LabVIEW Toolkit for LEGO® MINDSTORMS® NXT Programming Guide

POWERED BY NATIONAL INSTRUMENTS LabVIEW™

Table of Contents

Introduction

LEGO® MINDSTORMS® NXT Software 1.0 uses a graphical programming language called NXT-G, and this programming language is based on National Instruments LabVIEW 7.1. The high-level programming blocks in NXT-G block diagrams are built out of lower-level LabVIEW block diagrams. Ultimately, NXT-G blocks are just a special kind of VI developed for use with MINDSTORMS.

Using the LabVIEW Toolkit for LEGO MINDSTORMS NXT with LabVIEW 7.1 or 8.2 allows you to bypass the high-level limitations imposed by NXT-G. The toolkit enables you to create advanced programs using VIs and download them to the NXT Intelligent Brick. You can even extend the NXT-G programming language with new high-level blocks built out of LabVIEW VIs. Furthermore, LabVIEW provides a more advanced editing environment than LEGO MINDSTORMS NXT Software 1.0.

This document details the NXT-specific modifications and extensions to LabVIEW provided by the toolkit. Note that the toolkit does not support all LabVIEW features. Therefore, make sure you are aware of the caveats listed in this document when programming with the toolkit.

Refer to the National Instruments Web site at `ni.com/mindstorms` for more information about the LabVIEW Toolkit for LEGO MINDSTORMS NXT.

Assumptions

This document assumes that you are using the LabVIEW Toolkit for LEGO MINDSTORMS NXT with LabVIEW 7.1 or 8.2. This document also assumes that you are familiar with the following LabVIEW programming concepts:

- Data types: integers, Booleans, strings, arrays, and clusters.
- Basic VIs and functions used with the data types above.
- Case, loop, and sequence structures
- Usage of subVIs
- Properties and methods

Refer to the *LabVIEW Help*, available in LabVIEW by selecting **Help»Search the LabVIEW Help**, for more information about these concepts.

The toolkit provides special palettes for working with both front panels and block diagrams. These palettes contain only supported LabVIEW features. You are not required to use these palettes, but using them will help you remember which features the toolkit supports.

This document assumes that you are using NXT firmware 1.03 or a compatible official update. Version 1.03 is the most current version as of the publication of this document. Support for various data types and/or operations might change in future firmware versions.

NXT Firmware Concepts

The following sections provide a brief summary of how LabVIEW concepts correspond to NXT firmware concepts. Refer to the LEGO MINDSTORMS Web site at `mindstorms.lego.com` for the NXT SDK/HDK documents describing the NXT hardware and firmware platform.

NXT Programs and LabVIEW VIs

When you choose a VI to compile and download to an NXT brick using the toolkit, that VI becomes the *top-level VI* of the corresponding program file on the NXT brick. The toolkit compiles the top-level VI and all of its subVIs into one program file, then downloads the program file to the NXT brick using the same name as the top-level VI. You can think of the top-level VI and its subVIs as the complete source code for resulting binary program file.

Note the implication that subVIs have no direct analogue on the file system of the NXT brick. Inside the program file, subVIs become shared subroutines. Refer to the *SubVIs* section of this document for more information about using subVIs.

NXT Bytecode Instructions and LabVIEW Functions

The NXT firmware executes a program file by interpreting its contents as bytecode instructions. You can think of many of the bytecode instructions as loosely analogous to LabVIEW functions. For example, the NXT firmware has an ADD bytecode that shares its semantics with the Add function in LabVIEW. In other words, the ADD bytecode behaves similarly to the Add function for a given set of input data types and values.

The *LabVIEW Help* documentation describes how all of the supported functions operate in general. Refer to the *Functions* section of this document for a list of NXT-specific exceptions and behavior changes.

NXT Program Data Space

NXT programs maintain a data space for run-time data similarly to how LabVIEW maintains VI data spaces. You do not need to manage the data space because the toolkit compiler automatically organizes the data space of NXT programs and the NXT firmware manages the data space at run time.

This automatic management of the data space includes resizing arrays and strings as needed, much like LabVIEW. However, using arrays and strings requires a relatively large amount of memory and execution time on the NXT brick, so avoid overusing arrays and strings in the programs you write. If you extensively use dynamic array and string resizing while the program runs, such as placing Build Array functions in a loop, the program might intermittently run more slowly than normal.

Furthermore, extensive use of dynamic array and string resizing can cause the NXT firmware to run out of memory during program execution. If the total size of the data space, including the array and string data, exceeds the size of the allotted RAM pool in the NXT firmware, the program immediately aborts and prints an error message on the display of the NXT brick.

Parallel Execution on the NXT Brick

NXT firmware 1.03 supports parallel execution of VI code in a similar sense to the way that LabVIEW executes parallel code on single-processor PC. Parallel streams of functions or subVI calls share the CPU using an automatically managed cooperative multitasking system.

You do not need to perform any special steps to enable parallel execution. Write parallel code the same way you write parallel code for VIs that run on a PC.

LabVIEW Basics

Data Types

The toolkit only supports a limited subset of the many data types that LabVIEW supports. The toolkit supports the following data types:

- ✓ 8-bit signed and unsigned integer numerics
- ✓ 16-bit signed and unsigned integer numerics
- ✓ 32-bit signed and unsigned integer numerics
- ✓ Booleans
- ✓ Strings
- ✓ Arrays containing integers, Booleans, strings, or clusters
    - Only one-dimensional (1D) arrays are supported
- ✓ Clusters containing integers, Booleans, strings, 1D arrays, or other clusters

The toolkit does not support any data types not listed above. The following list includes some examples of unsupported data types:

- 🛇 Floating-point numerics (including complex)
- 🛇 Enumerations
- 🛇 Arrays of two or more dimensions (including tables)
- 🛇 Reference numbers (refnums) of any type other than those the toolkit provides
- 🛇 Paths
- 🛇 Waveform, digital waveform, or digital data table
- 🛇 Variant or ActiveX data

Controls, Indicators, and Constants

When you compile and download VIs to an NXT brick, the layout and appearance of their front panels do not directly influence the compiled code. Thus, you can use any front panel control or indicator style as long as the NXT firmware supports the data type. For example, you can use numeric ring, slide, knob, chart, and graph controls and indicators as long as they do not rely on floating-point numbers or multi-dimensional arrays.

The same principle is true for custom controls, type definitions, and block diagram constants. As long as the NXT firmware supports the data type of the wire connected to a particular block diagram terminal, you can use any style or type definition.

Sequence Flow Boolean Controls

The toolkit provides a special kind of Boolean control, Sequence Flow. When using Sequence Flow controls, be aware of the following special caveats:

1. The toolkit identifies Sequence Flow controls by name. If the name (label) of a Boolean control or indicator begins with "Sequence Flow", the toolkit compiler classifies that control or indicator as a Sequence Flow control.
2. The toolkit classifies wires connected to Sequence Flow controls as Sequence Flow wires. These wires are directly analogous to the white LEGO® TECHNIC® beams on NXT-G diagrams.
3. No Sequence Flow controls, indicators, or wires generate corresponding data space items in the compiled NXT program.

NXT Refnum Controls

The toolkit provides some special classes of generic refnum controls in order to expose NXT-specific features.

Refer to the *Extensions to LabVIEW* section of this document for specific information on how to use each new class of refnum. When using NXT refnums, be aware of the following special caveats:

1. The toolkit identifies NXT refnum controls by name. If the name (label) of a generic refnum control or indicator begins with "Generic Refnum Name", the toolkit compiler classifies that control or indicator as a NXT refnum.
2. NXT refnum controls, indicators, or wires do not generate corresponding data space items in the compiled NXT program. NXT refnum controls enable you to access new properties and methods that the toolkit provides.
3. Only wire these controls directly to the reference inputs of Property and Invoke Nodes. Do not assign NXT refnum controls to subVI connector pane terminals.

Functions

The toolkit does not support all LabVIEW functions, and some supported functions exhibit special behavior or limitations when you use them with the toolkit. In general, any function that relies on one or more unsupported data types is itself unsupported.

The following lists the functions that the toolkit supports, along with any special usage notes.

Numeric

- ✓ Add
- ✓ Subtract
- ✓ Multiply
- ✓ Quotient & Remainder (integer division)
- ✓ Increment
- ✓ Decrement
- ✓ Negate

Conversion

- ✓ Numeric Conversions
  - o Only conversions to integer data types are supported.
- ✓ Boolean to (0,1)
- ✓ String to Byte Array
- ✓ Byte Array to String
  - o Only true byte array inputs are supported (LabVIEW automatically coerces any integer array to bytes, but the toolkit does not support this feature)

Logic

- ✓ And
- ✓ Or
- ✓ Exclusive Or
- ✓ Not
  - o The Not function only supports Boolean inputs.

Strings

- ✓ String Length
- ✓ Concatenate Strings
  - o String array inputs are not supported.
- ✓ String Subset
- ✓ Number to Decimal String
  - o The width input is ignored.
- ✓ Decimal String to Number

Arrays

The toolkit supports several basic array functions, but you cannot resize most of them to accept arbitrary numbers of inputs. Remember that you can only use 1D arrays.

