# NeXT Byte Codes Manual

Explanations and Examples of
NeXT Byte Codes (NBC)

By

John Hansen & Michael Andersen

# 1. What is NBC

- **NeXT Byte Codes**
  - A simple language used to write programs for the standard NXT firmware
    - **Define – aggregate types (structs)**
    - **Declare – scalars, arrays, and aggregates**
      - Scalars can be byte, signed byte, word, signed word, dword, and signed dword types
      - Strings are supported in the form of arrays of byte
    - **Initialize – provide non-zero default values**
    - **Execute – use data as arguments for NXT executable code**

# 2. NBC Sample

```
// MySimpleProg.nbc
#include "NXTDefs.h"
dseg segment

; Define
MyStructureDef struct
 result sbyte
 freq word
 time word
 loop byte
 vol byte
MyStructureDef ends

; Declare
  theStruct MyStructureDef
  var1 byte TRUE
  strMsg byte[] 'Testing' ; Initialize
  aData dword[] 0x10, 0x20, 0x30, 0x40, 0x50
dseg ends

thread main
; Execute
Loop:
  /* loop body goes here */
  brtst EQ, Loop, var1 ; if false loop
theEnd:
  exit
endt
```

# 3. Sourcecode

- **Programs have zero or more data segments and a single code segment**
  - If you aren't in a data segment you are in the code segment
  - The code segment contains one or more threads which comprise the executable program logic
- **Data segments contain all type definitions and variable declarations**
  - You can have multiple data segments
  - All variables are global regardless of where they are declared
  - **Example:**

```
dseg segment
// type definitions and variable declarations go here
dseg ends
thread main
  dseg segment
    // or here
  dseg ends
endt
```

# 4. Preprocessing

- **Use standard preprocessor features such as #include, #define,** #ifdef, #else, #endif, etc…
- **NBC programs typically begin with #include "NXTDefs.h"**
  - This standard header file includes many important constants and macros that form the core NBC API (application programming interface).
- **Use #define to define your own constants for use throughout the program**
  - #define TurnTime 3000 ; 3 seconds
- **Define powerful multi-line function macros which can replace common chunks of program code**

```
#define MyMacro(x, y, berase, msg) \
mov dtArgs.Location.X, x \
mov dtArgs.Location.Y, y \
mov dtArgs.Options, berase \
mov dtArgs.Text, msg \
syscall DrawText, dtArgs
```

```
MyMacro(0, 0, TRUE, 'testing')
MyMacro(10, 20, FALSE, 'Please Work')
```

# 5. Parsing

- **Comments** (EOL = end of line)
  - ; text to EOL following a semicolon is a comment
  - // text to EOL following double forward slashes is a comment

  - /* everything contained within c-style block comment characters is a comment */
- **Case**
  - Compiler is case-sensitive; y is not the same identifier as Y
- **Strings**
  - Delimited by single quote characters. Embedded single quote characters are not supported in the initial NBC release.
- **Whitespace**
  - Ignored outside of string initializations and constant expressions
    - Whitespace in a constant expression will result in a compiler error
  - Required to separate labels, opcodes, and arguments
  - Required to separate variable names, variable types, and initial values
- **Multiple arguments are separated by commas**

# 6. Compiler Tokens

- **NBC supports special tokens which it replaces on compilation.**
  - Similar to preprocessor #defines but handled directly by the compiler.
- **Supported tokens**
  - __FILE__: Replaced with the currently active filename (no path)
  - __LINE__: Replaced with the current line number
  - __VER__: Replaced with the compiler version number
  - __THREADNAME__: Replaced with the current thread name
  - __I__, __J__: Replaced with the current value of I or J. These are both initialized to zero at the start of each thread or subroutine.
  - __ResetI__, __ResetJ__: Replaced with nothing. As a side effect the value of I or J is reset to zero.
  - __IncI__, __IncJ__: Replaced with nothing. As a side effect the value of I or J is incremented by one.
  - __DecI__, __DecJ__: Replaced with nothing. As a side effect the value of I or J is decremented by one.
  - ##: Replaced with nothing. This token allows you to make the use of other compiler tokens more readable.
    - __THREADNAME__##_##__I__: would become something like main_1:

# 7. Type Definitions

- **Contained in data segments**
  - **Used to define type aliases or aggregate types (struct)**
    - A structure definition
      - Begins with a type name followed by the keyword "struct"
      - Ends with the same type name followed by the keyword "ends"
  - **Example:**

```
MyPoint struct
```

```
  x byte
  y byte
MyPoint ends
big typedef dword     ; big is now an alias for the native dword type
ComplexStrut struct
  value1 big     // using a type alias
  value2 sdword
  buffer byte[]     /* array of byte */
  blen word
  extrastuff MyPoint[]     // array of structs
  pt_1 MyPoint     // struct contains struct instances
  pt_2 MyPoint
ComplexStruct ends
```

## 8. Type Declarations

- **Contained in data segments**
  - **Used to declare variables for use in the code segment**

```
x byte     // x = 0
y byte 12     // y = 12
z byte[] 'Testing'
// z is an array of byte = (0x54, 0x65, 0x73, 0x74, 0x69, 0x6e, 0x67, 0x00)
myBig big 0x12345     // use a type alias
myStruct ComplexStruct     // declare an instance of a struct
```

  - **Primary scalar types**
    - byte, sbyte, ubyte – unsigned and signed byte types (8 bits)
    - word, sword, uword – unsigned and signed word types (16 bits)
    - dword, sdword, udword – unsigned and signed double word types (32 bits)
      - long, slong, and ulong are aliases for these double word types
  - **Other scalar types**
    - void – supported by firmware and compiler but not used in executable code
    - mutex - used for calling a subroutine with exclusive access
  - **Arrays**
    - Declared using type name (scalar or aggregate) followed by []
    - Declare arrays of arrays by adding additional [] pairs

## 9. Executable Code

- **Contained in the code segment of an NBC program**
  - Tells the NXT what to do and when to do it
  - Consists of threads or subroutines containing operations (opcodes) and arguments which are assembled into a series of word (2 byte) values.
- **Basic unit of program flow is called a thread**
  - All programs must have at least one thread

- All threads must be finalized correctly
- A special type of thread is a subroutine
- **Example:**

```
thread main
/* body of main thread goes here */
  call mySub // compiler handles the subroutine return address
  exit // finalize execution (details are handled by the compiler)
endt

subroutine mySub
/* body of subroutine goes here */
  return // compiler handles the subroutine return address
ends
```

# 10. Threads

- **A thread may have dependant threads and subroutines which it uses to complete the logic of the entire program**
  - Thread dependants must be listed by name somewhere within the body of a thread using either the “precedes” keyword or the “follows” keyword.
  - Use "thread <name>" to begin a thread and "endt" to end it.
  - If the "exit" opcode is missing the compiler handles scheduling dependants.

```
thread main
  precedes waiter, worker
  /* thread body goes here */
  // finalize this thread and schedule the threads in the
  // specified range to execute
  exit // all dependants are automatically scheduled
endt

thread waiter
  /* thread body goes here */
//  exit ; exit is optional due to smart compiler finalization
endt

thread worker
  precedes waiter
  /* thread body goes here */
  exit // only one dependent – schedule it to execute
endt
```

# 11. Subroutines

- **A subroutine is a type of thread that is designed to be called explicitly by other threads or subroutines.**
  - You can pass arguments into and out of subroutines using global variables.

- If a subroutine is designed to be used by concurrently executing threads then calls to the subroutine must be protected by acquiring a mutex.
- Use "subroutine <name>" to begin a subroutine and "ends" to end it.
- To return to the calling thread a subroutine uses the subret or return opcodes.

```
thread main
  /* thread body goes here */
  acquire ssMutex
  call SharedSub ; automatic return address (must be a "subroutine")
  release ssMutex
  subcall AnotherSub, anothersub_returnaddress ; manual return address
  // exit ; exit is optional due to smart compiler finalization
endt

subroutine SharedSub
  /* subroutine body goes here */
  return ; return or subret is required as the last operation in a subroutine
ends

thread AnotherSub
  /* threads can be subroutines too */
  subret anothersub_returnaddress ; manually handling the return address
endt
```

# 12. Opcode Overview

- **Families**
  - Memory, Math, Logic, Bit Manipulation, Comparison, Control Flow, Timing, Arrays, Strings, Scheduling, and IO
- **Argument Types**
  - Valid argument types include variables, constants, IO Map addresses (IOMA), labels, and thread names
  - Output (dest) arguments must be a variable or an IOMA. Input (src) arguments can be a variable, an IOMA, or a constant expression
    - Variables can be scalar, array, or struct types
    - Use dot notation to reference struct members (x.y.z)
    - The compiler introduces a temporary variable for each constant expression used where a variable or IOMA is expected (only for input arguments)
  - IOMA are special constants which provide direct access to input and output field values

