

# nxtOSEK C API Reference

## [System architecture of LEGO MINDSTORMS NXT](#)

### [TOPPERS ATK API](#)

### [ECCRobot API](#)

[Servo Motor API](#)

[Light Sensor API](#)

[Touch Sensor API](#)

[Sound Sensor API](#)

[Ultrasonic Sensor API](#)

[I2C API](#)

[Bluetooth API](#)

[USB API](#) (Updated: January 2009)

[LCD display API](#)

[Sound API](#)

[NXT internal API](#)

[HiTechnic Acceleration Sensor API](#)

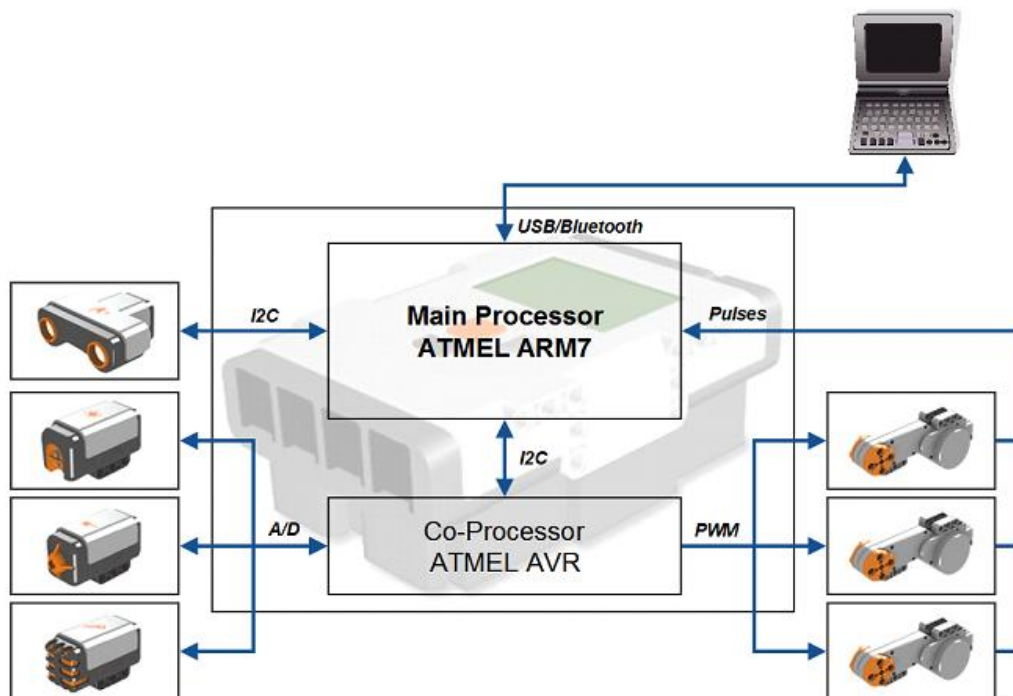
[HiTechnic Gyro Sensor API](#)

[RCX Sensors API](#)

### [nxtOSEK Hook routines](#)

## System architecture of LEGO MINDSTORMS NXT:

The below figure provides a graphical overview of LEGO MINDSTORMS NXT system architecture according to LEGO MINDSTORMS NXT Hardware Developer Kit.



This figure tells that communication between the main ARM7 processor (ATMEL AT91SAM7S256) and Sensors/Servo Motors is done via the co-processor (ATMEL AVR) except for the Ultrasonic Sensor and acquisition of Servo Motor revolutions. For nxtOSEK, most important factor to access Sensors/Servo Motors is the communication with the co-processor via I2C serial bus. This system architecture definitely influences on the software run-time environment of nxtOSEK. The main ARM7 processor accesses to Sensors (to read sensor A/D value) and Servo Motors (to set PWM duty ratio and brake mode) independently every 2 msec through a 1 msec periodical Interrupt Service Routine (ISR) of LEJOS NXJ platform. Servo Motors revolutions are directly captured by pulse triggered ISRs of LEJOS NXJ platform. Ultrasonic Sensor has its brain to directly communicate with the main ARM7 processor via another I2C communication channel.

