Code Sample
Read the force from the attached Vernier sensor in 10 N configuration	`v1=vernier ("IN1","force10")` `f1 = v1.measurement()`
Read the force from the attached Vernier sensor in 50 N configuration	`v1=vernier ("IN1","force50")` `f1 = v1.measurement()`

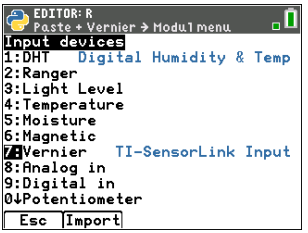
vernier() interface

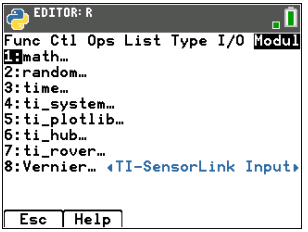
The Vernier sensors attached to a TI Sensor Link are accessible through the "vernier()" interface.

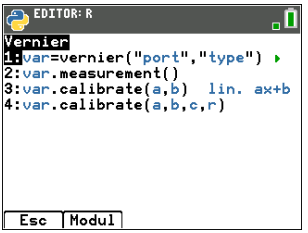
CE Family

CE family: Part of "input device" menu group in the "ti_hub" module.

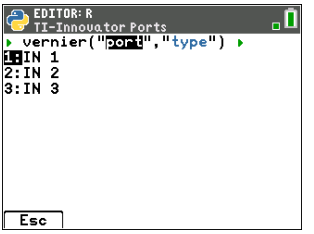
Once the 'from vernier import *' statement is in the program, a new menu for will be added to the 'Modules' menu. This menu has all the functions that can be used with Vernier sensors.

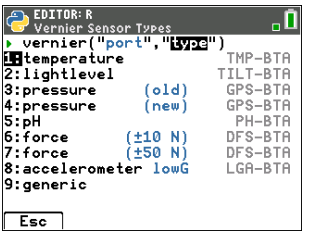


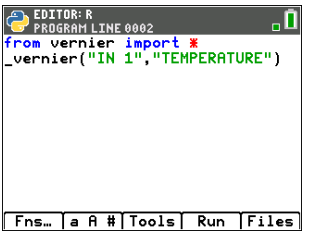




The "vernier()" function has two helper prompts – for the port and for the type of sensor attached.





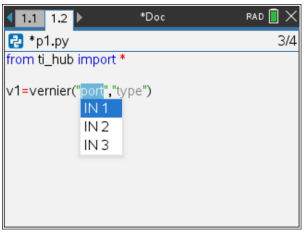


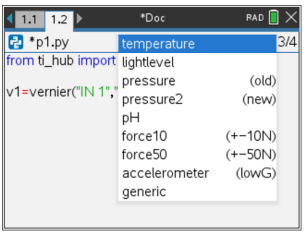
TI-Nspire CX II

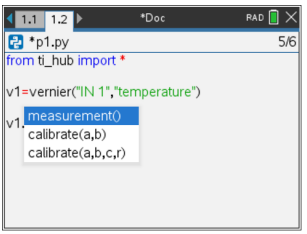
TI-Nspire CX II: The interface to Vernier sensors is part of the ti_hub module. To use this interface, the ti_hub module needs to be imported.

from ti_hub import *

Once the vernier() object has been initialized, all the associated functions can be chosen by type a “.” (period) after the variable name


1.1 1.2 *Doc RAD
*p1.py 3/4
from ti_hub import *
v1=vernier("port","type")
IN 1
IN 2
IN 3


1.1 1.2 *Doc RAD
*p1.py 3/4
from ti_hub import
v1=vernier("IN 1","
temperature
lightlevel
pressure (old)
pressure2 (new)
pH
force10 (+−10N)
force50 (+−50N)
accelerometer (lowG)
generic


1.1 1.2 *Doc RAD
*p1.py 5/6
from ti_hub import *
v1=vernier("IN 1","temperature")
v1.
measurement()
calibrate(a,b)
calibrate(a,b,c,r)

vernier()

Command:	vernier()
Command Syntax:	**v1 = vernier("port", "type")**

Python Code Sample:	Connect To a temperature sensor on connected to port IN1
	`v1=vernier("IN1","temperature")`
	`t1 = v1.measurement()`