- ✓ Array Size
- ✓ Index Array
  - o You cannot resize this function.
- ✓ Array Subset
  - o You cannot resize this function.
- ✓ Replace Array Subset
  - o You cannot resize this function.
- ✓ Initialize Array
  - o You cannot resize this function.
- ✓ Build Array
  - o You must enable the **Concatenate Inputs** option with array inputs.

Comparison

The comparison functions listed below support all compatible data types and the Compare Aggregates mode (where applicable).

- ✓ Equal?
- ✓ Not Equal?
- ✓ Equal to 0?
- ✓ Not Equal to 0?
- ✓ Greater?
- ✓ Less?
- ✓ Greater or Equal?
- ✓ Less or Equal?
- ✓ Greater than 0?
- ✓ Less than 0?
- ✓ Greater or Equal to 0?
- ✓ Less or Equal to 0?

Miscellaneous

- ✓ Bundle
  - The toolkit does not support Bundle by Name.
- ✓ Unbundle
  - The toolkit does not support Unbundle by Name.
- ✓ Select
- ✓ Tick Count
- ✓ Flatten to String
  - The toolkit does not support the output type string value.
- ✓ Unflatten from String
  - You must wire a fully constructed template for value into the type input. The size of data in type input must match the flattened data in binary string exactly, including array sizes.
- ✓ Stop

Structures

The toolkit supports the following structures, with some caveats:

- ✓ While Loops
  - Stacked shift registers are not supported.
  - Feedback nodes within while loops are not supported.
  - Auto-indexing on array tunnels is not supported.
- ✓ Case Structures
  - Only one specific value (or "Default") is allowed in the case selector label for each subdiagram. You cannot use ranges (".." notation) or comma-separated lists of values.
  - Usage of error cluster wires with the selector terminal is not supported.
- ✓ Sequence
  - Only one frame per sequence structure is allowed.

SubVIs

The NXT firmware run-time system is simpler than the LabVIEW run-time system, you must keep in mind the following few limitations when using subVIs:

- You can pass only supported data types through subVI connector panes.
- The toolkit does not support VI refnums, so you cannot call subVIs by reference.
- Most settings on the **VI Properties** page do not affect the execution of VIs on the NXT brick. For example, the NXT firmware does not support the concepts of "priority" or a "preferred execution system."

The **Reentrant execution** option in the **VI Properties** dialog box is a special exception. Reentrant execution enables a VI to be called by more than one caller. Normally, a VI can only be called by one caller at a time. However, if you want two callers to be able to call the same VI simultaneously, place a checkmark in the **Reentrant execution** checkbox. The toolkit makes a complete copy of reentrant VIs for each subVI call. In other words, every call to a reentrant subVI is completely independent in the NXT program and increases the overall size of the compiled program.

Remember that only one caller can have access to a non-reentrant subVI at any given time. The same issue exists here as with using non-reentrant subVIs in LabVIEW: parallel callers might have to wait (block) for access while another caller is running a given subVI.

Also remember that the toolkit compiler combines top-level VIs and all subVIs into a single program file on the NXT brick. This behavior is similar to using the LabVIEW Application Builder to compile all VIs into a single PC executable file – subVIs just become components inside the executable file.

Extensions to LabVIEW

The NXT firmware supports various I/O devices for which LabVIEW has no built-in interfaces. Examples include motor outputs, sensor inputs, and the NXT display. To support these devices, the toolkit provides extensions to LabVIEW in the form of four new classes of *generic reference numbers* (*refnums*) and an associated set of properties and/or methods with each class of refnums. Each property/method loosely corresponds to a particular piece of the I/O interface provided by the NXT bytecode interpreter.

The toolkit provides the following four new classes of generic refnums:

- NXTInput
- NXTOutput
- NXTOutputMulti
- NXTSyscall

The toolkit palettes provide type definition controls of each new refnum class.

NXTInput, NXTOutput, and NXTOutputMulti refnums are all used with Property Nodes to expose the interfaces to input and output devices. Connect any of these refnums to a Property Node, and then resize the Property Node to expose as many properties as you need to read or write.

Each Property Node must include an input to specify the physical port(s) to access. Refer to the specific Port or PortList property descriptions for more information each port. Each additional Property Node line corresponds to some configuration field (i.e. device mode, measured value, etc.) for the specified port(s).

NXTSyscall refnums are used with Invoke Nodes to expose system-level methods for access to various other features of the NXT firmware. Each method you use corresponds to a call to an internal system function matching the name of the method, or the *system call*. Most system call methods provide status codes as their return values. In general, status code values indicate one of the following three results:

- A status code of zero indicates a status of “OK”; no special action is required.
- A negative status code indicates an error.
- A positive status code indicates a warning.

System call methods might have more parameters listed below the return value. These parameters can be inputs, outputs, or both.

The toolkit does not support the usage of error cluster inputs or outputs on Property Nodes or Invoke Nodes.

The following sections describe how to program various device types in more detail.

Analog Input Devices (Touch, Light, and Sound Sensors)

The NXT bricks feature four input (sensor) ports, which you can connect interchangeably to both analog inputs (such as light sensors) and digital inputs (such as the NXT ultrasonic sensor). This section describes the interface provided to program analog input devices.

Input Port Properties

To configure or read values from analog I/O devices, use the NXTInput interface. To do so, connect an NXTInput refnum wire to the reference input of a Property Node, then resize the Property Node to add more terminals. Adding more terminals enables you to display the properties you want to access. This section describes each property that this toolkit supports.

Note that special rules apply to the Port property:

- Every NXTInput Property Node must include one and only one Port property.
- For a given Property Node, the value on the wire connected to Port specifies the physical input port to which all other properties apply.

Given the special nature of Port, National Instruments recommends always specifying the Port property as the first property in each NXTInput Property Node. This convention enhances the clarity of the code.

With the exception of the Port property, each NXTInput property corresponds to a configuration field in the input module of the NXT firmware. The toolkit compiler guarantees that all write operations execute before all read operations, and all operations execute in the order they appear in the Property Node (from top to bottom).

Port

U8

write-only

required

Legal value range: [0, 3]

This property specifies the physical input port to which all other property read/writes apply. Note that legal values are zero-based, such that the values 0, 1, 2, and 3 correspond to the physical port labels “1”, “2”, “3”, and “4”, respectively.

Every NXTInput Property Node must include only one wired Port property.

SensorType

U8

read-write

optional

Legal values:

0x00	*NO_SENSOR*	*No sensor configured*
0x01	*SWITCH*	*NXT or RCX touch sensor*
0x02	*TEMPERATURE*	*RCX temperature sensor*
0x03	*REFLECTION*	*RCX light sensor*
0x04	*ANGLE*	*RCX rotation sensor*
0x05	*LIGHT_ACTIVE*	*NXT light sensor with floodlight enabled*
0x06	*LIGHT_INACTIVE*	*NXT light sensor with floodlight disabled*
0x07	*SOUND_DB*	*NXT sound sensor; dB scaling*
0x08	*SOUND_DBA*	*NXT sound sensor; dBA scaling*
0x09	*CUSTOM*	*Unused*
0x0A	*LOWSPEED*	*I2C digital sensor*
0x0B	*LOWSPEED_9V*	*I2C digital sensor; 9V power*
0x0C	*HIGHSPEED*	*Unused*

This property specifies the sensor type for this port. The sensor type primarily affects scaling factors used to calculate the normalized sensor value, but some values have other side effects.

If you write to this property, also write a value of TRUE to the InvalidData property.

Unlike the RCX firmware, no default sensor modes are associated with each sensor type.

SensorMode

U8

read-write

optional

Legal values:

0x00	*RAWMODE*	*Report scaled value equal to raw value.*
0x20	*BOOLEANMODE*	*Report scaled value as one (TRUE) or zero (FALSE). Readings are FALSE if raw value exceeds 55% of total range; readings are TRUE if raw value is less than 45% of total range (inverse Boolean logic).*
0x40	*TRANSITIONCNTMODE*	*Report scaled value as number of transitions between TRUE and FALSE.*
0x60	*PERIODCOUNTERMODE*	*Report scaled value as number of transitions from FALSE to TRUE, then back to FALSE.*
0x80	*PCTFULLSCALEMODE*	*Report scaled value as percentage of full scale reading for configured sensor type.*
0xA0	*CELSIUSMODE*	*Scale TEMPERATURE reading to degrees Celsius.*
0xC0	*FAHRENHEITMODE*	*Scale TEMPERATURE reading to degrees Fahrenheit.*
0xE0	*ANGLESTEPMODE*	*Report scaled value as count of ticks on RCX-style rotation sensor.*

This property specifies the sensor mode for this port. The sensor mode affects the scaled value, which the NXT firmware calculates depending on the sensor type and sensor mode.

If you write to this property, also write a value of TRUE to the InvalidData property.

Raw

U16

read-only

optional

Legal value range: [0, 1023]

This property specifies the raw 10-bit value last read from the analog to digital converter on this port. Raw values produced by sensors typically cover some subset of the full 10-bit range.