## TOPPERS ATK API:

TOPPERS ATK is similar to the following version of OSEK OS/OIL according to TOPPERS project. OSEK API Reference and detailed information are described in the documents at the OSEK/VDX site:

[OSEK OS Version 2.2.1](#)

[OSEK OIL Version 2.5](#)

nxtOSEK restricts several TOPPERS ATK features due to the system architecture. User should not use ISR definitions and Interrupt handling API. For those who can understand Japanese, under `nxtOSEK\toppers_osek\doc` directory, there are detailed documents written in Japanese about TOPPERS ATK.

## ECRobot API:

ECRobot API consists of low level device API and ECRobot wrapper API (prefix of the API is `ecrobot_`) that is designed for real-time control application programming. Gray colored wrapper API in the below tables has duplicated feature of the original low level device API, so it is better to use the original API to reduce the run time overhead. When you use ECRobot API, you need to include `ecrobot_interface.h` in your source code.



Accessing to the Servo Motor is a bit complicated. NXT uses I2C communication between the ARM7 and the AVR to set Motor PWM value and brake mode. However, acquisition of the Motor count is done by pulse input capture of the ARM7 directly.

Servo Motor API	Description
<code>int nxt_motor_get_count(U32 n)</code>	gets Servo Motor revolution count in degree.  Parameters: <i>n</i> : NXT_PORT_A, NXT_PORT_B, NXT_PORT_C Returns: Servo Motors revolution in degree
<code>void nxt_motor_set_count(U32 n, int count)</code>	sets Servo Motor revolution count in degree.  Parameters: <i>n</i> : NXT_PORT_A, NXT_PORT_B, NXT_PORT_C <i>count</i> : Servo Motor revolution value Returns: none
<code>void nxt_motor_set_speed(U32 n, int speed_percent, int brake)</code>	sets Servo Motor PWM value and brake mode.  Parameters: <i>n</i> : NXT_PORT_A, NXT_PORT_B, NXT_PORT_C <i>speed_percent</i> : -100 to 100 <i>brake</i> : 0(float), 1(brake) Returns: none
<code>S32 ecrobot_get_motor_rev(U8 port_id)</code>	gets Servo Motor revolution value in degree. Wrapper of <code>nxt_motor_get_count</code> .  Parameters: <i>port_id</i> : NXT_PORT_A, NXT_PORT_B, NXT_PORT_C Returns: Servo Motors revolution in degree
<code>void ecrobot_set_motor_speed(U8 port_id, S8 speed)</code>	sets Servo Motor PWM value. Wrapper of <code>nxt_motor_set_speed</code> , but brake mode is fixed as brake.  Parameters: <i>port_id</i> : NXT_PORT_A, NXT_PORT_B, NXT_PORT_C <i>speed</i> : -100 to 100 Returns: none
<code>void ecrobot_set_motor_mode_speed(U8 port_id, S32 mode, S8 speed)</code>	sets Servo Motor brake mode and PWM value. Wrapper of <code>nxt_motor_set_speed</code>  Parameters: <i>port_id</i> : NXT_PORT_A, NXT_PORT_B, NXT_PORT_C <i>mode</i> : 0(float), 1(brake) <i>speed</i> : -100 to 100 Returns: none



NXT uses I2C communication between the ARM7 and the AVR to access the Light Sensor. The AVR uses 10bit A/D converter to get Light Sensor

data.

Light Sensor API	Description
void ecrobot_set_light_sensor_active(U8 <i>port_id</i> )	turns on infra-red light of Light Sensor. This function should be implemented in the device initialize hook routine.  Parameters: <i>port_id</i> : NXT_PORT_S1, NXT_PORT_S2, NXT_PORT_S3, NXT_PORT_S4 Returns: none
void ecrobot_set_light_sensor_inactive(U8 <i>port_id</i> )	turns off infra-red light of Light Sensor. This function should be implemented in the device terminate hook routine.  Parameters: <i>port_id</i> : NXT_PORT_S1, NXT_PORT_S2, NXT_PORT_S3, NXT_PORT_S4 Returns: none
U16 ecrobot_get_light_sensor(U8 <i>port_id</i> )	gets Light Sensor raw A/D data. Greater value means darker (or lower reflection).  Parameters: <i>port_id</i> : NXT_PORT_S1, NXT_PORT_S2, NXT_PORT_S3, NXT_PORT_S4 Returns: 0 to 1023



NXT uses I2C communication between the ARM7 and the AVR to access the Touch Sensor. The AVR uses 10bit A/D converter to get Touch Sensor data. In `ecrobot_get_touch_sensor` API, A/D data is converted to be 0/1.