Range:	N/A
Describe:	The **VERNIER** sensor object, provides a means of specifying the type of Vernier sensor being connected (and automatically applying the proper translation from voltage to sensor units). **VERNIER** sensors may be connected to the IN1, IN2 or IN3 port of the TI-Innovator™ Hub. **Note:** Support for three **VERNIER** objects is provided.
Result:	Connects the **VERNIER** Sensor to the TI-Innovator™ Hub. This establishes connections with the sensor.
Type or Addressable Component:	

.measurement()

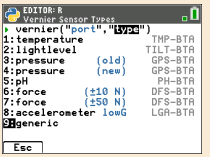
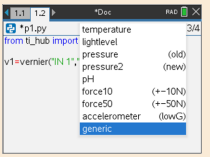
Command:	.measurement()
Command Syntax:	**.measurement()**

Python Code Sample:	Connect To a temperature sensor on connected to port IN1 and measure the temperature
	`v1=vernier("IN1","temperature")`
	`t1 = v1.measurement()`

Range:	Depends on the sensor being used.
Describe:	Returns the sensor data. The unit of the data depends on the specific sensor being used.
Result:	
Type or Addressable Component:	Vernier sensors through the TI Sensor Link adapter

.calibrate(a,b)

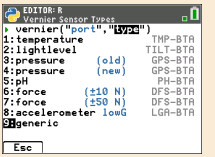
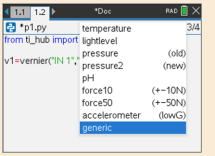
Command:	**.calibrate(a,b)**
Command Syntax:	**.calibrate(a,b)**

Python Code Sample:	`v1 = vernier("IN2"," ")`
	`v1.calibrate(16.325,0.0)`
	To use the calibrate function, pick the 'generic' sensor. These calibration values are for the Salinity sensor (SAL-BTA).
	CE family:
	 
	TI-Nspire CX II:
	 

Range:	
Describe:	Sets a linear calibration equation of ax+b on the specified sensor. Note that this is needed only when using a sensor that's not in the list of supported sensors.
Result:	The vernier() interface is calibrated with the new coefficients.
Type or Addressable Component:	TI Sensor Link

.calibrate(a,b,c,r)

Command:	**.calibrate(a,b,c,r)**
Command Syntax:	**.calibrate(a,b,c,r)**

Python Code Sample:	`v1 = vernier("IN2"," ")`
	`v1.calibrate(1.333342e-7, 2.22468e-4, 1.02119e-3)`
	To use the calibrate function, pick the 'generic' sensor.
	CE family:
	 
	TI-Nspire CX II:
	 

Range:	
Describe:	Sets a Steinhart calibration equation using a,b,c coefficients, and r as the resistance. Typically used with a thermistor. Note that this is needed only when using a sensor that's not in the list of supported sensors.
Result:	The vernier() interface is calibrated with the new coefficients.
Type or Addressable Component:	TI Sensor Link

TI-RGB Array Commands Using Python Programs

To use the TI-RGB Array in a Python program, you will need to import the right modules.

- TI-Nspire CX II: from `ti_hub import *`
- CE family: from `rgb_arr import *`

These modules provide an object oriented interface to the TI-RGB Array.

The program will have to first create an object for the TI-RGB Array and use the available functions to interact with it.

For additional commands see: education.ti.com/eguide

rgb_array()

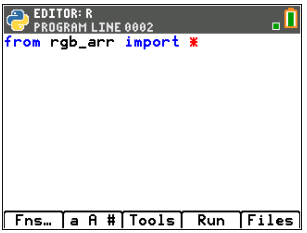
Command:	rgb_array()
Command Syntax:	**var = rgb_array()**
Python Code Sample:	`a1 = rgb_array()`
Range:	N/A
Describe:	This command configures the TI-Innovator™ Hub software to work with the TI-RGB Array.
Result:	Connects the TI-RGB Array to the TI-Innovator™ Hub. The TI-RGB Array is now ready to be programmed
Type or Addressable Component:	All components of the TI-RGB Array

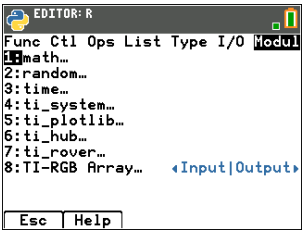
rgb_array("lamp")

