Worldwide Technical Support and Product Information

ni.com

National Instruments Corporate Headquarters
11500 North Mopac Expressway  Austin, Texas 78759-3504  USA  Tel: 512 683 0100

Worldwide Offices
Australia 1800 300 800, Austria 43 662 457990-0, Belgium 32 (0) 2 757 0020, Brazil 55 11 3262 3599,
Canada 800 433 3488, China 86 21 5050 9800, Czech Republic 420 224 235 774, Denmark 45 45 76 26 00,
Finland 358 (0) 9 725 72511, France 01 57 66 24 24, Germany 49 89 7413130, India 91 80 41190000,
Israel 972 3 6393737, Italy 39 02 41309277, Japan 0120-527196, Korea 82 02 3451 3400,
Lebanon 961 (0) 1 33 28 28, Malaysia 1800 887710, Mexico 01 800 010 0793, Netherlands 31 (0) 348 433 466,
New Zealand 0800 553 322, Norway 47 (0) 66 90 76 60, Poland 48 22 328 90 10, Portugal 351 210 311 210,
Russia 7 495 783 6851, Singapore 1800 226 5886, Slovenia 386 3 425 42 00, South Africa 27 0 11 805 8197,
Spain 34 91 640 0085, Sweden 46 (0) 8 587 895 00, Switzerland 41 56 2005151, Taiwan 886 02 2377 2222,
Thailand 662 278 6777, Turkey 90 212 279 3031, United Kingdom 44 (0) 1635 523545

For further support information, refer to the Technical Support and Professional Services appendix. To comment on National Instruments documentation, refer to the National Instruments Web site at ni.com/info and enter the info code feedback.

© 2008–2009 National Instruments Corporation. All rights reserved.
Important Information

**Warranty**
The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

**Copyright**
Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

National Instruments respects the intellectual property of others, and we ask our users to do the same. NI software is protected by copyright and other intellectual property laws. Where NI software may be used to reproduce software or other materials belonging to others, you may use NI software only to reproduce materials that you may reproduce in accordance with the terms of any applicable license or other legal restriction.

**Trademarks**
National Instruments, NI, ni.com, and LabVIEW are trademarks of National Instruments Corporation. Refer to the Terms of Use section on ni.com/legal for more information about National Instruments trademarks.

LEGO and MINDSTORMS are trademarks of the LEGO Group. Other product and company names mentioned herein are trademarks or trade names of their respective companies.

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

**Patents**
For patents covering National Instruments products/technology, refer to the appropriate location: Help»Patents in your software, the patents.txt file on your media, or the National Instruments Patent Notice at ni.com/patents.

**WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS**
(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS Whose Failure To PERFORM CAN REASONABLY BE Expected TO CAUSE SIGNIFICANT INJURY TO A HUMAn.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE INCOMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATION DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED “SYSTEM FAILURES”). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFER S FROM NATIONAL INSTRUMENTS TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION THEREIN, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.
The following conventions are used in this manual:

» The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File**»**Page Setup**»**Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.

⚠ This icon denotes a note, which alerts you to important information.

**bold**

Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names front panel controls and indicators, dialog boxes, sections of dialog boxes, menu names, and palette names.

*italic*

Italic text denotes variables, emphasis, a cross-reference, or an introduction to a key concept. Italic text also denotes text that is a placeholder for a word or value that you must supply.

**monospace**

Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions.
Contents

Chapter 1  
Introduction

Chapter 2  
Extensions to LabVIEW

Analog Input Devices  
Input Port Properties  
Port  
SensorType  
SensorMode  
Raw  
Normalized  
Scaled
## Contents

- **InvalidData** ................................................................. 2-10
- **NXTColorSensorRead** ..................................................... 2-11

### Digital I/O Devices .......................................................... 2-12

- **Digital I/O Port Setup** .................................................... 2-12
- **Low Speed Communication Methods** ............................... 2-12
  - **NXTCommLSCheckStatus** .............................................. 2-14
  - **NXTCommLSWrite** .......................................................... 2-15
  - **NXTCommLSRead** .......................................................... 2-16

### Output Devices: Motors and Lamps ..................................... 2-17

#### Output Port Properties .................................................... 2-17

- **Port** ........................................................................ 2-19
- **PortList** ................................................................... 2-20
- **UpdateFlags** ............................................................... 2-21
- **Mode** ....................................................................... 2-23
- **RegMode** ................................................................. 2-24
- **TurnRatio** ................................................................. 2-25
- **RunState** ................................................................. 2-26
- **Power** ................................................................... 2-27
- **ActualSpeed** .............................................................. 2-28
- **TachoLimit** ............................................................... 2-29
- **TachoCount** .............................................................. 2-30
- **BlockTachoCount** ..................................................... 2-31
- **RotationCount** .......................................................... 2-32
- **Overloaded** .............................................................. 2-33
- **RegPValue** ............................................................... 2-34
- **RegIValue** ............................................................... 2-35
- **RegDValue** ............................................................... 2-36

### Built-In I/O ........................................................................ 2-37

#### File Access Methods ....................................................... 2-37

- **File Handles** .............................................................. 2-39
- **File Access Status Codes** ........................................... 2-40
- **File Access Performance Issues** .................................... 2-41
- **NXTFileOpenRead** ...................................................... 2-42
- **NXTFileOpenWrite** ............................................................. 2-43
- **NXTFileOpenAppend** ...................................................... 2-44
- **NXTFileResolveHandle** .................................................. 2-45
- **NXTFileClose** ........................................................... 2-46
- **NXTFileRead** ............................................................... 2-47
- **NXTFileWrite** .............................................................. 2-48
- **NXTFileDelete** ............................................................. 2-49
- **NXTFileRename** ............................................................. 2-50
- **NXTListFiles** ............................................................. 2-51
Contents

NXT Display Methods ................................................................. 2-52
    NXTDrawText ................................................................. 2-52
    NXTDrawPoint ............................................................... 2-53
    NXTDrawLine ................................................................. 2-54
    NXTDrawCircle ............................................................... 2-55
    NXTDrawRect ................................................................. 2-56
    NXTDrawGraphic ............................................................ 2-57
    NXTSetScreenMode ......................................................... 2-58

NXT Button Method ................................................................. 2-59
    NXTReadButton ............................................................. 2-59

Sound Playback Methods ......................................................... 2-60
    NXTSoundPlayFile ......................................................... 2-60
    NXTSoundPlayTone ....................................................... 2-61
    NXTSoundGetState ....................................................... 2-62
    NXTSoundSetState ....................................................... 2-63