Touch Sensor API	Description
U8 ecrobot_get_touch_sensor(U8 <i>port_id</i> )	gets Touch Sensor status.  Parameters: <i>port_id</i> : NXT_PORT_S1, NXT_PORT_S2, NXT_PORT_S3, NXT_PORT_S4 Returns: 0(not touched), 1(touched)



NXT uses I2C communication between the ARM7 and the AVR to access the Sound Sensor. The AVR uses 10bit A/D converter to get Sound Sensor data.

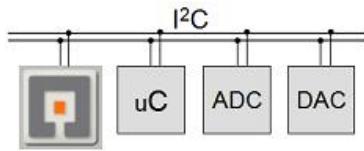
Sound Sensor API	Description
U16 ecrobot_get_sound_sensor(U8 <i>port_id</i> )	gets Sound Sensor raw A/D data. Smaller value means louder sound.  Parameters: <i>port_id</i> : NXT_PORT_S1, NXT_PORT_S2, NXT_PORT_S3, NXT_PORT_S4 Returns: 0 to 1023



Ultrasonic Sensor has its brain to communicate with the ARM7 via another I2C communication channel. `ecrobot_get_sonar_sensor` sends a protocol data to communicate with the Ultrasonic Sensor. However, actual data transaction between the ARM7 and the Ultrasonic Sensor is done by an ISR triggered by this function call, so there is one execution cycle delay to achieve consistent data acquisition.

Ultrasonic Sensor API	Description
void ecrobot_init_sonar_sensor(U8 <i>port_id</i> )	initializes a port for I2C communication for Ultrasonic Sensor. This function should be implemented in the device initialize hook routine.  Parameters: <i>port_id</i> : NXT_PORT_S1, NXT_PORT_S2, NXT_PORT_S3, NXT_PORT_S4 Returns: none
S32 ecrobot_get_sonar_sensor(U8 <i>port_id</i> )	gets Ultrasonic Sensor measurement data in cm via I2C.  Parameters:

	<code>port_id</code> : NXT_PORT_S1, NXT_PORT_S2, NXT_PORT_S3, NXT_PORT_S4 Returns: -1 to 255 (-1: not ready for measurement)
void ecrobot_term_sonar_sensor( <i>U8 port_id</i> )	terminates I2C communication used for Ultrasonic Sensor. This function should be implemented in the device terminate hook routine.  Parameters: <i>port_id</i> : NXT_PORT_S1, NXT_PORT_S2, NXT_PORT_S3, NXT_PORT_S4 Returns: none



I2C API	Description
void ecrobot_init_i2c( <i>U8 port_id</i> <i>U8 type</i> )	initializes a NXT sensor port for I2C communication This function should be implemented in the device initialize hook routine.  Parameters: <i>port_id</i> : NXT_PORT_S1, NXT_PORT_S2, NXT_PORT_S3, NXT_PORT_S4 <i>type</i> : LOWSPEED_9V/LOWSPEED Returns: none
SINT ecrobot_send_i2c( <i>U8 port_id</i> , <i>U32 address</i> , SINT <i>i2c_reg</i> , <i>U8 *buf</i> , <i>U32 len</i> )	send I2C data.  Parameters: <i>port_id</i> : NXT_PORT_S1, NXT_PORT_S2, NXT_PORT_S3, NXT_PORT_S4 <i>address</i> : 0x01 to 0x7F Note that addresses are from 0x01 to 0x7F not even numbers from 0x02 to 0xFE as given in some I2C device specifications. They are 7-bit addresses not 8-bit addresses. <i>i2c_reg</i> : I2C register e.g. 0x42 <i>buf</i> : buffer containing data to send <i>len</i> : length of the data to send Returns: 1(success)/0(failure)
SINT ecrobot_read_i2c( <i>U8 port_id</i> , <i>U32 address</i> , SINT <i>i2c_reg</i> , <i>U8 *buf</i> , <i>U32 len</i> )	read I2C data.  Parameters: <i>port_id</i> : NXT_PORT_S1, NXT_PORT_S2, NXT_PORT_S3, NXT_PORT_S4 <i>address</i> : 0x01 to 0x7F Note that addresses are from 0x01 to 0x7F not even numbers from 0x02 to 0xFE as given in some I2C device specifications. They are 7-bit addresses not 8-bit addresses. <i>i2c_reg</i> : I2C register e.g. 0x42 <i>buf</i> : buffer to return data <i>len</i> : length of the return data Returns: 1(success)/0(failure)
void ecrobot_term_i2c( <i>U8 port_id</i> )	terminates a NXT sensor port used for I2C communication. This function should be implemented in the device terminate hook routine.  Parameters: <i>port_id</i> : NXT_PORT_S1, NXT_PORT_S2, NXT_PORT_S3, NXT_PORT_S4 Returns: none