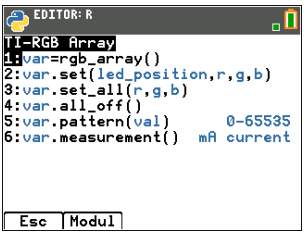
Command:	rgb_array("lamp")
Command Syntax:	**var = rgb_array("lamp")**
Python Code Sample:	`a1 = rgb_array("lamp")`
Range:	N/A
Describe:	This command will enable the "high brightness" mode of the TI-RGB Array as long as an external power source (like the USB battery) is connected to the **PWR** port. **Note:** "lamp" will need to be typed in.
Result:	The TI-RGB Array is now configured to be in high-brightness mode. If the external power is not connected, the "**LAMP**" has no effect – i.e. the brightness will be at the default level. Also note, an error will be indicated by a beep tone.
Type or Addressable Component:	All components of the TI-RGB Array

CE Family

CE family: Once the 'from rgb_arr import *' statement is in the program, a new menu for the TI-RGB Array will be added to the 'Modules' menu.

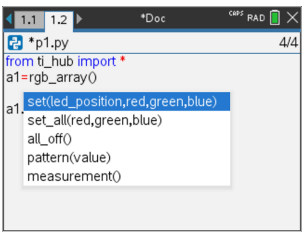






TI-Nspire CX II

TI-Nspire CX II: Once the rgb_array() object has been initialized, all the associated functions can be chosen by type a "." (period) after the variable name.



.set(n,r,g,b)

Command:	.set(led_position, red, green, blue)
Command Syntax:	**var = rgb_array("lamp")**
Python Code Sample:	`.set(n, r, g, b)` `.set( eval(n), r, g, b)`
Range:	0-15 for 'n', 0-255 for r,g,b
Describe:	The .set() command controls the brightness and color of each RGB LED in the TI-RGB Array.
Result:	The specific LED lights up with the specified color.
Type or Addressable Component:	All components of the TI-RGB Array See Also: set_all()

.set_all(r,g,b)

Command:	.set_all(r,g,b)
Command Syntax:	**.set_all(r,g,b)**
Python Code Sample:	`a1 = rgb_array()` `a1.set_all(255, 0, 255)`
	`a1.set_all(0,0,0)`
	`a1.set_all( eval(r), eval(g), eval(b))`

all_off()

Command:	all_off()
Command Syntax:	**all_off()**
Python Code Sample:	`a1.all_off()`
Range:	
Describe:	To control all the LEDs in a single command use: set_all(r, g, b).
Result:	Control all LEDs in a single command.
Type or Addressable Component:	All components of the TI-RGB Array See Also: set_all()

.pattern()

Command:	.pattern()
Command Syntax:	**.pattern(value)**
Python Code Sample:	`a1 = rgb_array()` `a1.pattern(value)`
Range:	value: 0 – 65535
Describe:	Displays the number specified by 'value' on the TI-RGB Array. The value can be specified in decimal or hexadecimal.
Result:	All components of the TI-RGB Array
Type or Addressable Component:	All components of the TI-RGB Array See Also: set_all()

.measurement()

Command:	**.measurement()**
Command Syntax:	**.measurement()**
Python Code Sample:	`a1 = rgb_array()` `current = a1.measurement()`
Range:	
Describe:	Returns the value of the current consumed by the TI-RGB Array in mA.
Result:	All components of the TI-RGB Array
Type or Addressable Component:	All components of the TI-RGB Array