Normalized

U16

read-only

optional

Legal value range: [0, 1023]

This property specifies a 10-bit sensor reading, scaled according to the current value of the SensorType property on this port. The NXT firmware automatically applies internal scaling factors such that the physical range of raw values produced by the sensor is mapped (normalized) to the full 10-bit range.

SensorMode is ignored when calculating the Normalized value.

The InvalidData property should be read in conjunction with this property. The Normalized value is only valid if InvalidData is FALSE.

Scaled

U16

read-write

optional

Legal value range depends on SensorMode, as listed below:

RAWMODE	*[0, 1023]*
BOOLEANMODE	*[0, 1]*
TRANSITIONCNTMODE	*[0, 65535]*
PERIODCOUNTERMODE	*[0, 65535]*
PCTFULLSCALEMODE	*[0, 100]*
CELSIUSMODE	*[-200, 700] (readings in 10th of a degree Celsius)*
FAHRENHEITMODE	*[-400, 1580] (readings in 10th of a degree Fahrenheit)*
ANGLESTEPMODE	*[0, 65535]*

This property specifies a sensor reading, scaled according to the current sensor type and mode on this port.

The InvalidData property should be read in conjunction with this property. The Scaled value is only valid if InvalidData is FALSE.

Because some combinations of sensor types and modes might lead to accumulation of count values in the Scaled value field, you can reset this count by writing zero to the Scaled property at any time. Note that you can write any value to this property at any time, but doing so is not generally very useful because outside of counter modes, the value is usually overwritten very quickly.

InvalidData

Boolean

read-write

optional

Legal values: TRUE, FALSE

This property signifies that the values of Raw, Normalized, and Scaled might be invalid due to sensor configuration changes that the NXT firmware has not processed yet. For example, the NXT firmware might not have processed the sensor type and/or mode changes immediately due to hardware limitations.

In all cases where you change SensorType or SensorMode, use the InvalidData property to ensure that the next value you read back has been properly processed. To do so, set InvalidData to TRUE in the same Property Node as you use to set SensorType and/or SensorMode, and then write a while loop that does not exit until InvalidData becomes FALSE. At that point, the Normalized and Scaled properties will return valid values for the new configuration.

Digital I/O Devices (Ultrasonic Sensors)

To program digital I/O devices, you need to use both the NXTInput and NXTSyscall interfaces. You need both interfaces because digital and analog devices share the same physical ports.

Digital I/O Port Setup

First, use an NXTInput Property Node, as described above, to set the SensorType property of the port to a value compatible with the sensor. The NXT firmware supports the following two SensorType values for use with digital devices:

- ✓ LOWSPEED—Use this type if the sensor does not require 9V power.
- ✓ LOWSPEED_9V—Use this type if the sensor requires 9V power.

These low speed sensor types are for use with sensors using the I2C protocol to communicate with the NXT brick. The ultrasonic sensor included with NXT bricks requires 9V power, so use LOWSPEED_9V when this sensor is connected.

Remember that you also need to set the InvalidData property to TRUE after setting a new SensorType, then wait (via a While Loop) for the NXT firmware to set it back to FALSE. This process ensures that the firmware has time to initialize the port properly, including the 9V power lines, if applicable. Some digital devices might need additional time to initialize after power up.

Low Speed Communication Methods

After the port is initialized, you need to access the I2C communication subsystem of the NXT firmware to communicate with the device.

Use NXTSyscall refnums with invoke nodes to expose the interface to this communication system. The toolkit provides the following three methods for low speed digital communication: NXTCommLSWrite, NXTCommLSCheckStatus, and NXTCommLSRead.

To understand how to use these methods for communication, you need to understand that the firmware uses a *master/slave* setup in which the NXT brick is always the master device. That is, the NXT firmware is responsible for controlling the write and read operations. Furthermore, the firmware maintains write and read buffers for each port, and the three system call methods provided enable you to access these buffers.

A call to NXTCommLSWrite constitutes the start of an asynchronous *transaction* between the NXT brick and a digital device, such that the program continues to run while the firmware manages sending bytes from the write buffer and reading the response bytes from the device. Because the NXT is the master device, you must also specify the number of bytes to expect from the device in response to each write operation. You can exchange up to 16 bytes in each direction per transaction.

After you start a write transaction with NXTCommLSWrite, use NXTCommLSCheckStatus in a While Loop to check the status of the port. If NXTCommLSCheckStatus returns a status code of zero and a count of bytes available in the read buffer, the system is ready for you to use NXTCommLSRead to copy the data from the read buffer into your own buffer.

Note that any of these calls might return various status codes at any time. A status code of zero means the port is idle and the last transaction (if any) did not result in any errors. Negative status codes indicate errors, while the positive status code STAT_COMM_PENDING indicates that a transaction is in progress on the specified port.

The following sections provide more information about each low speed communication method.

NXTCommLSCheckStatus

Return Value:

Status Code	*I8*	

Parameters:

Port	*U8*	*input*	*Port, [0, 3]*
BytesReady	*U8*	*input*	*Number of bytes ready for reading, if any*

This system call method checks the status of the I2C communication on the specified port. If the last operation on this port was a successful NXTCommLSWrite operation that requested response data from a device, BytesReady indicates the number of bytes in the internal read buffer. You can access this information using NXTCommLSRead.

If the return value is zero, the port is idle and the last operation (if any) did not cause any errors.

This method can return the following non-zero status codes:

32	*0x20*	*STAT_COMM_PENDING*	*Port is busy performing a transaction.*
-35	*0xDD*	*ERR_COMM_BUS_ERR*	*Last transaction failed (possible device failure)*
-33	*0xDF*	*ERR_COMM_CHAN_INVALID*	*Specified port is out of range*
-32	*0xE0*	*ERR_COMM_CHAN_NOT_READY*	*Specified port is not properly configured*

Commonly, ERR_COMM_BUS_ERR means that either no digital device is connected to the specified port or the connected device is configured incorrectly. Attempt to write new data to the device to clear the error condition.

If you see ERR_COMM_CHAN_NOT_READY, make sure SensorType is set properly and InvalidData is FALSE for this port before attempting further transactions.

If you see STAT_COMM_PENDING, an operation is in progress and you should not interrupt it. Avoid calls to NXTCommLSRead or NXTCommLSWrite until NXTCommLSCheckStatus returns zero or a negative error code.

NXTCommLSWrite

Return Value:

Status Code	*I8*	

Parameters:

Port	*U8*	*input*	*Port, [0-3]*
Buffer	*U8 array*	*input*	*Up to 16 bytes for writing to device*
ReturnLength	*U8*	*input*	*Number of bytes expected from device in response to writing data in Buffer; maximum 16*

This system call method copies data from the buffer input to an internal write buffer and instructs the NXT firmware to perform a transaction by sending the write buffer to the device and reading ReturnLength bytes back into the internal read buffer.

If the return value is zero, the method successfully started a communication transaction. Use NXTCommLSCheckStatus to monitor the status of the transaction.

This method can return the following non-zero status codes:

-33	*0xDF*	*ERR_COMM_CHAN_INVALID*	*Specified port is out of range*
-32	*0xE0*	*ERR_COMM_CHAN_NOT_READY*	*Specified port is busy or not properly configured*
-19	*0xED*	*ERR_INVALID_SIZE*	*Either buffer size or ReturnLength exceeded 16-byte maximum*

NXTCommLSRead

Return Value:

Status Code	*I8*	

Parameters:

Port	*U8*	*Input*	*Port, [0-3]*
Buffer	*U8 array*	*Out*	*Bytes read from device, if any*
BufferLength	*U8*	*Input*	*Upper bound on number of bytes to read into Buffer; actual number returned limited to bytes available in the internal read buffer*

This system call method attempts to copy BufferLength bytes from the internal read buffer to your buffer.

If the return value is zero, the read operation succeeded and Buffer contains all bytes available in the internal buffer. Successive NXTCommLSRead method calls will read new data each time.

This method can return the following non-zero status codes:

32	*0x20*	*STAT_COMM_PENDING*	*Port is busy performing a transaction*
-35	*0xDD*	*ERR_COMM_BUS_ERR*	*Last transaction failed (possible device failure)*
-33	*0xDF*	*ERR_COMM_CHAN_INVALID*	*Specified port is out of range*
-32	*0xE0*	*ERR_COMM_CHAN_NOT_READY*	*Specified port is not configured properly*

If any negative status code (error) is returned, Buffer is an empty array.

Output Devices (Motors and Lamps)

Two different kinds of output devices are compatible with the NXT brick: interactive motors (NXT motors with integrated position encoders) and non-interactive motors or lamps (RCX-compatible outputs). All of these devices are programmed using a common set of Property Nodes in LabVIEW. The only difference is that non-interactive devices ignore the properties that do not affect them. For example, instructing a lamp to rotate counter-clockwise for 360 degrees does not make sense, the lamp only response to power levels. Furthermore, VIs written to use interactive device features might not operate correctly when used with non-interactive devices. For example, you might write a VI to instruct an interactive motor to rotate 360 degrees, and then wait in a While Loop for the NXT firmware to complete the action. Non-interactive devices, such as lamps, do not include position encoders, so such a loop never will receive the necessary feedback to finish executing.