Inside of the NXT, there is a Bluecore chip to handle Bluetooth communication. The ARM7 uses UART to communicate with the Bluecore chip. In LEGO MINDSTORMS NXT Bluetooth Developer Kit, standard LEGO firmware supports a lot of communication protocols, however, ECRobot API does not support all LEGO standard communication protocol (i.e. no direct control, no diagnosis...).

- In case of NXT-PC Bluetooth communication, A Windows application program ([NXT GamePad](#)) is available to remotely control a NXT and acquire the NXT internal data (data logging).

- In case of NXT-NXT, only one NXT to one NXT communication is supported. Device configurations for a Bluetooth connection needs to be implemented in the code.

Bluetooth API	Description
void ecrobot_init_bt_master(const <i>U8 *bd_addr</i> , const <i>CHAR *pin</i> )	initializes NXT as the Bluetooth master device and establish a connection with a slave NXT. Bluetooth Device Address (BD_ADDR) consists of 48 bits, however, NXT requires additional 8 bits (total 7 bytes), so the last 8 bits should always be 0x00. This API needs to be implemented in a loop (e.g. <code>ecrobot_device_initialize</code> or a background Task).

	Parameters: <i>bd_addr</i> : Slave NXT's Bluetooth Device Address. If a slave NXT had 00:16:53:04:F1:B3 BD_ADDR, <i>bd_addr</i> should be defined such as: const U8 <i>bd_addr</i> [7] = {0x00, 0x16, 0x53, 0x04, 0xF1, 0xB3, 0x00};  <i>pin</i> : pin code string. Maximum character length is 16. (e.g. "LEJOS-OSEK") Returns: none
void ecrobot_init_bt_slave(const CHAR * <i>pin</i> )	initializes NXT as the Bluetooth slave device and establish a connection with a master device (PC, master NXT). This API needs to be implemented in a loop (e.g. <i>ecrobot_device_initialize</i> or a background Task).  Parameters: <i>pin</i> : pin code string. Maximum character length is 16. (e.g. "LEJOS-OSEK") Returns: none
void ecrobot_init_bt_connection(void)	initializes Bluetooth device and establish a connection with PC. PINCODE for the connection is pre-fixed as "MATLAB". This API needs to be implemented in a loop (e.g. <i>ecrobot_device_initialize</i> or a background Task).  Parameters: none Returns: none
SINT ecrobot_get_bt_status(void)	gets the status of Bluetooth device. To communicate with other device, Bluetooth device status has to be BT_STREAM.  Parameters: none Returns: Bluetooth device status BT_NO_INIT (device is not initialized) BT_INITIALIZED (device is initialized) BT_CONNECTED (device is connected through pairing process) BT_STREAM (device is ready for read/send user data)
U32 ecrobot_send_bt_packet(U8 * <i>buf</i> , U32 <i>bufLen</i> )	sends data packet to a Bluetooth device (PC, master/slave NXT).  Parameters: <i>buf</i> : Bluetooth send data packet <i>bufLen</i> : size of the data packet (max. 254) Returns: number of data to be sent to Bluecore chip
U32 ecrobot_read_bt_packet(U8 * <i>buf</i> , U32 <i>bufLen</i> )	reads data packet from a Bluetooth device (PC, master/slave NXT).  Parameters: <i>buf</i> : Bluetooth receive data packet <i>bufLen</i> : size of the data packet (max. 254) Returns: number of data to be received
void ecrobot_term_bt_connection(void)	terminates Bluetooth connection with a device (PC, master/slave NXT).  Parameters: none Returns: none
void ecrobot_bt_data_logger(S8 data1, S8 data2)	sends pre-fixed NXT internal status data. Data packet size is 32bytes and the following NXT internal status data is sent to PC via Bluetooth. This API is designed for NXT GamePad and NXT GamePad will save all data as a CSV file.  data packet    0-3bytes: system tick in msec, data type U32 data packet    4bytes: data1 (i.e. left analog stick input), data type S8 data packet    5bytes: data2 (i.e. right analog stick input), data type S8 data packet    6-7bytes: battery voltage in mV, data type U16 data packet    8-11bytes: Servo Motor revolution at Port A, data type S32 data packet 12-15bytes: Servo Motor revolution at Port B, data type S32 data packet 16-19bytes: Servo Motor revolution at Port C, data type S32 data packet 20-21bytes: Analog Sensor value at Port S1, data type S16 data packet 22-23bytes: Analog Sensor value at Port S2, data type S16 data packet 24-25bytes: Analog Sensor value at Port S3, data type S16 data packet 26-27bytes: Analog Sensor value at Port S4, data type S16 data packet 28-31bytes: I2C Sensor value, data type S32  Parameters: <i>data1</i> : user selected data (i.e. left analog stick input) <i>data2</i> : user selected data (i.e. right analog stick input) Returns: none