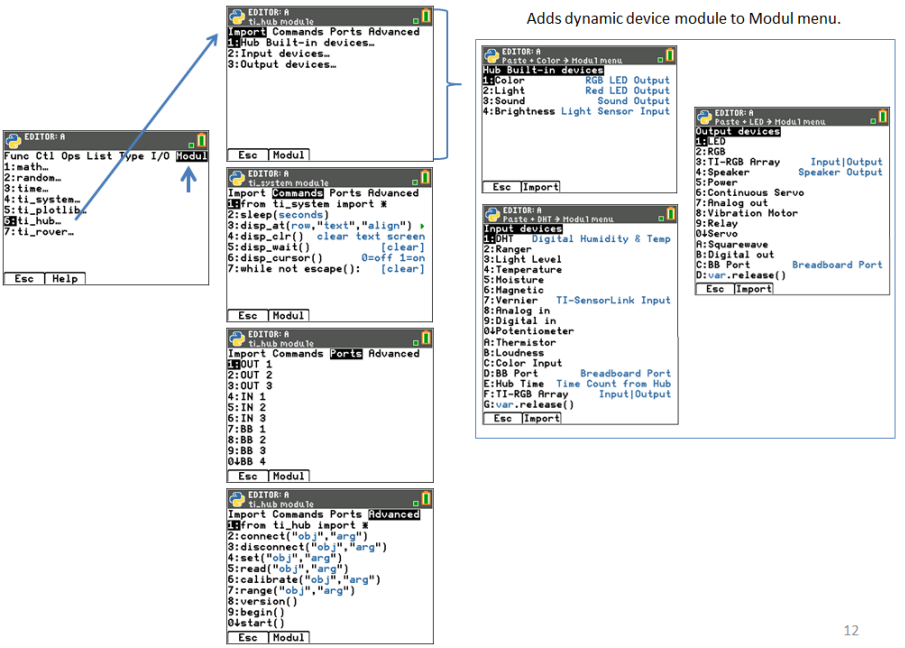
Appendix

Floating Point Values and Digits after Decimal Point

Python ti_hub Module

As it appears in the CE Graphing Calculators

[Fns…]>Modul: ti_hub module



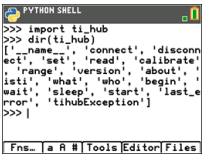
12

The **ti_hub** module contains the low-level command interfaces to send commands directly to the TI-Innovator™ Hub. The tables below show the function and command interfaces provided by this module.

Command:	**from ti_hub import ***
Command Syntax:	**from ti_hub import ***
Method:	
Describe:	Imports all methods from the ti_hub module. **Note:** When creating a new program that uses this module, it is recommended to use the **Hub Project** program type. This will ensure that all the relevant modules are imported.

Command:	from ti_hub import *
Result:	
Error:	
Type or Addressable Component:	

CE Calculators **TI-Nspire™ CX**



See: TI-Hub Python Commands

connect()

Command:	connect("obj","arg")
Command Syntax:	**connect**("obj","arg")
Method:	connect("obj","arg")
Object:	String that describes the TI-Innovator™ object to be connected.
Argument:	Contains a string with possible index number and any additional connect parameters required such as ports, etc. This parameter may be an empty string if no additional information is needed.
Range:	
Describe:	Connects a previously connected sensor or control from the TI-Innovator™.
Result:	
Error:	
Type or Addressable Component:	

disconnect()

Command:	**disconnect("obj","arg")**
Command Syntax:	**disconnect("obj","arg")**
Method:	disconnect("obj","arg")
Object:	String that describes the TI-Innovator™ object to be disconnected.
Argument:	Contains a string with possible index number. This parameter may be an empty string if no additional information is needed..
Range:	
Describe:	Disconnects a previously connected sensor or control from the TI-Innovator™.
Result:	
Error:	
Type or Addressable Component:	

set()

Command:	**set("obj","arg")**
Command Syntax:	**set**("**obj**","**arg**")
Method:	set("obj","arg")
Object:	String that describes the TI-Innovator™ object that the **SET** command will be directed to.
Argument:	Contains a string with additional information needed for the **SET** operation such as index number of object, value to set, and other parameter values needed based on the operation to perform.
Range:	
Describe:	Sends a value or set of values to a connected sensor or control, such as setting a pin value high or low, turning on an **LED**, etc.
Result:	
Error:	
Type or Addressable Component:	

read()

Command:	**read("obj","arg")**
Command Syntax:	**read("obj","arg")**
Method:	read("obj","arg")
Object:	String that describes the TI-Innovator™ object that the **READ** command will be directed to.
Argument:	Contains a string with additional information needed for the **READ** operation such as index number of object, and other parameter values needed based on the value to read.
Range:	
Describe:	Read a sensor value or other connected sensor or control value or property. Also can be used to read TI-Innovator™ specific information.
Result:	
Error:	
Type or Addressable Component:	

calibrate()

Command:	**calibrate("obj", "arg")**
Command Syntax:	**calibrate**("**obj**", "**arg**")
Method:	calibrate("obj","arg")
Object:	String that describes the TI-Innovator™ object that the **CALIBRATE** command will be directed to.
Argument:	Contains a string with additional information needed for the **CALIBRATE** operation such as index number of object, and the coefficient values to be set for the calibration operation.
Range:	
Describe:	Sends calibration coefficients to sensors that require calibration values or can have their default calibration values adjusted.
Result:	
Error:	
Type or Addressable Component:	

range()