You can access output ports using one of the following two classes of Property Nodes: NXTOutput and NXTOutputMulti. NXTOutput only allows accessing parameters of one port at a time, while NXTOutputMulti accepts an array of port numbers so you can configure multiple ports in a single operation. This ability of NXTOutputMulti is useful to ensure that the firmware is able to synchronize multiple motors tightly. Conversely, you can only read properties from one output at a time, so NXTOutput is useful for inspecting the state (i.e. power level or motor position) of an output port.

Use NXTOutputMulti to synchronize the behavior of two or more motors, and use NXTOutput to read properties of an output or to only configure one output at a time.

Output Port Properties

To configure or read values from output devices, connect an NXTOutput or NXTOutputMulti refnum wire to any reference input of the Property Node, and then resize the Property Node to add more terminals. Adding more terminals enables you to display the properties you want to access.

This section describes each property that this toolkit supports. Because NXTOutput and NXTOutputMulti Property Nodes share many properties, this section describes the properties only once. Each property description specifies the access rules (read-only, write-only, or read-write) provided by each interface.

Note that special rules apply to the Port and PortList properties:

- Every NXTOutput or NXTOutputMulti Property Node must include one and only one Port of the PortList property, respectively.
- For a given Property Node, the value(s) on the wire connected to Port/PortList specify the physical output port to which all other properties apply.

Given the special nature of Port/PortList, National Instruments recommends always specifying Port/PortList as the top property in each NXTOutput/NXTOutputMulti node. This coding convention enhances code clarity.

With the exception of Port/PortList, each property corresponds to a write or read of the associated field in the output module of the NXT firmware. All write operations are guaranteed to execute before all read operations, and all operations execute in the order they occur on the Property Node (from top to bottom). Furthermore, the toolkit compiler combines all write operations in each Property Node into a single bytecode instruction, such that all write operations in a Property Node occur simultaneously.

Some property write operations also require an additional step to “commit” a new value to the internal configuration data of the NXT firmware. You must also set the appropriate bit on the UpdateFlags property. The flags are defined in the description of the UpdateFlags property and the descriptions of properties that require flags to be set specify which flags are applicable.

Port

U8

NXTOutput: write-only

NXTOutputMulti: unsupported

Required

Legal value range: [0, 2]

This property specifies the physical port to which all other property read/writes apply. Note that legal values are zero-based, such that the values 0, 1, 2, and 3 correspond to the physical port labels “1”, “2”, “3”, and “4”, respectively.

Every NXTOutput Property Node must include one and only one Port property.

PortList

U8 array

NXTOutput: unsupported

NXTOutputMulti: write-only

Required

Legal value range: [0, 2] (arrays of 1, 2, or 3 elements in this range)

This property specifies a list of physical ports to which all other property writes apply. Note that the values 0, 1, and 2 correspond to the physical port labels “A”, “B”, and “C”, respectively.

Every NXTOutputMulti Property Node must include one and only one PortList property.

UpdateFlags

U8

NXTOutput: read-write

NXTOutputMulti: write-only

Optional

Bitfield including any combination of the following flag bits:

0x01	*UPDATE_MODE*	*Commits changes to the Mode property.*
0x02	*UPDATE_SPEED*	*Commits changes to the Power property.*
0x04	*UPDATE_TACHO_LIMIT*	*Commits changes to the TachoLimit property (interactive motors only).*
0x08	*UPDATE_RESET_COUNT*	*Resets internal movement counters and cancels current goal (interactive motors only).*
0x10	*UPDATE_PID_VALUES*	*Commits changes to PID regulation parameters RegPValue, RegIValue, and/or RegDValue (interactive motors only).*
0x20	*UPDATE_RESET_BLOCK_COUNT*	*Resets block-relative position counter (interactive motors only).*
0x40	*UPDATE_RESET_ROTATION_COUNT*	*Resets program-relative position counter (interactive motors only).*

This property is an unsigned byte bitfield with zero or more of the bit values above set.

You can use UPDATE_MODE, UPDATE_SPEED, UPDATE_TACHO_LIMIT, and UPDATE_PID_VALUES in conjunction with other properties to commit changes to the internal state of the NXT firmware module. That is, you must set the appropriate flags after setting one or more of these properties before the changes actually take affect. For example, write a value of 0x03 (UPDATE_MODE | UPDATE_SPEED) to UpdateFlags immediately after you write new values to the Mode and Power properties.

The "reset" flags are independent of other properties and produce the side effects described above.

For UPDATE_RESET_BLOCK_COUNT, "block-relative" refers to the way this flag is used in LEGO MINDSTORMS NXT Software 1.0. By convention, this flag is set every time an NXT-G motor control block starts execution. This convention means that the BlockTachoCount property always provides a position measurement relative to the last NXT-G motor control block to execute. The VIs included with the toolkit follow the same convention, but you do not need to follow this convention.

For UPDATE_RESET_ROTATION_COUNT, "program-relative" refers to the fact that this position counter is reset automatically at the beginning of every program. Set UPDATE_RESET_ROTATION_COUNT to reset this counter manually during program execution.

Mode

U8

NXTOutput: read-write

NXTOutputMulti: write-only

Optional

Bitfield including any combination of the following flag bits:

0x01	*MOTORON*	*Enables PWM power to port(s) according to value of the Power property.*
0x02	*BRAKE*	*Applies electronic braking to port(s).*
0x04	*REGULATED*	*Enables active power regulation according to value of RegMode (interactive motors only).*

This property is an unsigned byte bitfield with zero or more mode bits set. Clearing all bits (a Mode value of 0x00) is considered COAST mode; motors connected to the specified port(s) will rotate freely.

You must set the MOTORON bit for the NXT firmware to provide any power to the specified output port(s). Power is provided as a PWM waveform modulated by the Power property.

The BRAKE bit enables application of electronic braking to the circuit. "Braking" in this sense means that the output voltage is not allowed to float between active PWM pulses. Electronic braking improves the accuracy of motor output, but uses slightly more battery power.

You must use the REGULATED bit in conjunction with the RegMode property. Refer to the *RegMode* section for more information about using the REGULATED bit with the RegMode property.

You must set the UPDATE_MODE bit in the UpdateFlags bitfield to commit changes to the Mode property.

RegMode

U8

NXTOutput: read-write

NXTOutputMulti: write-only

Optional

Interactive motors only

Legal Values:

0x00	*REG_IDLE*	*Disables regulation.*
0x01	*REG_SPEED*	*Enables speed regulation.*
0x02	*REG_SYNC*	*Enables synchronization between any two motors.*

This property specifies the regulation mode to use with the specified port(s).

This property is ignored if you do not set the REGULATED bit in the Mode property. Unlike the Mode property, RegMode is not a bitfield. You can set only one RegMode value at a time.

Speed regulation means that the NXT firmware attempts to maintain a certain speed according to the Power set-point. To accomplish this, the NXT firmware automatically adjusts the actual PWM duty cycle if the motor is affected by a physical load. This automatic adjustment is reflected by the value of the ActualSpeed property.

Synchronization means that the firmware attempts keep any two motors in synch regardless of varying physical loads. A common usage of this mode is to maintain a straight path for a vehicle robot automatically, but you also can use this mode with the TurnRatio property to provide proportional turning. You must set REG_SYNC on at least two motor ports to have the desired affect. If REG_SYNC is set on all three motor ports, only the first two (A & B) are synchronized.

TurnRatio

I8

NXTOutput: read-write

NXTOutputMulti: write-only

Optional

Interactive motors only

Legal value range: [-100, 100]

This property specifies the proportional turning ratio for synchronized turning using two motors.

You must set some other properties appropriately on at least two motor ports for TurnRatio to have the desired effect. Use the following guidelines when setting this property:

- Mode bitfield must include MOTORON and REGULATED bits. BRAKE bit is optional.
- RegMode must be set to REG_SYNCH.
- RunState must be set to a non-IDLE value.
- Power must be set to a non-zero value.

After you set these property values, the NXT firmware uses the TurnRatio value to adjust relative power settings for the left and right motors automatically.

“Left” and “right” refer to the physical arrangement of the output plugs on an NXT brick (when facing the display screen). The following table includes the only three valid combinations of left and right motors that you can use with TurnRatio:

Left	**Right**
Output Port A	Output Port B
Output Port B	Output Port C
Output Port A	Output Port C

Note that this definition of “left” and “right” is independent of the LEGO model in use.

Negative TurnRatio values shift power towards the left motor, while positive TurnRatio values shift power towards the right motor. In both cases, the actual power applied is proportional to the Power set-point, such that an absolute value of 50% for TurnRatio normally results in one motor stopping, and an absolute value of 100% for TurnRatio normally results in the two motors turning in opposite directions at equal power.