Bluetooth Communication Methods .......................................... 2-64
    NXTCommBTCheckStatus .............................................. 2-65
    NXTCommBTWrite ....................................................... 2-66
    NXTBTPower ............................................................... 2-67
    NXTBTConnection ....................................................... 2-68
    NXTMessageRead ....................................................... 2-69
    NXTMessageWrite ....................................................... 2-70

Low-Level System Methods ..................................................... 2-71
    NXTGetStartTick ......................................................... 2-71
    NXTKeepAlive ............................................................. 2-72
    NXTRandomNumber ..................................................... 2-73
    NXTIOMapRead ........................................................... 2-74
    NXTIOMapWrite .......................................................... 2-75
    NXTSetSleepTimeout .................................................... 2-76

Using Dynamic System Calls .................................................... 2-77
    Creating Access to a Dynamic System Call ....................... 2-77
    Using the NXTDynamicSyscall Refnum ......................... 2-78

Appendix A
Technical Support and Professional Services
Introduction

The LEGO® MINDSTORMS® NXT software uses a graphical programming language called NXT-G that 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 LEGO MINDSTORMS.

The LabVIEW LEGO® MINDSTORMS® NXT Module requires LabVIEW 2009 and does not support the NXT-G blocks. Instead, the NXT Module includes a new high-level API built out of LabVIEW VIs that provide a more powerful functionality than the NXT-G blocks. Using the NXT Module with LabVIEW 2009 allows you to bypass the high-level limitations imposed by NXT-G. The module enables you to create advanced programs using VIs and download them to the NXT brick. Furthermore, LabVIEW provides a more advanced editing environment than the LEGO MINDSTORMS NXT software.

Note To develop blocks that you can import into the LEGO MINDSTORMS NXT software, you must use the LabVIEW Toolkit for LEGO MINDSTORMS NXT with LabVIEW 7.1.

This document details the NXT-specific modifications and extensions to LabVIEW provided by the module. Note that the module does not support all LabVIEW features. Therefore, you should carefully review all notes and warnings listed in this document when programming with the module.

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

Related Documentation

The following documents contain information that you may find helpful as you read this manual:

- Getting Started with the LabVIEW LEGO® MINDSTORMS® NXT Module
- LabVIEW Help
Assumptions

This document assumes that you are using the LabVIEW LEGO MINDSTORMS NXT Module with LabVIEW 2009. 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 listed data types
- 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 module 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 module supports.

This document assumes that you are using official LEGO or NI NXT firmware. Refer to the LEGO MINDSTORMS Web site at mindstorms.lego.com for more information about firmware updates.

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 and 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 module, that VI becomes the top-level VI of the corresponding program file on the NXT brick. The module compiles the top-level VI and all of its subVIs into one program file, and then downloads the program file to the NXT brick. You can think of the top-level VI and its subVIs as the complete source code for the resulting binary program file.

Note the implication that subVIs have no analogous feature 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 module compiler automatically organizes the data space of NXT programs and the NXT firmware manages the data space at run time.

Note 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.

Additionally, 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. Indexing an empty array also causes the program to abort.

Parallel Execution on the NXT Brick

NXT firmware supports parallel execution of VI code in a similar sense to the way that LabVIEW executes parallel code on a 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 module only supports a limited subset of the data types that LabVIEW supports.

**Supported Data Types**

The module 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
- Single-precision floating point numbers

**Unsupported Data Types**

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

- Enumerations
- Arrays of two or more dimensions (including tables)
- Reference numbers (refnums) of any type other than those the module provides
- Paths
- Waveform, digital waveform, or digital data tables
- 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 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 module provides a special kind of Boolean control, Sequence Flow. Use the following tips when working with the Sequence Flow control:

- The module identifies Sequence Flow controls by name. If the name of your Boolean control or indicator begins with Sequence Flow, the module compiler classifies that control or indicator as a Sequence Flow control.
- The module 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.
- No Sequence Flow controls, indicators, or wires generate corresponding data space items in the compiled NXT program.

**NXT Refnum Controls**

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

Refer to Chapter 2, *Extensions to LabVIEW*, for specific information on how to use each new class of refnum. Use the following tips when working with the NXT refnums:

- The module identifies NXT refnum controls by name. If the name of your generic refnum control or indicator begins with Generic Refnum Name, the module compiler classifies that control or indicator as an NXT refnum.
- 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 the new properties and methods that the module provides.
- 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 module does not support all LabVIEW functions, and some supported functions exhibit special behavior or limitations when you use them with the module. In general, any function that relies on one or more unsupported data types is itself unsupported.

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

**Numeric**

- Add
- Subtract
- Multiply
- Divide
- Quotient and Remainder (integer division)
- Increment
- Decrement
- Negate
- Square Root

**Conversion**

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

**Logic**

- And
- Or
- Exclusive Or
- Not
  - The Not function only supports Boolean inputs.
Chapter 1 Introduction

Strings

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

Arrays

The module 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
  - You cannot resize this function.
- Array Subset
  - You cannot resize this function.
- Replace Array Subset
  - You cannot resize this function.
- Initialize Array
  - You cannot resize this function.
- Build Array
  - You must enable the Concatenate Inputs option with array inputs.

Comparison

The following comparison functions 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 module does not support Bundle by Name.
• Unbundle
  – The module does not support Unbundle by Name.
• Select
• Tick Count
• Flatten to String
  – The module 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

Custom G-code Functions
The module compiler and firmware do not support some LabVIEW functions. The following custom functions appear similar to their corresponding functions in LabVIEW and provide similar functionality.

Numeric
• Sign
• Add Array Elements
• Multiply Array Elements
• Square
• Reciprocal
• Transcendental Functions
Chapter 1 Introduction

- Max and Min
- In Range and Coerce
- Generate Random Number

Boolean
- Not And
- Not Or
- Not Exclusive Or
- And Array Elements
- Or Array Elements

String
- Trim White Space
- Reverse String
- Replace Substring
- Search and Replace String
- To Lower Case
- To Upper Case
- Search and Split String
- Rotate String

Array
- Insert Into Array
- Reverse Array
- Rotate Array
- Search Array
- Sort Array

Structures

The module supports the following structures.

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 (the “..” 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. Remember the following limitations when using subVIs:

- You can pass only supported data types through subVI connector panes.
- The module 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.

Note 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 module 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 module 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 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 module provides extensions to LabVIEW in the form of five classes of *generic reference numbers*, or *refnums*, and an associated set of properties and/or methods with each class of refnums. Each property or method loosely corresponds to a particular piece of the I/O interface provided by the NXT bytecode interpreter.