nxtOSEK uses LibUsb and LibNXT for USB host application program. Under samples directory, a USB target example (usbtst) and a USB host example (usbhost) are available. Currently, only one NXT can be communicated with a host via USB.

January, 2009: The new nxtOSEK v2.05 uses LEGO Fantom driver for USB host application program and USB communication protocol becomes more consistent(i.e. explicit connect/disconnect) and provides more functionalities(i.e. ~~multiple NXT connections with a host~~) compared to the previous version. For more detailed information about nxtOSEK USB API, see samples\usbtst directory. On the host side programming, see also samples\usbtst\usbhost directory.

USB API	Description
void ecrobot_init_usb(void)	initializes USB functionality in the NXT. This function must be invoked in ecrobot_device_initialize.  Parameters: none Returns: none
SINT ecrobot_set_name_usb(U8* name)	sets name to the NXT. By default, "nxt" is set as the device name in ecrobot_init_usb.  Parameters: name: device name of the NXT (character length is up to 16) Returns: 1(success)/0(failure)
U8 ecrobot_process1ms_usb(void)	USB process handler to establish a connection with a host. This function must be invoked every 1msec. (i.e. in a loop with 1msec wait, OSEK/JSP 1msec peiodical Task)  Parameters: none Returns: status of USB(USB_NO_INIT/USB_INIT/USB_CONNECTED)
SINT ecrobot_read_usb(U8* buf, U32 off, U32 len)	reads USB data from host after a USB connection was established.  Parameters: buf: data buffer off: offset in the buffer len: length of data to read (length is up to 64) Returns: length of read data
SINT ecrobot_send_usb(U8* buf, U32 off, U32 len)	sends USB data to host after a USB connection was established.  Parameters: buf: data buffer off: offset in the buffer len: length of data to send(length is up to 64) Returns: length of sent data
SINT ecrobot_disconnect_usb(void)	disconnects a current connection with host. This API enables to implement connect/disconnect sequence in application. This function makes USB device name being default, so it needs to invoke ecrobot_set_name_usb to change the device name to be user specified device name.  Parameters: none Returns: 1(success)/0(failure)
void ecrobot_term_usb(void)	terminates USB functionality in the NXT.  Parameters: none Returns: none



nxtOSEK supports for monochrome BMP files to draw graphics in the LCD display. User can design LCD graphics in BMP editor on PC and the BMP image can be displayed in the LCD on the NXT. GCC provides objcopy which enables to link BMP files to a nxtOSEK binary executable. Under samples directory, there are two BMP file examples (anime and bmpst).