RunState

U8

NXTOutput: read-write

NXTOutputMulti: write-only

Optional

Legal Values:

0x00	*RUN_STATE_IDLE*	*Disables power to specified port(s).*
0x10	*RUN_STATE_RAMPUP*	*Enables automatic ramp-up to the Power set-point (interactive motors only).*
0x20	*RUN_STATE_RUNNING*	*Enables power to specified port(s) at the Power set-point.*
0x40	*RUN_STATE_RAMPDOWN*	*Enables automatic ramp-down to the Power set-point (interactive motors only).*

This property specifies an auxiliary "state" to use with Mode, RegMode, and Power on the specified port(s). Set only one of the legal values at a given time.

RUN_STATE_RUNNING is the most common setting. Use RUN_STATE_RUNNING to enable power to any output device connected to the specified port(s).

RUN_STATE_RAMPUP enables automatic ramping to a new Power set-point that is greater than the current Power set-point. When you use RUN_STATE_RAMPUP in conjunction with appropriate TachoLimit and Power values, the NXT firmware attempts to increase the actual power smoothly to the new Power set-point over the number of degrees specified by TachoLimit.

RUN_STATE_RAMPDOWN enables automatic ramping to a new Power set-point that is less than the current Power set-point. When you use RUN_STATE_RAMPDOWN in conjunction with appropriate TachoLimit and Power values, the NXT firmware attempts to smoothly decrease the actual power to the new Power set-point over the number of degrees specified by TachoLimit.

Power

I8

NXTOutput: read-write

NXTOutputMulti: write-only

Optional

Legal value range: [-100, 100]

This property specifies the power level set-point for the specified port(s).

The absolute value of Power is used as a percentage of the full power capabilities of the motor.

The sign of Power specifies rotation direction. Positive values for Power instruct the firmware to turn the motor forward, while negative values instruct the firmware to turn the motor backward. “Forward” and “backward” are relative to a standard orientation for a particular type of motor.

Note that direction is not a meaningful concept for outputs like lamps. Lamps are only affected by the absolute value of Power.

You must set some other properties appropriately for the Power setpoint to have the desired effect. Use the following guidelines when setting this property:

- Mode bitfield must include MOTORON bit. BRAKE bit is optional.
- RunState must be set to a non-IDLE value.

You must set the UPDATE_SPEED bit in the UpdateFlags bitfield to commit changes to the Power property.

ActualSpeed

I8

NXTOutput: read-only

NXTOutputMulti: unsupported

Optional

Interactive motors only

Legal value range: [-100, 100]

This read-only property returns the actual percentage of full power that the NXT firmware is applying to the output currently. This value can vary from the current Power set-point when the internal auto-regulation code of the NXT firmware responds to drag on the output axle.

TachoLimit

U32

NXTOutput: read-write

NXTOutputMulti: write-only

Optional

Interactive motors only

Legal value range: [0, 4294967295]

This property specifies the rotational distance in degrees that you want to turn the motor.

Set the UPDATE_TACHO_LIMIT flag to commit changes to TachoLimit. The NXT firmware treats the new TachoLimit value as a relative distance from the motor position at the moment that the UPDATE_TACHO_LIMIT flag is processed. Remember that the sign of the Power property specifies the direction of rotation.

TachoCount

I32

NXTOutput: read-only

NXTOutputMulti: unsupported

Optional

Interactive motors only

Legal value range: [-2147483648, 2147483647]

This read-only property reports the internal position counter value for the specified port. This internal count is reset automatically when a new goal is set using the TachoLimit and the UPDATE_TACHO_LIMIT flag.

Set the UPDATE_RESET_COUNT flag in UpdateFlags to request that the NXT firmware resets TachoCount and cancels any current programmed goals.

The sign of TachoCount specifies rotation direction. Positive values correspond to forward rotation while negative values correspond to backward rotation. "Forward" and "backward" are relative to a standard orientation for a particular type of motor.

BlockTachoCount

I32

NXTOutput: read-only

NXTOutputMulti: unsupported

Optional

Interactive motors only

Legal value range: [-2147483648, 2147483647]

This read-only property reports the block-relative position counter value for the specified port.

Refer to the *UpdateFlags* section for more information about using block-relative position counts.

Set the UPDATE_RESET_BLOCK_COUNT flag in UpdateFlags to request that the firmware resets BlockTachoCount.

The sign of BlockTachoCount specifies the rotation direction. Positive values correspond to forward rotation while negative values correspond to backward rotation. “Forward” and “backward” are relative to a standard orientation for a particular type of motor.

RotationCount

I32

NXTOutput: read-only

NXTOutputMulti: unsupported

Optional

Interactive motors only

Legal value range: [-2147483648, 2147483647]

This read-only property reports the program-relative position counter value for the specified port.

Refer to the *UpdateFlags* section for more information about using program-relative position counts.

Set the UPDATE_RESET_ROTATION_COUNT flag in UpdateFlags to request that the NXT firmware resets the RotationCount property.

The sign of RotationCount specifies rotation direction. Positive values correspond to forward rotation while negative values correspond to backward rotation. “Forward” and “backward” are relative to a standard orientation for a particular type of motor.

Overloaded

Boolean

NXTOutput: read-only

NXTOutputMulti: unsupported

Optional

Interactive motors only

Legal values: TRUE, FALSE

This read-only property returns TRUE if the speed regulation functionality of the NXT firmware is unable to overcome physical load on the motor (i.e. the motor is turning more slowly than expected). Otherwise, this property returns FALSE.

You must set some other appropriately for the value of Overloaded to be meaningful. Use the following guidelines when setting this property:

- Mode bitfield must include MOTORON and REGULATED bits.
- RegMode must be set to REG_SPEED.
- RunState must be set to a non-IDLE value.

RegPValue

U8

NXTOutput: read-write

NXTOutputMulti: write-only

Optional

Interactive motors only

Legal value range: [0, 255]

This property specifies the proportional term used in the internal PID control algorithm.
Set the UPDATE_PID_VALUES to commit changes to RegPValue, RegIValue, or RegDValue.

RegIValue

U8

NXTOutput: read-write

NXTOutputMulti: write-only

Optional

Interactive motors only

Legal value range: [0, 255]

This property specifies the integral term used in the internal PID control algorithm.
Set the UPDATE_PID_VALUES to commit changes to RegPValue, RegIValue, or RegDValue.

RegDValue

U8

NXTOutput: read-write

NXTOutputMulti: write-only

Optional

Interactive motors only

Legal value range: [0, 255]

This property specifies the derivative term used in the internal PID control algorithm.
Set the UPDATE_PID_VALUES to commit changes to RegPValue, RegIValue, or RegDValue.

Built-in I/O

The toolkit provides access to several kinds of built-in I/O, including the following:

- Manipulating files in a simple flash file system using the **File Access Methods**
- Drawing to the NXT display using the **NXT Display Methods**
- Reading the built-in buttons on the NXT brick using the **NXT Button Method**
- Playing back recorded or synthesized sound using the **Sound Playback Methods**
- Communication with connected Bluetooth peers using the **Bluetooth Communications Methods**

To use any of these built-in I/O features in a program, connect an NXTSyscall refnum to the reference input of any Invoke Node then select the appropriate method.

File Access Methods

Use the file access methods to create, modify, rename, or delete files stored on the NXT brick.

The NXT firmware organizes all files in a simple flat file system stored in the flash memory on the NXT brick. Files are listed and referred to by name, and files can have any size up to the amount of available flash. The file system is flat because the file system does not support organizing files into hierarchical folders. NXT firmware 1.03 allows you to create up to 63 files (assuming flash space is available).

NXT filenames include up to 15 characters for the main name, a dot, and 3 characters for the extension (which specify the file type). This convention is referred to as “15.3 filenames”. A file extension serves as a cue to the NXT firmware as to how and where files are listed and treated.

The NXT firmware 1.03 and the LEGO MINDSTORMS NXT Software 1.0 use the following main file extensions:

- RXE: Executable files compiled from NXT-G, LabVIEW, or another compatible programming environment
- RPG: Simple 5-step programs created using the “NXT Program” UI on the NXT brick
- RTM: Built-in “Try Me” programs
- RIC: Image (graphic) files for use with the NXTDrawGraphic system call method in RXE programs
- RSO: Sound files
- SYS: Internal NXT firmware files
- CAL: Sensor calibration file
- TXT: ASCII text file using CR/LF (Windows) end-of-line convention

When manipulating files with a program, you can create, modify, rename, or delete any file in the system. Typical NXT programs use simple text files with the TXT extension. You then can upload these files easily to a PC, which then can read the files. You can use any extension or file encoding you choose, but make sure not to interfere with the various files that the NXT firmware and/or other programs use.

The default NXT firmware configuration includes several built-in files. You can delete any of these files to free up space, but you lose any associated functionality until you restore these files.

To manipulate files, wire an NXTSyscall refnum to any Invoke Node, then choose a method beginning with “NXTFile”. The following sections list these individual methods along with some concepts with which you need to be familiar.

File Handles

You must open a handle to a file before you can use read or write methods on the file. The NXTFileOpenRead, NXTFileOpenWrite, or NXTFileOpenAppend methods return a unique handle value. The NXT firmware also automatically registers open handles in an internal table such that the NXTResolveHandle system call method can look up any open handle by filename.