The module provides the following five classes of generic refnums:

- NXTInput
- NXTOutput
- NXTOutputMulti
- NXTSyscall
- NXTDynamicSyscall

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

You can use the NXTInput, NXTOutput, and NXTOutputMulti refnums 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 on each port. Each additional Property Node line corresponds to some configuration field, such as device mode or measured value, for the specified port(s).

You use the NXTSyscall and NXTDynamicSyscall refnums with Invoke Nodes to expose system-level methods for access to various other features of the NXT firmware. Use the NXTSyscall refnum with methods that the LabVIEW LEGO MINDSTORMS NXT Module provides. Use the NXTDynamicSyscall refnum with custom methods that you create.
Note  Refer to the *Using Dynamic System Calls* section of this document for more information about using the NXTDynamicSyscall refnum.

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.

Note  The module does not support using 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**

The NXT brick features 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.

Analog input devices includes touch, light, sound, and color sensors.

**Input Port Properties**

To configure or read values from analog I/O devices, use the NXTInput interface. 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 module supports.
The Port property has the following considerations:

- 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 module compiler guarantees that 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 reads and 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>NO_SENSOR</td>
<td>No sensor configured</td>
</tr>
<tr>
<td>0x01</td>
<td>SWITCH</td>
<td>NXT or Legacy(^1) touch sensor</td>
</tr>
<tr>
<td>0x02</td>
<td>TEMPERATURE</td>
<td>Legacy temperature sensor</td>
</tr>
<tr>
<td>0x03</td>
<td>REFLECTION</td>
<td>Legacy light sensor</td>
</tr>
<tr>
<td>0x04</td>
<td>ANGLE</td>
<td>Legacy rotation sensor</td>
</tr>
<tr>
<td>0x05</td>
<td>LIGHT_ACTIVE</td>
<td>NXT light sensor with floodlight enabled</td>
</tr>
<tr>
<td>0x06</td>
<td>LIGHT_INACTIVE</td>
<td>NXT light sensor with floodlight disabled</td>
</tr>
<tr>
<td>0x07</td>
<td>SOUND_DB</td>
<td>NXT sound sensor; dB scaling</td>
</tr>
<tr>
<td>0x08</td>
<td>SOUND_DBA</td>
<td>NXT sound sensor; dBA scaling</td>
</tr>
<tr>
<td>0x09</td>
<td>CUSTOM</td>
<td>Unused</td>
</tr>
<tr>
<td>0x0A</td>
<td>LOWSPEED</td>
<td>I2C digital sensor</td>
</tr>
<tr>
<td>0x0B</td>
<td>LOWSPEED_9V</td>
<td>I2C digital sensor; 9V power</td>
</tr>
<tr>
<td>0x0C</td>
<td>HIGHSPEED</td>
<td>Unused</td>
</tr>
<tr>
<td>0x0D</td>
<td>COLORFULL</td>
<td>NXT color sensor in color detector mode</td>
</tr>
<tr>
<td>0x0E</td>
<td>COLORRED</td>
<td>NXT color sensor in light sensor mode with red light</td>
</tr>
<tr>
<td>0x0F</td>
<td>COLORGREEN</td>
<td>NXT color sensor in light sensor mode with green light</td>
</tr>
<tr>
<td>0x10</td>
<td>COLORBLUE</td>
<td>NXT color sensor in light sensor mode with blue light</td>
</tr>
<tr>
<td>0x11</td>
<td>COLORNONE</td>
<td>NXT color sensor in light sensor mode with no light</td>
</tr>
</tbody>
</table>

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 Legacy firmware, no default sensor modes are associated with each sensor type.\(^1\)

\(^1\) Legacy refers to the RCX programmable controller used in previous versions of LEGO MINDSTORMS.


## SensorMode

U8

Read-write

Optional

Legal values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>RAWMODE</td>
<td>Report scaled value equal to raw value.</td>
</tr>
<tr>
<td>0x20</td>
<td>BOOLEANMODE</td>
<td>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).</td>
</tr>
<tr>
<td>0x40</td>
<td>TRANSITIONCNTMODE</td>
<td>Report scaled value as number of transitions between TRUE and FALSE.</td>
</tr>
<tr>
<td>0x60</td>
<td>PERIODCOUNTERMODE</td>
<td>Report scaled value as number of transitions from FALSE to TRUE, then back to FALSE.</td>
</tr>
<tr>
<td>0x80</td>
<td>PCTFULLSCALEMODE</td>
<td>Report scaled value as percentage of full scale reading for configured sensor type.</td>
</tr>
<tr>
<td>0xA0</td>
<td>CELSIUSMODE</td>
<td>Scale TEMPERATURE reading to degrees Celsius.</td>
</tr>
<tr>
<td>0xC0</td>
<td>FAHRENHEITMODE</td>
<td>Scale TEMPERATURE reading to degrees Fahrenheit.</td>
</tr>
<tr>
<td>0xE0</td>
<td>ANGLESTEPMODE</td>
<td>Report scaled value as count of ticks on Legacy rotation sensor.</td>
</tr>
</tbody>
</table>

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.

You should use Normalized in conjunction with the InvalidData property. The Normalized value is only valid if InvalidData is FALSE.
**Scaled**

U16

Read-write

Optional

Legal value range depends on SensorMode, as listed in the following table:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Value Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAWMODE</td>
<td>[0, 1023]</td>
</tr>
<tr>
<td>BOOLEANMODE</td>
<td>[0, 1]</td>
</tr>
<tr>
<td>TRANSITIONCNTMODE</td>
<td>[0, 65535]</td>
</tr>
<tr>
<td>PERIODCOUNTERMODE</td>
<td>[0, 65535]</td>
</tr>
<tr>
<td>PCTFULLSCALEMODE</td>
<td>[0, 100]</td>
</tr>
<tr>
<td>CELSIUSMODE</td>
<td>[−200, 700] (readings in 10th of a degree Celsius)</td>
</tr>
<tr>
<td>FAHRENHEITMODE</td>
<td>[−400, 1580] (readings in 10th of a degree Fahrenheit)</td>
</tr>
<tr>
<td>ANGLESTEPMODE</td>
<td>[0, 65535]</td>
</tr>
</tbody>
</table>

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

You should read the Scaled property in conjunction with the InvalidData 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 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. Set InvalidData to TRUE in the same Property Node as you use to set SensorType and 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.
### NXTColorSensorRead

**Return Value:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port</td>
<td>U8</td>
<td>input</td>
<td>Port, [0–3]</td>
</tr>
<tr>
<td>SensorValue</td>
<td>U8</td>
<td>output</td>
<td>Detected color number, [1–black, 2–blue, 3–green, 4–yellow, 5–red, 6–white]</td>
</tr>
<tr>
<td>Raw</td>
<td>U16</td>
<td>output</td>
<td>Returns the 10-bit value last read from the analog to digital converter on this specified port, as an array of red, green, and blue data</td>
</tr>
<tr>
<td>Normalized</td>
<td>U16</td>
<td>output</td>
<td>Returns a 10-bit sensor reading scaled according to the current value of the SensorType property on the specified port as an array of red, green, and blue data</td>
</tr>
<tr>
<td>Scaled</td>
<td>I16</td>
<td>output</td>
<td>Returns a sensor reading as an array of red, green, and blue data, scaled according to the current sensor type and mode on the specified port</td>
</tr>
<tr>
<td>Invalid Data?</td>
<td>Boolean</td>
<td>output</td>
<td>Signifies whether the values of Raw, Normalized, and Scaled might be invalid due to sensor configuration changes</td>
</tr>
</tbody>
</table>

This system call method returns the value of the color sensor and accommodates the RGB data format.