LCD display API	Description
-----------------	-------------

SINT ecrobot_bmp2lcd(const CHAR * <i>file</i> , U8 * <i>lcd</i> , S32 <i>width</i> , S32 <i>height</i> )	converts a monochrome BMP file image to a BMP array data.  Parameters: <i>file</i> : BMP_DATA_START (BMP file name) macro has to be assigned. BMP file name should NOT contains file extension (.bmp) <i>lcd</i> : LCD array buffer (maximum data size is 800) <i>width</i> : LCD data width in bits (maximum width is 100) <i>height</i> : LCD data height in bits (maximum height is 64) Returns: 1: BMP file could be converted. 0: BMP file conversion was failed.
void display_bitmap_copy(const U8 * <i>data</i> , U32 <i>width</i> , U32 <i>depth</i> , U32 <i>x</i> , U32 <i>y</i> )	displays a BMP array data to LCD.  Parameters: <i>data</i> : Start address of a BMP array data <i>width</i> : BMP array data width in bits (maximum width is 100) <i>depth</i> : BMP array data height in bytes (maximum height is 8) <i>x</i> : BMP array data horizontal position in bits (0 - 100) <i>y</i> : BMP array data vertical position in bytes (0 - 8) Returns: none
void display_update(void)	updates LCD display.  Parameters: none Returns: none
void display_clear(U32 <i>updateToo</i> )	clears LCD display buffer.  Parameters: <i>updateToo</i> : 0 (not update LCD display), 1(update LCD display after clearing the buffer) Returns: none
void display_goto_xy(int <i>x</i> , int <i>y</i> )	specifies text display position. Top left is (0,0).  Parameters: <i>x</i> : horizontal position (0 to 15) <i>y</i> : vertical position (0 to 7) Returns: none
void display_string(const char * <i>str</i> )	stores string into LCD display buffer.  Parameters: <i>str</i> : supported ASCII characters are 0x00 to 0x7F Returns: none
void display_hex(U32 <i>val</i> , U32 <i>places</i> )	stores integer value into LCD display buffer to display hex expression.  Parameters: <i>val</i> : integer value to be displayed <i>places</i> : number of characters to be reserved Returns: none
void display_unsigned(U32 <i>val</i> , U32 <i>places</i> )	stores unsigned integer value into LCD display buffer.  Parameters: <i>val</i> : unsigned integer value to be displayed <i>places</i> : number of characters to be reserved Returns: none
void display_int(int <i>val</i> , U32 <i>places</i> )	stores signed integer value into LCD display buffer.  Parameters: <i>val</i> : signed integer value to be displayed <i>places</i> : number of characters to be reserved Returns: none
void ecrobot_status_monitor(const CHAR * <i>target_name</i> )	displays NXT internal status (application name, system tick, battery voltage, raw A/D data, motor revolutions, Bluetooth connection status, and Ultrasonic Sensor data) on the LCD. It is recommended to invoke this API in a low speed periodical Task (i.e. 500msec).  Parameters: <i>target_name</i> : target name string Returns: none



nxtOSEK supports for 8bit monoral PCM WAV file to play a sound. WAV file sound (e.g. music, voice...) can be directly played in the NXT. GCC provides objcopy which enables to link WAV files to a nxtOSEK binary executable. Under samples directory, there is a WAV file example (wavtest).