You must close each file handle with the NXTFileClose method before using the file to which it refers for any other purpose. When a program ends, the NXT firmware automatically closes any handles left open by the program.

The NXT firmware restricts the maximum number of concurrently open file handles to 16. Note that running programs uses up one file handle, and any current sound playback or other background process can take up additional file handles. In practice, a program can generally open 10-12 handles, but opening that many handles is fairly rare.

File Access Status Codes

File access system call methods return a different set of status codes from most other system call methods. File access status codes are unsigned 16-bit integers rather than signed 8-bit integers. In most cases, you only need to check if file access methods return status codes other than zero (SUCCESS), but the table below lists specific codes which you might see in your programs.

0x0000	*SUCCESS*	*Operation succeeded.*
0x8100	*NO_HANDLES*	*The firmware is not able to allocate any more file handles for this operation.*
0x8300	*NO_FILES*	*The firmware is not able to create any more files.*
0x8400	*PARTIAL_WRITE*	*Write request exceeded space available in file; partial write performed.*
0x8500	*EOF*	*File operation reached end of file.*
0x8700	*FILE_NOT_FOUND*	*Specified file not found.*
0x8800	*FILE_CLOSED*	*Specified file or handle is already closed.*
0x8900	*NO_LINEAR_SPACE*	*The firmware is not able to allocate requested linear file system space for specified file.*
0x8A00	*GENERIC_ERROR*	*A generic (unspecified) error condition occurred.*
0x8B00	*FILE_BUSY*	*Cannot acquire file handle for specified file because some other operation has opened the file.*
0x8C00	*NO_WRITE_BUFFERS*	*Cannot open file for write operation because all write buffers are already in use. Only four files can be open for write simultaneously.*
0x8D00	*ILLEGAL_APPEND*	*Cannot open file for append.*
0x8E00	*FILE_FULL*	*Allocated space for specified file is full (no more write operations allowed).*
0x8F00	*FILE_EXISTS*	*Failure to create or rename file due to name collision.*
0x9200	*ILLEGAL_FILE_NAME*	*Illegal file name provided. Ensure file name consists of 15.3 (or fewer) printable characters.*
0x9300	*ILLEGAL_HANDLE*	*Allocated space for specified file is full.*

File Access Performance Issues

Writing to flash memory is very slow compared to RAM access on the NXT brick. Writing to flash memory also ties up the CPU. File write operations are buffered in RAM whenever possible, but the firmware is subject to pauses of 6-30 ms while these buffers are committed to flash. If you experience significant delays during program operations, consider minimizing the amount of flash writing you perform.

Most file access method calls are subject to delays averaging 6 ms or less, with NXTFileOpenWrite subject to the most delay per call. The following sections describe the methods and include specific notes about potential flash writing performance issues where applicable.

NXTFileOpenRead

Return Value:

Status Code	*U16*

Parameters:

File Handle	*U8*	*output*	*Unique handle to identify open file*
Filename	*string*	*input*	*Maximum of 19 characters (15.3 filename)*
Length	*U32*	*output*	*Length of file in bytes*

This system call method attempts to open the file specified by Filename for read operations.

If the return value is zero, the file open operation succeeded. File Handle is assigned a unique handle for use with NXTFileRead and NXTFileClose, and Length is assigned the current length of the file. Furthermore, this file is registered for use with NXTFileResolveHandle.

If the return value is non-zero, an error occurred attempting to open the file. You can ignore File Handle and Length. This file is not registered for use with NXTFileResolveHandle.

NXTFileOpenWrite

Return Value:

Status Code	*U16*

Parameters:

File Handle	*U8*	*output*	*Unique handle to identify open file*
Filename	*string*	*input*	*Maximum of 19 characters (15.3 filename)*
Length	*U32*	*input-output*	*Length of file in bytes*

This system call method attempts to create a file with the name specified by Filename and size in bytes specified by Length, then keep the file open for write operations. You must specify the total file size using the Length parameter when you create the file. If you do not use all of the memory allocated for a particular file before closing the file handle, you can open the file for further write operations using the NXTFileOpenAppend system call method.

If the return value is zero, the file creation operation was successful. The File Handle output is assigned a unique handle for use with NXTFileWrite and NXTFileClose and the Length output is assigned the current length of the file. Furthermore, this file is registered for use with NXTFileResolveHandle.

If the return value is non-zero, an error occurred attempting to open the file. You can ignore the File Handle. This file is not registered for use with NXTFileResolveHandle.

Only four files can be concurrently opened for write operations. This limit includes files opened with NXTFileOpenAppend.

Note that file creation involves writing data to flash, so this system call method is subject to the performance issues described above. NXTFileOpenWrite is potentially subject to the most flash writing delay of any method. Creating a very large file can result in a delay of up to 30ms.

NXTFileOpenAppend

Return Value:

Status Code	*U16*

Parameters:

File Handle	*U8*	*output*	*Unique handle to identify open file*
Filename	*string*	*input*	*Maximum of 19 characters (15.3 filename)*
Length	*U32*	*output*	*Number of bytes remaining in file*

This system call method attempts to open the file specified by Filename for append (write) operations. This operation is useful only if a file already exists and has been closed before all of its allocated space has been filled.

If the return value is zero, the file open operation succeeded. The File Handle output contains a unique handle for use with NXTFileWrite and NXTFileClose and the Length output contains the number of unused bytes remaining in the file. The internal file cursor is set automatically to the end of existing data such that future calls to NXTFileWrite do not overwrite any data. Furthermore, this file is registered for use with NXTFileResolveHandle.

If the return value is non-zero, an error occurred attempting to open the file. You can ignore the File Handle. This file is not registered for use with NXTFileResolveHandle.

Only four files may be concurrently opened for write operations. This limit includes files opened with NXTFileOpenAppend.

NXTFileResolveHandle

Return Value:

Status Code	*U16*

Parameters:

File Handle	*U8*	*output*	*Unique handle to identify open file*
Write Handle?	*Boolean*	*output*	*TRUE if returned handle is for write operations.*
Filename	*string*	*input*	*Maximum of 19 characters (15.3 filename)*

The NXT firmware has a list of open file handles. This system call method searches the list of open file handles by Filename. To succeed, Filename must contain the exact filename of an already opened file.

If the return value is zero, the file is already open. The File Handle output is assigned a unique handle for use with NXTFileRead/NXTFileWrite and NXTFileClose. If the Boolean output Write Handle? is set to TRUE, the file is open for write operations. Otherwise the file is open for read operations.

If the return value is non-zero, the file is not yet open. You can ignore File Handle and Write Handle?. If you intend to use the file, call the appropriate open method for the file.

NXTFileClose

Return Value:

Status Code	*U16*

Parameters:

File Handle	*U8*	*input*	*Unique handle to identify open file*

This system call method closes the handle specified by File Handle.

If the return value is zero, the method successfully closed the file handle and removed the file from the list of open files that NXTResolveHandle uses.

If the return value is non-zero, the close operation failed. Either the specified file handle was invalid or no file was currently open using that file handle.

All files opened by an NXT program are automatically closed when the program finishes (or is aborted).

Note that a successful NXTFileClose operation commits any pending file write buffers to flash memory, so this operation is subject to the performance issues described above.

NXTFileRead

Return Value:

Status Code	*U16*

Parameters:

File Handle	*U8*	*input-output*	*Unique handle to identify open file*
Buffer	*string*	*output*	*File data in string format*
Length	*U32*	*input-output*	*Length of file in bytes*

This system call method attempts to read Length bytes of data from the file opened with the handle specified by File Handle.

If the return value is zero, the file read operation succeeded. Buffer contains Length bytes of file data in string format. If the file contains text data, use the string like a normal text string (i.e. for display to the screen). If the file contains flattened binary data, use the Unflatten from String function to unflatten it.

If the return value is non-zero, an error occurred attempting to read bytes from the file. Note that Buffer might still contain valid data if the method encountered the end of the file before reading Length bytes. In this case, the Length output contains the count of bytes that were actually read.

Successive calls to NXTFileRead with the same file handle will read new data each time (i.e. each read operation advances the internal file read cursor).

NXTFileWrite

Return Value:

Status Code	*U16*

Parameters:

File Handle	*U8*	*input-output*	*Unique handle to identify open file*
Buffer	*string*	*input-output*	*File data in string format*
Length	*U32*	*input-output*	*Number of bytes to write (in); number of bytes written (out)*

This system call method attempts to write Length bytes of data to the file opened with the handle specified by File Handle. If you use all of the memory allocated for the specified file, the NXTFileWrite method writes partial contents of the Buffer data to the file. The method sets Length to the number of bytes actually written, but does not modify the Buffer. If you intend to write more data, you need to close this file and use NXTOpenWrite to create a new file.

If the return value is zero, the file write operation succeeded.

If the return value is non-zero, an error occurred attempting to write bytes from the file.

Successive calls to NXTFileWrite with the same file handle write new data each time. Each write operation advances the internal file write cursor.

Note that NXTFileWrite involves writing to flash memory, so this system call method is subject to the performance issues described above. Because NXTFileWrite often is called many times in quick succession (to stream data to a file), the NXT firmware provides some buffering to help minimize the performance cost. This buffering means that a 4-6 ms delay might occur for every 256 bytes written.