**Note** This system call method is different from others in that it returns an array of values.

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

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>–32</td>
<td>0xE0</td>
<td>ERR_COMM_CHAN_NOT_READY</td>
</tr>
</tbody>
</table>
Digital I/O Devices

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 devices includes ultrasonic and temperature sensors.

Digital I/O Port Setup

First, use an NXTInput Property Node, 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 you connect to this sensor.

Remember that you also need to set the InvalidData property to TRUE after setting a new SensorType, then wait, using 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 module provides the following three methods for low speed digital communication: NXTCommLSWrite, NXTCommLSCheckStatus, and NXTCommLSRead.

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. 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 brick 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**

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>I8</td>
<td>output</td>
<td>Refer to the following status codes.</td>
</tr>
<tr>
<td>Port</td>
<td>U8</td>
<td>input</td>
<td>Port, [0, 3]</td>
</tr>
<tr>
<td>BytesReady</td>
<td>U8</td>
<td>input</td>
<td>Number of bytes ready for reading, if any</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>0x20</td>
<td>STAT_COMM_PENDING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Port is busy performing a transaction.</td>
</tr>
<tr>
<td>–35</td>
<td>0xDD</td>
<td>ERR_COMM_BUS_ERR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Last transaction failed (possible device failure).</td>
</tr>
<tr>
<td>–33</td>
<td>0xDF</td>
<td>ERR_COMM_CHAN_INVALID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specified port is out of range.</td>
</tr>
<tr>
<td>–32</td>
<td>0xE0</td>
<td>ERR_COMM_CHAN_NOT_READY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specified port is not properly configured.</td>
</tr>
</tbody>
</table>

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

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>I8</td>
<td>output</td>
<td>Refer to the following status codes.</td>
</tr>
<tr>
<td>Port</td>
<td>U8</td>
<td>input</td>
<td>Port, [0–3]</td>
</tr>
<tr>
<td>Buffer</td>
<td>U8 array</td>
<td>input</td>
<td>Up to 16 bytes for writing to device</td>
</tr>
<tr>
<td>ReturnLength</td>
<td>U8</td>
<td>input</td>
<td>Number of bytes expected from device in response to writing data in Buffer; maximum 16</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>–33</td>
<td>0xDF</td>
<td>ERR_COMM_CHAN_INVALID</td>
</tr>
<tr>
<td></td>
<td>ERR_COMM_CHAN_INVALID</td>
<td>Specified port is out of range.</td>
</tr>
<tr>
<td>–32</td>
<td>0xE0</td>
<td>ERR_COMM_CHAN_NOT_READY</td>
</tr>
<tr>
<td></td>
<td>ERR_COMM_CHAN_NOT_READY</td>
<td>Specified port is busy or not properly configured.</td>
</tr>
<tr>
<td>–19</td>
<td>0xED</td>
<td>ERR_INVALID_SIZE</td>
</tr>
<tr>
<td></td>
<td>ERR_INVALID_SIZE</td>
<td>Either buffer size or ReturnLength exceeded 16-byte maximum.</td>
</tr>
</tbody>
</table>
**NXTCommLSRead**

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>I8</td>
<td>output</td>
<td>Refer to the following status codes.</td>
</tr>
<tr>
<td>Port</td>
<td>U8</td>
<td>input</td>
<td>Port, [0–3]</td>
</tr>
<tr>
<td>Buffer</td>
<td>U8 array</td>
<td>output</td>
<td>Bytes read from device, if any</td>
</tr>
<tr>
<td>BufferLength</td>
<td>U8</td>
<td>input</td>
<td>Upper bound on number of bytes to read into Buffer; actual number returned limited to bytes available in the internal read buffer</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>0x20 STAT_COMM_PENDING</td>
<td>Port is busy performing a transaction.</td>
</tr>
<tr>
<td>−35</td>
<td>0xDD ERR_COMM_BUS_ERR</td>
<td>Last transaction failed (possible device failure).</td>
</tr>
<tr>
<td>−33</td>
<td>0xDF ERR_COMM_CHAN_INV</td>
<td>Specified port is out of range.</td>
</tr>
<tr>
<td>−32</td>
<td>0xE0 ERR_COMM_CHAN_NOT_READY</td>
<td>Specified port is not configured properly.</td>
</tr>
</tbody>
</table>

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, Legacy-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 is invalid because the lamp only responds 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 receives 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 synchronizes multiple motors tightly. Conversely, you can only read properties from one output at a time, so NXTOutput is useful for inspecting the state, such as 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 configure only 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 module 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.
Chapter 2  Extensions to LabVIEW

Use the following tips when working with 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.

**Note**  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 module 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 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>UPDATE_MODE</td>
<td>Commits changes to the Mode property</td>
</tr>
<tr>
<td>0x02</td>
<td>UPDATE_SPEED</td>
<td>Commits changes to the Power property</td>
</tr>
<tr>
<td>0x04</td>
<td>UPDATE_TACHO_LIMIT</td>
<td>Commits changes to the TachoLimit property (interactive motors only)</td>
</tr>
<tr>
<td>0x08</td>
<td>UPDATE_RESET_COUNT</td>
<td>Resets internal movement counters and cancels current goal (interactive motors only)</td>
</tr>
<tr>
<td>0x10</td>
<td>UPDATE_PID_VALUES</td>
<td>Commits changes to PID regulation parameters RegPValue, RegIValue, and/or RegDValue (interactive motors only)</td>
</tr>
<tr>
<td>0x20</td>
<td>UPDATE_RESET_BLOCK_COUNT</td>
<td>Resets block-relative position counter (interactive motors only)</td>
</tr>
<tr>
<td>0x40</td>
<td>UPDATE_RESET_ROTATION_COUNT</td>
<td>Resets program-relative position counter (interactive motors only)</td>
</tr>
</tbody>
</table>