Sound API	Description
SINT ecrobot_sound_tone(U32 <i>freq</i> , U32 <i>ms</i> , U32 <i>vol</i> )	plays a tone, given its frequency, duration and volume. Frequency is audible from about 31 to 2100 Hertz. The duration argument is in hundreds of a seconds (centiseconds, not milliseconds) and is truncated at 256, so the maximum duration of a tone is 2.56 seconds.  Parameters: <i>freq</i> : frequency of a tone in Herz. <i>ms</i> : duration of a tone in hundreds of a second. <i>vol</i> : sound volume (0 - 100). 0 means mute. Returns: 1: sound tone is playing
SINT ecrobot_sound_wav(const CHAR * <i>file</i> , U32 <i>length</i> , U32 <i>freq</i> , U32 <i>vol</i> )	plays a 8bit monoral PCM WAV sound file.  Parameters: <i>file</i> : WAV_DATA_START (WAV file name) macro has to be assigned. WAV file name should NOT contains file extension (.wav) <i>length</i> : WAV_DATA_SIZE (WAV file name) macro has to be assigned. WAV file name should NOT contains file extension (.wav) <i>freq</i> : sampling frequency in Herz. (2000Hz - 22050Hz. -1: use WAV file original sampling frequency) <i>vol</i> : sound volume (0 - 100). 0 means mute. Returns: 1: WAV file is playing. 0: Sound resource is busy. -1: non-supported WAV file.
void sound_freq(U32 <i>freq</i> , U32 <i>ms</i> )	plays a tone, given its frequency and duration. Frequency is audible from about 31 to 2100 Hertz. The duration argument is in hundreds of a seconds (centiseconds, not milliseconds) and is truncated at 256, so the maximum duration of a tone is 2.56 seconds.  Parameters: <i>freq</i> : frequency of the tone in Herz. <i>ms</i> : duration of the tone in hundreds of a second. Returns: none



NXT internal API	Description
U8 ecrobot_is_ENTER_button_pressed(void)	returns the status of ENTER (ENTR) button.  Parameters: none Returns: Status of ENTER (ENTR) button 1: button is pressed 0: button is not pressed
U8 ecrobot_is_RUN_button_pressed(void)	returns the status of RUN button.  Parameters: none Returns: Status of RUN button 1: button is pressed 0: button is not pressed
U16 ecrobot_get_battery_voltage(void)	gets battery voltage data.  Parameters: none Returns: battery voltage in mV. (i.e. 9000 = 9.000V)
U32 systick_get_ms(void)	gets system tick in msec. system tick is started when the NXT is turned on (not started when an application begins)  Parameters: none Returns: system tick in msec



U32 ecrobot_get_systick_ms(void)	gets system tick in msec. Wrapper of systick_get_ms.  Parameters: none Returns: system tick in msec
void systick_wait_ms(U32 ms)	waits for specified msec.  Parameters: ms: wait time in msec Returns: none



NXT acquires three axes acceleration data from the HiTechnic Acceleration Sensor (NAC1040) via I2C communication. However, actual data transaction between the ARM7 and the Acceleration Sensor is done by an ISR triggered by a function call, so there is one execution cycle delay to achieve consistent data acquisition. According to HiTechnic, data refresh rate on the Sensor is approximately 100 times per second and acceleration is measured in the range of  $-2g$  to  $+2g$  with scaling of approximately 200 counts per g. For more detailed information about Acceleration Sensor, please visit [HiTechnic Web site](#).

HiTechnic Acceleration Sensor API	Description
void ecrobot_init_accel_sensor(U8 port_id)	initializes a port for I2C communication for Acceleration Sensor. This function should be implemented in the device initialize hook routine.  Parameters: port_id: NXT_PORT_S1, NXT_PORT_S2, NXT_PORT_S3, NXT_PORT_S4 Returns: none
void ecrobot_get_accel_sensor(U8 port_id, S16 *buf)	gets acceleration data in three axes.  Parameters: port_id: NXT_PORT_S1, NXT_PORT_S2, NXT_PORT_S3, NXT_PORT_S4 buf: acceleration data in three axes (buf[0]: x axis, buf[1]: y axis, buf[2]: z axis) Returns: none
void ecrobot_term_accel_sensor(U8 port_id)	terminates I2C communication used for Acceleration Sensor. This function should be implemented in the device terminate hook routine.  Parameters: port_id: NXT_PORT_S1, NXT_PORT_S2, NXT_PORT_S3, NXT_PORT_S4 Returns: none



NXT uses I2C communication between the ARM7 and the AVR to access the HiTechnic Gyro Sensor (NGY1044). The AVR uses 10bit A/D converter to get Gyro Sensor data. According HiTechnic, The Gyro Sensor can measure up to  $\pm 360^\circ$  per second of rotation and the rotation rate can be read up to approximately 300 times per second. For more detailed information about Gyro Sensor, please visit [HiTechnic Web site](#).