Command:	**range("obj","arg")**
Command Syntax:	**range("obj","arg")**
Method:	range("obj","arg")
Object:	String that describes the TI-Innovator™ object that the **RANGE** command will be directed to.
Argument:	Contains a string with additional information needed for the **RANGE** operation such as index number of object, and the minimum and maximum range values to be set.
Range:	
Describe:	Sets a mapping of an input sensors ADC reading to a different range.
Result:	
Error:	
Type or Addressable Component:	

version()

Command:	version get(string) return string
Command Syntax:	**version** get(string) return string
Method:	version()
Object:	
Argument:	
Range:	
Describe:	Returns the current TI-Innovator™ Hub "sketch" firmware version information.
Result:	
Error:	
Type or Addressable Component:	

begin()

Command:	begin Wait 0.5 seconds get(string) "READY" return string
Command Syntax:	**begin** Wait 0.5 seconds get(string) "READY" return string
Method:	begin()
Object:	
Argument:	
Range:	
Describe:	Perform a "soft-reset" of the TI-Innovator™ Hub, disconnects all user sensors and controls, and resets all values to defaults.
Result:	
Error:	
Type or Addressable Component:	

about()

Command:	about get(string) return string
Command Syntax:	**about** get(string) return string
Method:	about()
Object:	
Argument:	
Range:	
Describe:	Retrieve copyright information on the TI-Innovator™ Hub firmware as a string.
Result:	
Error:	
Type or Addressable Component:	

isti()

Command:	**isti get(string): "TISTEM" return string**
Command Syntax:	**isti** get(string): "TISTEM" return string
Method:	isti()
Object:	
Argument:	
Range:	
Describe:	Can be used to detect a TI-Innovator™ as present. A valid response of "**TISTEM**" to the "**ISTI**" command indicates a TI-Innovator™ connected.
Result:	
Error:	
Type or Addressable Component:	

what()

Command:	what get(string) return string
Command Syntax:	**what** get(string) return string
Method:	what()
Object:	
Argument:	
Range:	
Describe:	Returns information related to the platform that is the TI-Innovator™.
Result:	
Error:	
Type or Addressable Component:	

who()

Command:	who get(string) return string
Command Syntax:	**who** get(string) return string
Method:	who()
Object:	
Argument:	
Range:	
Describe:	Returns information related to the platform that is the TI-Innovator™.
Result:	
Error:	
Type or Addressable Component:	

last_error()

Command:	last_error()
Command Syntax:	**read last error** get(string) return string
Method:	last_error()
Object:	
Argument:	
Range:	
Describe:	Provides simple access to the **READ LAST ERROR** property of the TI-Innovator™. Useful or debugging programs.
Result:	
Error:	
Type or Addressable Component:	

sleep()

Command:	**sleep(seconds)**
Command Syntax:	**sleep(seconds)**
Method:	sleep(seconds)
Object:	time in seconds
Argument:	
Range:	
Describe:	Performs same operation as the **sleep()** function in the Python time module.
Result:	
Error:	
Type or Addressable Component:	

Floating Point Values and Digits after Decimal Point

In the ti_rover module, and TI-Innovator objects, various parameters are represented by floating point values. Numerous routines also return floating point values. These values are formatted in a specific manner for sending to the TI-Innovator™ Hub, and return values that represent sensor readings and other floating point return values are rounded to a specific number of digits after the decimal point for return.

The following information details these constraints.

Floating Point Return Values

Any module returning sensor readings, and other values that are represented by a floating point value will have their value rounded to 3 decimal positions to the right of the decimal point.

The following table shows the formatting for floating point values being sent for a variety of parameters to modules.

Digits to right of decimal point	Parameter Represents...	Python Format
1	Speed, Position (servos)	**:.1f**
1	Resistance (calibration)	**:.1f**
2	Seconds (time, duration)	**:.2f**
2	Frequency (blink, sound)	**:.2f**
2	Grid Units (TI-Rover)	**:.2f**
2	Position (TI-Rover)	**:.2f**
3	Distance (TI-Rover)	**:.3f**
3	Speed (TI-Rover)	**:.3f**
3	Angles (TI-Rover)	**:.3f**
3	Range (min, max)	**:.3f**
9	Calibration Coeffs	**:.9e**