NXTFileDelete

Return Value:

Status Code	*U16*

Parameters:

Filename	*string*	*input*	*Maximum of 19 characters (15.3 filename)*

This system call method deletes the file specified by Filename.

If the return value is zero, the delete operation succeeded.

If the return value is non-zero, the delete operation failed. Either the specified file does not exist or an open handle is associated with the file.

Be careful to close any open handles associated with a file before deleting it.

Note that file deletion involves writing internal file system data to flash memory, so this system call method is subject to the performance issues described above.

NXTFileRename

Return Value:

Status Code	*U16*

Parameters:

Old Filename	*String*	*input*	*Maximum of 19 characters (15.3 filename)*
New Filename	*String*	*input*	*Maximum of 19 characters (15.3 filename)*

This system call method renames the file specified by Old Filename to New Filename.

If the return value is zero, the rename operation succeeded.

If the return value is non-zero, the rename operation failed. Either the specified file does not exist or an open handle is associated with the file.

Be careful to close any open handles associated with a file before renaming it.

Note that file renaming involves writing internal file system data to flash memory, so this system call method is subject to the performance issues described above.

NXT Display Methods

Use the display methods to draw text, points, shapes, or graphic files to the built-in display on the NXT brick.

NXTDrawText

Return Value:

Status Code	*I8*	*Unused*

Parameters:

Location	*cluster* *I16 (X)* *I16 (Y)*	*input-output*	*Coordinates from lower left of screen*
Text	*string*	*input*	*Text string to draw*
Options	*U32*	*input*	*Bitfield of draw options*

This system call method renders the string Text at the coordinates specified by Location. The method only renders printable characters – non-ASCII or non-printable characters are ignored. The method does not wrap text at the screen edges.

All draw coordinates are relative to the lower left corner of the screen on the NXT brick. The NXTDrawText method coerces y-coordinates to multiples of 8 (rounding down) such that text always appears on one of 8 distinct lines on the display.

Set the least significant bit of Options to 1 to clear the entire screen before drawing. If you do not set this bit, the method overlays the text on top of any pixels already drawn by the program.

The first drawing system call method to execute in a program automatically clears the screen regardless of the value of Options.

The return value is always zero.

NXTDrawPoint

Return Value:

Status Code	*I8*	*Unused*

Parameters:

Location	*cluster* *I16 (X)* *I16 (Y)*	*Input*	*Coordinates from lower left of screen*
Options	*U32*	*Input*	*Bitfield of draw options*

This system call method draws a single, black pixel at the coordinates that Location specifies.

All draw coordinates are relative to the lower left corner of the screen on the NXT brick.

Set the least significant bit of Options to 1 to clear the entire screen before drawing. If you do not set this bit, the method overlays the point on top of any pixels already drawn by the program.

The first drawing system call method to execute in a program automatically clears the screen regardless of the value of Options.

The return value is always zero.

NXTDrawLine

Return Value:

Status Code	*I8*	*Unused*

Parameters:

StartLocation	*cluster* *I16 (X)* *I16 (Y)*	*input-output*	*Coordinates from lower left of screen*
EndLocation	*cluster* *I16 (X)* *I16 (Y)*	*input-output*	*Coordinates from lower left of screen*
Options	*U32*	*Input*	*Bitfield of draw options*

This system call method draws a black line one pixel wide, from StartLocation to EndLocation.

All draw coordinates are relative to the lower left corner of the screen on the NXT brick.

Set the least significant bit of Options to 1 to clear the entire screen before drawing. If you do not set this bit, the method overlays the line on top of any pixels already drawn by the program.

The first drawing system call method to execute in a program automatically clears the screen regardless of the value of Options.

The return value is always zero.

NXTDrawCircle

Return Value:

Status Code	*I8*	*Unused*

Parameters:

Center	*cluster* *I16 (X)* *I16 (Y)*	*input-output*	*Coordinates from lower left of screen*
Radius	*U8*	*input*	*Radius, in pixels*
Options	*U32*	*input*	*Bitfield of draw options*

This system call method draws a circle outline specified by Center coordinates and Radius (in pixels).

All draw coordinates are relative to the lower left corner of the screen on the NXT brick.

Set the least significant bit of Options to 1 to clear the entire screen before drawing. If you do not set this bit, the method overlays the circle on top of any pixels already drawn by the program. The circle outline is transparent; the method never modifies pixels inside the circle.

The first drawing system call method to execute in a program automatically clears the screen regardless of the value of Options.

The return value is always zero.

NXTDrawRect

Return Value:

Status Code	*I8*	*Unused*

Parameters:

Location	*cluster* *I16 (X)* *I16 (Y)*	*input-output*	*Coordinates from lower left of screen*
Size	*cluster* *I16 (Width)* *I16 (Height)*	*input-output*	*Dimensions of rectangle, in pixels*
Options	*U32*	*input*	*Bitfield of draw options*

This system call method draws a rectangle outline with one corner at the coordinates specified by Location. The Size cluster specifies relative coordinates of the corner opposite from Location.

Set the least significant bit of Options to 1 to clear the entire screen before drawing. If you do not set this bit, the method overlays the rectangle on top of any pixels already drawn by the program. The rectangle outline is transparent; the method never modifies pixels inside the rectangle.

The first drawing system call method to execute in a program automatically clears the screen regardless of the value of Options.

The return value is always zero.

NXTDrawGraphic

Return Value:

Status Code	*I8*	

Parameters:

Location	*cluster* *I16 (X)* *I16 (Y)*	*input-output*	*Coordinates from lower left of screen*
Filename	*string*	*input*	*Maximum of 19 characters (15.3 filename)*
Variables	*I32 array*	*input*	*Optional parameters as defined by the RIC file.*
Options	*U32*	*input*	*Bitfield of draw options*

This system call method renders the RIC-format graphic file specified by Filename. Location specifies coordinates of the lower left corner of the rendered image.

All draw coordinates are relative to the lower left corner of the screen on the NXT brick.

The Variables argument specifies an array of arbitrary numeric parameters you certain RIC files might use. Most files ignore these variables.

Set the least significant bit of Options to 1 to clear the entire screen before drawing. If you do not set this bit, the method overlays the graphic on top of any pixels already drawn by the program.

The first drawing system call method to execute in a program automatically clears the screen regardless of the value of Options.

If return value is non-zero, an error occurred while attempting to draw the file. Either the specified file does not exist, or is not a valid RIC format file.

NXTSetScreenMode

Return Value:

Status Code	*I8*	

Parameters:

ScreenMode	*U32*	*input*	*New screen mode*

This system call method sets a new mode for the display screen on the NXT brick. The only valid mode for the NXT firmware 1.03 is RESTORE_NXT_SCREEN (value: 0). Use this screen mode to restore the default status screen after using any of the NXTDraw system call methods.

The return value is always zero.

NXT Button Method

Use the NXT button method to read the status of the built-in buttons on the NXT brick. Note that you can read only the top three buttons. The bottom button always aborts the program.

NXTReadButton

Return Value:

Status Code	*I8*	

Parameters:

Index	*U8*	*input*	*Button index: RIGHT, LEFT, or ENTER*
Pressed	*Boolean*	*output*	*TRUE if specified button is currently depressed*
Count	*U8*	*output*	*Number of times specified button has been pressed and released since last reset.*
Reset?	*Boolean*	*input*	*Set to TRUE to reset Count after reading state.*

This system call method reads the state of the built-in NXT button specified by Index (RIGHT, LEFT, or ENTER).

Set Reset? to TRUE to reset Count after reading the state.

If the status code is zero, Pressed returns TRUE when the button is depressed, and Count returns the number of times button has been depressed and released since last reset.

If the status code is non-zero, you specified an invalid value for Index.

Legal values for Index:

0x01	*RIGHT*	*Right arrow button*
0x02	*LEFT*	*Left arrow button*
0x03	*ENTER*	*Center square button*

Sound Playback Methods

Use the sound playback methods to control the sound module on the NXT brick.

NXTSoundPlayFile

Return Value:

Status Code	*I8*	*Unused*

Parameters:

Filename	*string*	*input*	*Maximum of 19 characters (15.3 filename)*
Loop?	*Boolean*	*input*	*Set to TRUE to enable automatic looping of sound file*
Volume	*U8*	*input*	*Volume of playback: [0, 4]*

This system call method starts playback of the sound file specified by Filename. The file must be a valid RSO-format sound file.

Set Loop? to TRUE to loop playback automatically and indefinitely without further system call methods.

The integer values for Volume correspond to the following behaviors:

0	*Sound playback disabled*
1	*25% of full volume*
2	*50% of full volume*
3	*75% of full volume*
4	*100% of full volume*

NXTSoundPlayTone

Return Value:

Status Code	*I8*	*Unused*

Parameters:

Frequency	*U16*	*input*	*Frequency of tone in Hertz*
Duration	*U16*	*input*	*Duration of tone in milliseconds*
Loop?	*Boolean*	*input*	*Set to TRUE to enable automatic looping of tone*
Volume	*U8*	*input*	*Volume of playback: [0, 4]*

This system call method starts playback of a tone specified by Frequency (Hz) and Duration (ms).