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. 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 automatically reset the rotation value to zero when used.
For UPDATE_RESET_BLOCK_COUNT, block-relative refers to the way this flag is used in the LEGO MINDSTORMS NXT software. 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 module 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>MOTORON</td>
<td>Enables PWM power to port(s) according to value of the Power property</td>
</tr>
<tr>
<td>0x02</td>
<td>BRAKE</td>
<td>Applies electronic braking to port(s)</td>
</tr>
<tr>
<td>0x04</td>
<td>REGULATED</td>
<td>Enables active power regulation according to value of RegMode (interactive motors only)</td>
</tr>
</tbody>
</table>

This property is an unsigned byte bitfield with zero or more mode bits set. Clearing all bits with 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>REG_IDLE</td>
<td>Disables regulation</td>
</tr>
<tr>
<td>0x01</td>
<td>REG_SPEED</td>
<td>Enables speed regulation</td>
</tr>
<tr>
<td>0x02</td>
<td>REG_SYNC</td>
<td>Enables synchronization between any two motors</td>
</tr>
</tbody>
</table>

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 and B, are synchronized.
TurnRatio

<table>
<thead>
<tr>
<th>Left</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output Port A</td>
<td>Output Port B</td>
</tr>
<tr>
<td>Output Port B</td>
<td>Output Port C</td>
</tr>
<tr>
<td>Output Port A</td>
<td>Output Port C</td>
</tr>
</tbody>
</table>

Note that this definition of left and right is independent of the LEGO model you are using.

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>RUN_STATE_IDLE</td>
<td>Disables power to specified port(s)</td>
</tr>
<tr>
<td>0x10</td>
<td>RUN_STATE_RAMPUP</td>
<td>Enables automatic ramp-up to the Power set-point (interactive motors only)</td>
</tr>
<tr>
<td>0x20</td>
<td>RUN_STATE_RUNNING</td>
<td>Enables power to specified port(s) at the Power set-point</td>
</tr>
<tr>
<td>0x40</td>
<td>RUN_STATE_RAMPDOWN</td>
<td>Enables automatic ramp-down to the Power set-point (interactive motors only)</td>
</tr>
</tbody>
</table>

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** 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, such as the motor is turning more slowly than expected. Otherwise, this property returns FALSE.

You must set other properties 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. Refer to the Proportional-Integral-Derivative (PID) topic in the LabVIEW Help for more information.

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 module 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 and 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 1.05 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 specifies 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 1.05 and the LEGO MINDSTORMS NXT software 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
Chapter 2  Extensions to LabVIEW

- **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
- **LOG**—ASCII text file created by NXT datalogging

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 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 method concepts.
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.

Note When NXT firmware 1.26 or later automatically closes a file, the firmware removes all unused space to maximize free space on the device.

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 following table lists specific codes which you might see in your programs.

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

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>U16</td>
<td>output</td>
<td>Refer to the File Access Status Codes section.</td>
</tr>
<tr>
<td>File Handle</td>
<td>U8</td>
<td>output</td>
<td>Unique handle to identify open file</td>
</tr>
<tr>
<td>Filename</td>
<td>string</td>
<td>input</td>
<td>Maximum of 19 characters (15.3 filename)</td>
</tr>
<tr>
<td>Length</td>
<td>U32</td>
<td>output</td>
<td>Length of file in bytes</td>
</tr>
</tbody>
</table>

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**

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>U16</td>
<td>output</td>
<td>Refer to the <em>File Access Status Codes</em> section.</td>
</tr>
<tr>
<td>File Handle</td>
<td>U8</td>
<td>output</td>
<td>Unique handle to identify open file</td>
</tr>
<tr>
<td>Filename</td>
<td>string</td>
<td>input</td>
<td>Maximum of 19 characters</td>
</tr>
<tr>
<td>Length</td>
<td>U32</td>
<td>input-output</td>
<td>Length of file in bytes</td>
</tr>
</tbody>
</table>

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**  
File creation involves writing data to flash, so this system call method is subject to the performance issues previously described. 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 30 ms.
**NXTFileOpenAppend**

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>U16</td>
<td>output</td>
<td>Refer to the <em>File Access Status Codes</em> section.</td>
</tr>
<tr>
<td>File Handle</td>
<td>U8</td>
<td>output</td>
<td>Unique handle to identify open file</td>
</tr>
<tr>
<td>Filename</td>
<td>string</td>
<td>input</td>
<td>Maximum of 19 characters (15.3 filename)</td>
</tr>
<tr>
<td>Length</td>
<td>U32</td>
<td>output</td>
<td>Number of bytes remaining in file</td>
</tr>
</tbody>
</table>

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

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>U16</td>
<td>output</td>
<td>Refer to the File Access Status Codes section.</td>
</tr>
<tr>
<td>File Handle</td>
<td>U8</td>
<td>output</td>
<td>Unique handle to identify open file</td>
</tr>
<tr>
<td>Write Handle?</td>
<td>Boolean</td>
<td>output</td>
<td>TRUE if returned handle is for write operations.</td>
</tr>
<tr>
<td>Filename</td>
<td>string</td>
<td>input</td>
<td>Maximum of 19 characters (15.3 filename)</td>
</tr>
</tbody>
</table>

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

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>U16</td>
<td>output</td>
<td>Refer to the File Access Status Codes section.</td>
</tr>
<tr>
<td>File Handle</td>
<td>U8</td>
<td>input</td>
<td>Unique handle to identify open file</td>
</tr>
</tbody>
</table>

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 a 6–30 ms delay in performing the operation.
### NXTFileRead

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, such as the 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 reads new data each time. For example, each read operation advances the internal file read cursor.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>U16</td>
<td>output</td>
<td>Refer to the File Access Status Codes section.</td>
</tr>
<tr>
<td>File Handle</td>
<td>U8</td>
<td>input-output</td>
<td>Unique handle to identify open file</td>
</tr>
<tr>
<td>Buffer</td>
<td>string</td>
<td>output</td>
<td>File data in string format</td>
</tr>
<tr>
<td>Length</td>
<td>U32</td>
<td>input-output</td>
<td>Length of file in bytes</td>
</tr>
</tbody>
</table>
NXTFileWrite

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>U16</td>
<td>output</td>
<td>Refer to the File Access Status Codes section.</td>
</tr>
<tr>
<td>File Handle</td>
<td>U8</td>
<td>input-output</td>
<td>Unique handle to identify open file</td>
</tr>
<tr>
<td>Buffer</td>
<td>string</td>
<td>input-output</td>
<td>File data in string format</td>
</tr>
<tr>
<td>Length</td>
<td>U32</td>
<td>input-output</td>
<td>Number of bytes to write (in); number of bytes written (out)</td>
</tr>
</tbody>
</table>