HiTechnic Gyro Sensor API	Description
U16 ecrobot_get_gyro_sensor(U8 port_id)	gets Gyro Sensor raw A/D data. The sensor data has offset value (approximately 600).  Parameters: port_id: NXT_PORT_S1, NXT_PORT_S2, NXT_PORT_S3, NXT_PORT_S4 Returns: none



RCX Active Sensor (e.g. Light Sensor) on NXT requires power source to drive the sensor. RCX Active Sensor API may be used for other RCX Active Sensors (not tested all).

RCX Active Sensor API	Description
void	supplies power source to RCX Active Sensor. This function should be implemented in the

ecrobot_set_RCX_power_source(U8 <i>port_id</i> )	device initialize hook routine.  Parameters: <i>port_id</i> : NXT_PORT_S1, NXT_PORT_S2, NXT_PORT_S3, NXT_PORT_S4 Returns: none
S16 ecrobot_get_RCX_sensor(U8 <i>port_id</i> )	gets RCX Active Sensor raw data.  Parameters: <i>port_id</i> : NXT_PORT_S1, NXT_PORT_S2, NXT_PORT_S3, NXT_PORT_S4 Returns: sensor raw data
void ecrobot_term_RCX_power_source(U8 <i>port_id</i> )	stops power source to RCX Active Sensor. This function should be implemented in the device terminate hook routine.  Parameters: <i>port_id</i> : NXT_PORT_S1, NXT_PORT_S2, NXT_PORT_S3, NXT_PORT_S4 Returns: none



RCX Touch Sensor is compatible with NXT A/D Sensors such as NXT Touch Sensor. This API is implement to just distinguish those RCX/NXT Touch Sensors. (ecrobot\_get\_touch\_sensor also could be used to access RCX Touch Sensor)

RCX Touch Sensor API	Description
U8 ecrobot_get_RCX_touch_sensor(U8 <i>port_id</i> )	gets Touch Sensor status.  Parameters: <i>port_id</i> : NXT_PORT_S1, NXT_PORT_S2, NXT_PORT_S3, NXT_PORT_S4 Returns: 0(not touched), 1(touched)

### nxtOSEK Hook routines:

nxtOSEK provides 3 hook routines for user. These hook routines have to be defined in the user code. ecrobot\_main.c file of NXTway sample is useful to understand how to use these hook routines.

- void ecrobot\_device\_initialize(void) : This hook routine is invoked when the NXT was turned on. User can implement device initialize functions in this function.

```
void ecrobot_device_initialize(void)
{
    /* Initialize ECRobot used devices */
    ecrobot_set_light_sensor_active(NXT_PORT_S1);
    ecrobot_set_light_sensor_active(NXT_PORT_S3);
    ecrobot_init_sonar_sensor(NXT_PORT_S2);
    ecrobot_init_bt_connection();
}
```

- void ecrobot\_device\_terminate(void) : This hook routine is invoked when STOP or EXIT button was pressed. User can implement device terminate functions in this function.

```
void ecrobot_device_terminate(void)
{
    /* Terminate ECRobot used devices */
    ecrobot_set_light_sensor_inactive(NXT_PORT_S1);
    ecrobot_set_light_sensor_inactive(NXT_PORT_S3);
    ecrobot_set_motor_speed(NXT_PORT_B, 0);
    ecrobot_set_motor_speed(NXT_PORT_C, 0);
    ecrobot_term_sonar_sensor(NXT_PORT_S2);
    ecrobot_term_bt_connection();
}
```

- void user\_lms\_isr\_type2(void) : This hook routine is invoked from a 1msec periodical ISR in category 2. For example, user can implement an OSEK Alarm counter for Rate Monotonic Scheduling in this hook routine.

```
#include "kernel.h"
#include "kernel_id.h"

void user_lms_isr_type2(void)
{
    StatusType ercd;

    /* Increment System Timer Count */
    ercd = SignalCounter(SysTimerCnt);
    if (ercd != E_OK)
    {
        ShutdownOS(ercd);
    }
}
```

[Home](#)