Set Loop? to TRUE to loop playback automatically and indefinitely without further system call methods.

The integer values for Volume correspond to the following behaviors:

0	*Sound playback disabled*
1	*25% of full volume*
2	*50% of full volume*
3	*75% of full volume*
4	*100% of full volume*

NXTSoundGetState

Return Value:

State	*U8*	

Parameters:

Flags	*U8*	*output*	*Bitfield of sound module flags*

This system call method reads the internal state and flags of the sound module of the NXT brick. If Flags is non-zero, the sound module has playback operations pending or in progress. Normally, you can ignore the return value.

The return value might report the following values:

0x00	*SOUND_IDLE*	*Sound module is idle, but there might be a pending request to playback sound*
0x02	*SOUND_FILE*	*Sound module is playing an RSO file*
0x03	*SOUND_TONE*	*Sound module is playing a tone*
0x04	*SOUND_STOP*	*Request to stop playback in progress*

Flags might report the following values:

0x00	*SOUND_FLAGS_IDLE*	*No flags set; sound module is completely idle*
0x01	*SOUND_FLAGS_UPDATE*	*A request for playback is pending*
0x02	*SOUND_FLAGS_RUNNING*	*Playback in progress*

NXTSoundSetState

Return Value:

State	*U8*	

Parameters:

State	*U8*	*input*	*New state for sound module*
Flags	*U8*	*input*	*Bitfield of sound module flags*

This system call method writes new State and Flags values to the sound module of the NXT brick. Use this system call method with caution because it directly influences the internals of the sound module on the NXT brick. Use this method only for stopping current playback by writing a new State value of SOUND_STOP.

The following value is the only value you can write to State:

0x04	*SOUND_STOP*	*Request to stop playback*

The following value is the only value you can write to Flags:

0x00	*SOUND_FLAGS_IDLE*	*No flags set*

Bluetooth Communication Methods

Use the Bluetooth communication methods to send packets of information to other devices connected to the NXT brick via Bluetooth. You also use these methods to access the messaging queue system of the NXT firmware.

The NXT firmware uses a master/slave serial port system for Bluetooth communication. You must designate one Bluetooth device as the master device before you run a program using Bluetooth. If the master device is the NXT brick, you can configure up to three slave devices using serial ports 1, 2, and 3 on the NXT brick. If the slave device is an NXT brick, port 0 on the NXT brick is reserved for the master device.

Programs running on the master NXT brick can send packets of data to any connected slave devices using the NXTCommBTWrite method. However, programs on slave devices cannot send packets to master devices. The firmware of slave NXT bricks automatically handles responses sent by programs on master devices.

Refer to the LEGO MINDSTORMS Web site at `mindstorms.lego.com` for the communications protocol documentation for Bluetooth.

This section also includes descriptions of the system call methods for accessing the *mailboxes*, or message queues, on the NXT brick. By using the direct command protocol, a master device can send messages to slave NXT bricks in the form of text strings addressed to a particular mailbox. Each mailbox on the slave NXT brick is a circular message queue holding up to five messages. Each message can be up to 58 bytes long.

To send messages from a master NXT brick to a slave brick, use NXTCommBTWrite on the master brick to send a MessageWrite protocol packet to the slave. Then, use NXTMessageRead on the slave brick to read the message. The slave NXT brick must be running a program when an incoming message packet is received. If no program is running, the slave NXT brick ignores the message, and the message is lost.

To exchange numeric data using the message system, use the OP_FLATTEN and OP_UNFLATTEN instructions to convert data to and from text strings.

NXTCommBTCheckStatus

Return Value:

Status Code	*I8*	

Parameters:

Connection	*U8*	*input*	*Connection (port), [0, 3]*

This system call method checks the status of the Bluetooth communication on the specified port.

If the return value is zero, the port is idle and the last operation (if any) did not cause any errors.

This method can return the following non-zero status codes:

32	*0x20*	*STAT_COMM_PENDING*	*Port is busy performing a transaction.*
-35	*0xDD*	*ERR_COMM_BUS_ERR*	*Last transaction failed*
-33	*0xDF*	*ERR_COMM_CHAN_INVALID*	*Specified port is out of range [0, 3]*
-32	*0xE0*	*ERR_COMM_CHAN_NOT_READY*	*Specified port is not properly configured*

If you see ERR_COMM_CHAN_NOT_READY, make sure a Bluetooth connection is configured on the specified port.

If you see STAT_COMM_PENDING, an operation is in progress, so do not interrupt it. Avoid calls to NXTCommBTWrite until NXTCommBTCheckStatus returns zero or a negative error code.

NXTCommBTWrite

Return Value:

Status Code	*I8*	

Parameters:

Connection	*U8*	*input*	*Connection (port), [0, 3]*
Buffer	*U8 array*	*input*	*Up to 256 bytes for writing to specified port*

This system call method copies data from the buffer input to an internal Bluetooth buffer and instructs the NXT firmware to send the data to the device configured on the specified port.

If the return value is zero, the method successfully started a communication transaction. Use NXTCommBTCheckStatus to monitor the status of the transaction.

This method can return the following non-zero status codes:

-33	*0xDF*	*ERR_COMM_CHAN_INVALID*	*Specified port is out of range*
-32	*0xE0*	*ERR_COMM_CHAN_NOT_READY*	*Specified port is busy or not properly configured*
-19	*0xED*	*ERR_INVALID_SIZE*	*Buffer size exceeded maximum size*

If you are sending data to another NXT brick, the buffer must contain a complete packet conforming to the NXT communication protocol.

Refer to the LEGO MINDSTORMS Web site at mindstorms.lego.com for the communications protocol documentation for Bluetooth.

NXTMessageRead

Return Value:

Status Code	*I8*	

Parameters:

QueueID	*U8*	*input*	*Mailbox queue, [0, 9]*
Remove (T)	*Boolean*	*input*	*If TRUE, remove message from specified queue after reading data*
Message	*string*	*output*	*Message data*

This system call method reads the oldest message available in the specified mailbox queue and optionally removes that message from the queue.

If the return value is zero, the specified message queue was not empty and the Message output contains the oldest message from the queue.

This method can return the following non-zero status codes:

64	*0x40*	*STAT_MSG_EMPTY_MAILBOX*	*Specified mailbox queue is empty*
-18	*0xEE*	*ERR_INVALID_QUEUE*	*Invalid mailbox queue specified*

If you are calling NXTMessageRead on a master NXT brick with slave devices connected, this method will also periodically check for outgoing messages on the slave devices by automatically exchanging Bluetooth protocol packets with slaves.

NXTMessageWrite

Return Value:

Status Code	*I8*	

Parameters:

QueueID	*U8*	*input*	*Mailbox queue, [0, 19]*
Message	*string*	*input*	*Message data*

This system call method writes a new message to the specified mailbox queue. If there are already five messages in the specified queue, the oldest message will be deleted.

If the return value is zero, the method succeeded in writing the message to the specified mailbox queue.

This method can return the following non-zero status codes:

-18	*0xEE*	*ERR_INVALID_QUEUE*	*Invalid mailbox queue specified*
-19	*0xED*	*ERR_INVALID_SIZE*	*Message too large*

If you are calling NXTMessageWrite on a slave NXT brick, use mailbox queues 10 through 19 as "outboxes". When the master NXT brick reads messages, it will check these upper 10 mailboxes for outgoing message on the slave device.

Low-Level System Methods

Use the low-level system methods to access miscellaneous low-level features of the NXT firmware system.

NXTGetStartTick

Return Value:

Program Start Tick	*U32*	

This system call method returns the value of the system millisecond timer corresponding to the start of execution of the current program. This system call method is useful for measuring the time difference relative to the start of program execution.

NXTKeepAlive

Return Value:

Sleep Time Limit	*U32*	

This system call method resets the internal sleep timer in the NXT firmware and returns the current time limit in milliseconds until the next automatic sleep. Use this method to keep the NXT brick from automatically turning off. Use the UI menu on the NXT brick to configure the sleep time limit.

NXTRandomNumber

Return Value:

Random Number	*I16*	

This system call method returns a signed 16-bit random number. Internally, the NXT firmware selects new random seeds after every twenty calls to this system call method.

NXTIOMapRead

Return Value:

Status Code	*I8*	

Parameters:

ModuleName	*string*	*input*	*Name of the firmware module*
Offset	*U16*	*input*	*Offset from beginning of the firmware module I/O map*
Count	*U16*	*input*	*Number of bytes to read*
Buffer	*U8 array*	*output*	*I/O map data*

This system call method reads internal firmware module state information. This method is reserved for internal use only.

NXTIOMapWrite

Return Value:

Status Code	*I8*	

Parameters:

ModuleName	*string*	*input*	*Name of the firmware module*
Offset	*U16*	*input*	*Offset from beginning of the firmware module I/O map*
Buffer	*U8 array*	*input*	*I/O map data*

This system call method writes internal firmware module state information. This method is reserved for internal use only.