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: 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

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>U16</td>
<td>output</td>
<td>Refer to the File Access Status Codes section.</td>
</tr>
<tr>
<td>Filename</td>
<td>string</td>
<td>input</td>
<td>Maximum of 19 characters (15.3 filename)</td>
</tr>
</tbody>
</table>

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 a 6–30 ms delay in performing the operation.
NXTFileRename

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>U16</td>
<td>output</td>
<td>Refer to the File Access Status Codes section.</td>
</tr>
<tr>
<td>Old Filename</td>
<td>String</td>
<td>input</td>
<td>Maximum of 19 characters (15.3 filename)</td>
</tr>
<tr>
<td>New Filename</td>
<td>String</td>
<td>input</td>
<td>Maximum of 19 characters (15.3 filename)</td>
</tr>
</tbody>
</table>

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 a 6–30 ms delay in performing the operation.
NXTListFiles

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>I8</td>
<td>output</td>
<td></td>
</tr>
<tr>
<td>Pattern</td>
<td>String</td>
<td>input</td>
<td>File extension, such as *.rxe.</td>
</tr>
<tr>
<td>File List</td>
<td>String array</td>
<td>input-output</td>
<td>List of files on the NXT brick with the Pattern file extension.</td>
</tr>
</tbody>
</table>

This system call method lists the files on the NXT brick that have the file extension, or Pattern, you specify. You must wire an array to the File List input. This system call method then populates that array and returns the results in the File List output.

If the status code is zero, the method successfully returned the list of files on the NXT brick. If no files of the extension you specify exist on the NXT brick, the method returns an empty File List.

If the status code is non-zero, the method failed due to a memory error and did not return the list of files on the NXT brick.
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

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>I8</td>
<td>output</td>
<td>Unused</td>
</tr>
<tr>
<td>Location</td>
<td>cluster</td>
<td>input-output</td>
<td>Coordinates from lower left of screen</td>
</tr>
<tr>
<td></td>
<td>I16 (X)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>I16 (Y)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text</td>
<td>string</td>
<td>input</td>
<td>Text string to draw</td>
</tr>
<tr>
<td>Options</td>
<td>U32</td>
<td>input</td>
<td>Bitfield of draw options</td>
</tr>
</tbody>
</table>

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 eight by rounding down such that text always appears on one of eight 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

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

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Date Code</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>I8</td>
<td>output</td>
<td>Unused</td>
</tr>
<tr>
<td>StartLocation</td>
<td>cluster</td>
<td>input-output</td>
<td>Coordinates from lower left of screen</td>
</tr>
<tr>
<td></td>
<td>I16 (X)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>I16 (Y)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EndLocation</td>
<td>cluster</td>
<td>input-output</td>
<td>Coordinates from lower left of screen</td>
</tr>
<tr>
<td></td>
<td>I16 (X)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>I16 (Y)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options</td>
<td>U32</td>
<td>input</td>
<td>Bitfield of draw options</td>
</tr>
</tbody>
</table>

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**

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>I8</td>
<td>output</td>
<td>Unused</td>
</tr>
<tr>
<td>Center</td>
<td>cluster</td>
<td>input-output</td>
<td>Coordinates from lower left of screen</td>
</tr>
<tr>
<td></td>
<td>I16 (X)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>I16 (Y)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radius</td>
<td>U8</td>
<td>input</td>
<td>Radius, in pixels</td>
</tr>
<tr>
<td>Options</td>
<td>U32</td>
<td>input</td>
<td>Bitfield of draw options</td>
</tr>
</tbody>
</table>

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

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>I8</td>
<td>output</td>
<td>Unused</td>
</tr>
<tr>
<td>Location</td>
<td>cluster</td>
<td>input-output</td>
<td>Coordinates from lower left of screen</td>
</tr>
<tr>
<td>Size</td>
<td>cluster</td>
<td>input-output</td>
<td>Dimensions of rectangle, in pixels</td>
</tr>
<tr>
<td>Options</td>
<td>U32</td>
<td>input</td>
<td>Bitfield of draw options</td>
</tr>
</tbody>
</table>

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

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>I8</td>
<td>output</td>
<td>Unused</td>
</tr>
<tr>
<td>Location</td>
<td>cluster I16 (X) I16 (Y)</td>
<td>input-output</td>
<td>Coordinates from lower left of screen</td>
</tr>
<tr>
<td>Filename</td>
<td>string</td>
<td>input</td>
<td>Maximum of 19 characters (15.3 filename)</td>
</tr>
<tr>
<td>Variables</td>
<td>I32 array</td>
<td>input</td>
<td>Optional parameters as defined by the RIC file.</td>
</tr>
<tr>
<td>Options</td>
<td>U32</td>
<td>input</td>
<td>Bitfield of draw options</td>
</tr>
</tbody>
</table>

This system call method renders the RIC-format graphic file (the LEGO standard image format) 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 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

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>I8</td>
<td>output</td>
<td>Unused</td>
</tr>
<tr>
<td>ScreenMode</td>
<td>U32</td>
<td>input</td>
<td>New screen mode</td>
</tr>
</tbody>
</table>

This system call method sets a new mode for the display screen on the NXT brick. The only valid mode for the NXT firmware 1.05 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**

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>I8</td>
<td>output</td>
<td>Unused</td>
</tr>
<tr>
<td>Index</td>
<td>U8</td>
<td>input</td>
<td>Button index: RIGHT, LEFT, or ENTER</td>
</tr>
<tr>
<td>Pressed</td>
<td>Boolean</td>
<td>output</td>
<td>TRUE if specified button is currently depressed</td>
</tr>
<tr>
<td>Count</td>
<td>U8</td>
<td>output</td>
<td>Number of times specified button has been pressed and released since last reset.</td>
</tr>
<tr>
<td>Reset?</td>
<td>Boolean</td>
<td>input</td>
<td>Set to TRUE to reset Count after reading state.</td>
</tr>
</tbody>
</table>

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 pressed and released since last reset.

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

Legal values for Index:

<table>
<thead>
<tr>
<th>Value</th>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>RIGHT</td>
<td>Right arrow button</td>
</tr>
<tr>
<td>0x02</td>
<td>LEFT</td>
<td>Left arrow button</td>
</tr>
<tr>
<td>0x03</td>
<td>ENTER</td>
<td>Center square button</td>
</tr>
</tbody>
</table>
Sound Playback Methods

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

**NXTSoundPlayFile**

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>I8</td>
<td>output</td>
<td>Unused</td>
</tr>
<tr>
<td>Filename</td>
<td>string</td>
<td>input</td>
<td>Maximum of 19 characters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(15.3 filename)</td>
</tr>
<tr>
<td>Loop?</td>
<td>Boolean</td>
<td>input</td>
<td>Set to TRUE to enable automatic looping of sound file</td>
</tr>
<tr>
<td>Volume</td>
<td>U8</td>
<td>input</td>
<td>Volume of playback: [0, 4]</td>
</tr>
</tbody>
</table>

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

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:

<table>
<thead>
<tr>
<th>Integer Value</th>
<th>Value Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Sound playback disabled</td>
</tr>
<tr>
<td>1</td>
<td>25% of full volume</td>
</tr>
<tr>
<td>2</td>
<td>50% of full volume</td>
</tr>
<tr>
<td>3</td>
<td>75% of full volume</td>
</tr>
<tr>
<td>4</td>
<td>100% of full volume</td>
</tr>
</tbody>
</table>
NXTSoundPlayTone

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>I8</td>
<td>output</td>
<td>Unused</td>
</tr>
<tr>
<td>Frequency</td>
<td>U16</td>
<td>input</td>
<td>Frequency of tone in Hertz</td>
</tr>
<tr>
<td>Duration</td>
<td>U16</td>
<td>input</td>
<td>Duration of tone in milliseconds</td>
</tr>
<tr>
<td>Loop?</td>
<td>Boolean</td>
<td>input</td>
<td>Set to TRUE to enable automatic looping of tone</td>
</tr>
<tr>
<td>Volume</td>
<td>U8</td>
<td>input</td>
<td>Volume of playback: [0, 4]</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Integer Value</th>
<th>Value Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Sound playback disabled</td>
</tr>
<tr>
<td>1</td>
<td>25% of full volume</td>
</tr>
<tr>
<td>2</td>
<td>50% of full volume</td>
</tr>
<tr>
<td>3</td>
<td>75% of full volume</td>
</tr>
<tr>
<td>4</td>
<td>100% of full volume</td>
</tr>
</tbody>
</table>
NXTSoundGetState

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>U8</td>
<td>output</td>
<td>Sound module state</td>
</tr>
<tr>
<td>Flags</td>
<td>U8</td>
<td>output</td>
<td>Bitfield of sound module</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>SOUND_IDLE</td>
<td>Sound module is idle, but there might be a pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td>request to playback sound</td>
</tr>
<tr>
<td>0x02</td>
<td>SOUND_FILE</td>
<td>Sound module is playing an RSO file</td>
</tr>
<tr>
<td>0x03</td>
<td>SOUND_TONE</td>
<td>Sound module is playing a tone</td>
</tr>
<tr>
<td>0x04</td>
<td>SOUND_STOP</td>
<td>Request to stop playback in progress</td>
</tr>
</tbody>
</table>

Flags might report the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>SOUND_FLAGS_IDLE</td>
<td>No flags set; sound module is completely idle</td>
</tr>
<tr>
<td>0x01</td>
<td>SOUND_FLAGS_UPDATE</td>
<td>A request for playback is pending</td>
</tr>
<tr>
<td>0x02</td>
<td>SOUND_FLAGS_RUNNING</td>
<td>Playback in progress</td>
</tr>
</tbody>
</table>
NXTSoundSetState

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>U8</td>
<td>input</td>
<td>New state for sound module</td>
</tr>
<tr>
<td>Flags</td>
<td>U8</td>
<td>input</td>
<td>Bitfield of sound module flags</td>
</tr>
</tbody>
</table>

This system call method writes new State and Flags values to the sound module of the NXT brick.

**Note** 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x04</td>
<td>SOUND_STOP</td>
<td>Request to stop playback</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>SOUND_FLAGS_IDLE</td>
<td>No flags set</td>
</tr>
</tbody>
</table>
Bluetooth Communication Methods

Use the Bluetooth communication methods to send packets of information to other devices connected to the NXT brick through 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 device, use NXTCommBTWrite on the master device to send a MessageWrite protocol packet to the slave. Then, use NXTMessageRead on the slave device 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

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>I8</td>
<td>output</td>
<td>Refer to the following status codes.</td>
</tr>
<tr>
<td>Connection</td>
<td>U8</td>
<td>input</td>
<td>Connection (port), [0, 3]</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>0x20 STD_COMM_PENDING</td>
<td>Port is busy performing a transaction.</td>
</tr>
<tr>
<td>-35</td>
<td>0xDD ERR_COMM_BUS_ERR</td>
<td>Last transaction failed</td>
</tr>
<tr>
<td>-33</td>
<td>0xDF ERR_COMM_CHAN_INVALID</td>
<td>Specified port is out of range [0,3]</td>
</tr>
<tr>
<td>-32</td>
<td>0xE0 ERR_COMM_CHAN_NOT_READY</td>
<td>Specified port is not properly configured</td>
</tr>
</tbody>
</table>

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 and do not interrupt it. Avoid calls to NXTCommBTWrite until NXTCommBTCheckStatus returns zero or a negative error code.
**NXTCommBTWrite**

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>I8</td>
<td>output</td>
<td>Refer to the following status codes.</td>
</tr>
<tr>
<td>Connection</td>
<td>U8</td>
<td>input</td>
<td>Connection (port), [0, 3]</td>
</tr>
<tr>
<td>Buffer</td>
<td>U8 array</td>
<td>input</td>
<td>Up to 256 bytes for writing to specified port</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>–33</td>
<td>ERR_COMM_CHAN_INVALID</td>
<td>Specified port is out of range</td>
</tr>
<tr>
<td>–32</td>
<td>ERR_COMM_CHAN_NOT_READY</td>
<td>Specified port is busy or not properly configured</td>
</tr>
<tr>
<td>–19</td>
<td>ERR_INVALID_SIZE</td>
<td>Buffer size exceeded maximum size</td>
</tr>
</tbody>
</table>

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.
NXTBTPower

Return value:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power State</td>
<td>Boolean</td>
<td>input</td>
<td>Signifies on or off</td>
</tr>
</tbody>
</table>

This dynamic system call method turns the Bluetooth device power on or off.
NXTBTTConnection

Return value:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>U8</td>
<td>input</td>
<td>[0–Initiate connection, 1–Close connection]</td>
</tr>
<tr>
<td>Partner Name</td>
<td>string</td>
<td>input</td>
<td>Name of device</td>
</tr>
<tr>
<td>Connection</td>
<td>U8</td>
<td>input</td>
<td>Connection (port), [0, 3]</td>
</tr>
</tbody>
</table>

This dynamic system call method initiates or closes a connection to another device.
**NXTMessageRead**

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>I8</td>
<td>output</td>
<td>Refer to the following status codes.</td>
</tr>
<tr>
<td>QueueID</td>
<td>U8</td>
<td>input</td>
<td>Mailbox queue, [0, 9]</td>
</tr>
<tr>
<td>Remove (T)</td>
<td>Boolean</td>
<td>input</td>
<td>If TRUE, remove message from specified queue after reading data</td>
</tr>
<tr>
<td>Message</td>
<td>string</td>
<td>output</td>
<td>Message data</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>0x40 STAT_MSG_EMPTY_MAILBOX</td>
<td>Specified mailbox queue is empty</td>
</tr>
<tr>
<td>-18</td>
<td>0xEE ERR_INVALID_QUEUE</td>
<td>Invalid mailbox queue specified</td>
</tr>
</tbody>
</table>

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

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>I8</td>
<td>output</td>
<td>Refer to the following status codes.</td>
</tr>
<tr>
<td>QueueID</td>
<td>U8</td>
<td>input</td>
<td>Mailbox queue, [0, 19]</td>
</tr>
<tr>
<td>Message</td>
<td>string</td>
<td>input</td>
<td>Message data</td>
</tr>
</tbody>
</table>

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 is 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:

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Enum</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>–18 0xEE</td>
<td>ERR_INVALID_QUEUE</td>
<td>Invalid mailbox queue specified</td>
</tr>
<tr>
<td>–19 0xED</td>
<td>ERR_INVALID_SIZE</td>
<td>Message too large</td>
</tr>
</tbody>
</table>

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, the device checks 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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Start Tick</td>
<td>U32</td>
<td>output</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sleep Time Limit</td>
<td>U32</td>
<td>output</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Number</td>
<td>I16</td>
<td>output</td>
</tr>
</tbody>
</table>

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

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>I8</td>
<td>output</td>
<td>Unused</td>
</tr>
<tr>
<td>ModuleName</td>
<td>string</td>
<td>input</td>
<td>Name of the firmware module</td>
</tr>
<tr>
<td>Offset</td>
<td>U16</td>
<td>input</td>
<td>Offset from beginning of the firmware module I/O map</td>
</tr>
<tr>
<td>Count</td>
<td>U16</td>
<td>input</td>
<td>Number of bytes to read</td>
</tr>
<tr>
<td>Buffer</td>
<td>U8 array</td>
<td>output</td>
<td>I/O map data</td>
</tr>
</tbody>
</table>

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

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Code</td>
<td>I8</td>
<td>output</td>
<td>Unused</td>
</tr>
<tr>
<td>ModuleName</td>
<td>string</td>
<td>input</td>
<td>Name of the firmware module</td>
</tr>
<tr>
<td>Offset</td>
<td>U16</td>
<td>input</td>
<td>Offset from beginning of the firmware module I/O map</td>
</tr>
<tr>
<td>Buffer</td>
<td>U8 array</td>
<td>input</td>
<td>I/O map data</td>
</tr>
</tbody>
</table>

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

Return Value:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Code</th>
<th>I/O Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SleepTimeout</td>
<td>U32</td>
<td>input</td>
<td>Sleep timeout value in milliseconds</td>
</tr>
</tbody>
</table>

This dynamic system call method used by NXT datalogging sets the brick sleep timeout during datalogging programs. This prevents the brick from going to sleep during datalogging.
Using Dynamic System Calls

Dynamic system calls are custom functions that you create in the NXT firmware.

Note When you create a dynamic system call, ensure the ID of the system call is unique.

You can create system-level methods that access dynamic system calls from an NXT VI. You then can use the NXTDynamicSyscall refnum to expose these methods.

You must update the NXT firmware to support the dynamic system call and build a new firmware image before you can access the system call from an NXT VI.

Creating Access to a Dynamic System Call

Complete the following steps to create access to a dynamic system call.

1. Create a .rc file that defines the system-level method with which you want to access the dynamic system call.
2. Save the .rc file in the labview\resource\objmgr directory. The name of the .rc file must begin with NXT.
3. Create a VI that outputs the data type and default return value of the dynamic system call as a LabVIEW variant data type. The module compiler uses this VI to determine the data to send to the dynamic system call in the firmware.
4. Select the 4 × 2 × 2 × 4 connector pane pattern for the VI. Refer to the LabVIEW Help, available by selecting Help>Search the LabVIEW Help, for information about building connector panes.
5. Assign the top-right terminal of the connector pane to the variant output, as shown at left.
6. Save the VI in the labview\vi.lib\addons\NXTModule\DynamicSystemCalls directory. The name of the VI must be the same as the name of the system-level method you created.
Using the NXTDynamicSyscall Refnum

Complete the following steps to use the NXTDynamicSyscall refnum to access a dynamic system call you created.

1. Create a new NXT VI.
2. Place the NXTDynamicSyscall refnum on the front panel.
3. Press the <Ctrl-E> keys or select Window»Show Block Diagram to display the block diagram.
4. Place an Invoke Node on the block diagram.
5. Wire the NXTDynamicSyscall refnum to the reference input of the Invoke Node.
6. Click the Method element of the Invoke Node and select the system-level method you created from the shortcut menu.
Technical Support and Professional Services

Visit the following sections of the award-winning National Instruments Web site at ni.com for technical support and professional services:

- **Support**—Technical support at [ni.com/support](http://ni.com/support) includes the following resources:
  - **Self-Help Technical Resources**—For answers and solutions, visit [ni.com/support](http://ni.com/support) for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on. Registered users also receive access to the NI Discussion Forums at [ni.com/forums](http://ni.com/forums). NI Applications Engineers make sure every question submitted online receives an answer.
  - **Standard Service Program Membership**—This program entitles members to direct access to NI Applications Engineers via phone and email for one-to-one technical support as well as exclusive access to on demand training modules via the Services Resource Center. NI offers complementary membership for a full year after purchase, after which you may renew to continue your benefits.
    
    For information about other technical support options in your area, visit [ni.com/services](http://ni.com/services), or contact your local office at [ni.com/contact](http://ni.com/contact).

- **Training and Certification**—Visit [ni.com/training](http://ni.com/training) for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.

- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit [ni.com/alliance](http://ni.com/alliance).
If you searched ni.com and could not find the answers you need, contact your local office or NI corporate headquarters. Phone numbers for our worldwide offices are listed at the front of this manual. You also can visit the Worldwide Offices section of ni.com/niglobal to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.