# 13. Memory

- **All opcodes in this family have one output argument and one input argument**

**mov**

```
      mov x, y     // set x equal to y
```

**set**

```
set x, 10     // set x equal to 10
```

- **mov arguments must be compatible (struct to struct, array to array, scalar to scalar)**
  - Scalar types do not have to match
- **set requires that its output argument be a scalar variable or an IOMA. Its input argument must be a constant or constant expression.**
  - Input argument is limited to a 16 bit signed or unsigned value

# 14. Math

- **All opcodes in this family have one output argument and two input arguments except negate, absolute value, and sign**
  - Can use scalar, array, and aggregate argument types

**add**

```
set incr, 1
add x, x, incr     // increment x by 1
```

**sub**

```
set decr, 1
sub x, x, decr     // decrement x by 1
```

**mul**

```
mul x, x, x     // x = sqr(x)
```

**div**

```
div x, x, 2     // x = x / 2 (integer division)
// (a temporary will be introduced by the compiler
// in this case)
```

**mod**

```
mod x, x, 4     // x = x % 4 (remainder)
// (set x to 0..3 – uses a temporary in this case)
```

**neg**

```
neg x, x     // x = -x (unary operator)
```

**abs**

```
abs x, y     // x = abs(y) (absolute value)
```

**sign**

```
sign x, y     // x = sign(y) (-1, 0, 1)
```

# 15. Logic

- **All opcodes in this family have one output argument and two input arguments except logical not**

- Can use scalar, array, and aggregate argument types

**and**

```
and x, x, y     // x = x & y
// (if x = b1010 and y = b0110 then x & y = 0010)
```

**or**

```
or x, x, y   // x = x | y
// (if x = b1010 and y = b0110 then x | y = 1110)
```

**xor**

```
xor x, x, y     // x = x ^ y
// (if x = b1010 and y = b0110 then x ^ y = 1100)
```

**not**

```
not x, x     // x = !x
// the firmware implements not as a logical
// operation – not a bitwise operation
```

# 16. Bit Manipulation

- **All opcodes in this family have one output argument and two input arguments**

**shr**

```
//shr result, value, bits
shr myVal, myVal, 3 // divide myVal by 2 three times
```

**shl**

```
//shl result, value, bits
shl myVal, myVal, 2 // multiply myVal by 2 twice
```

- **These opcodes are implemented by the compiler since the standard NXT firmware does not support shift operations at this time.**

# 17. Comparison

- **These opcodes all take a comparison code constant as their first argument**
  - Can use scalar, array, and aggregate types for the compare or test argument(s)
  - Valid comparison codes
    - Less than: LT = 0x00 ('<' may also be used)
    - Greater than: GT = 0x01 ('>' may also be used)
    - Less than or equal to: LTEQ = 0x02 ('<=' may also be used)
    - Greater than or equal to: GTEQ = 0x03 ('>=' may also be used)
    - Equal to: EQ = 0x04 ('==' may also be used)
    - Not equal to: NEQ = 0x05 ('!=' or '<>' may also be used)

**cmp cc, dest, src1, src2**

```
cmp EQ, bEqual, x, y  // set bEqual to true if x == y
```

**tst cc, dest, src**

```
tst GT, bGTZero, x  // set bGTZero to true if x > 0
```

# 18. Control Flow

- **The jump opcode takes a label as its only argument and unconditionally jumps to that program location**

**jmp label**

```
LoopStart:
// loop body
jmp LoopStart  // jump to LoopStart label
```

- **The branch opcodes take a comparison code constant as their first argument (see Comparison) and a label for the second argument**

**brcmp cc, label, src1, src2**

```
brcmp EQ, EqualValues, x, y  // jump to EqualValues label if x == y
```

**brtst cc, label, src**

```
brtst GT, XGtZero, x  // jump to XGtZero label if x > 0
```

- **The stop opcode stops program execution depending on the value of its boolean input argument**

**stop stop?**

```
stop bStop  // stop the program if bStop is true (non-zero)
```

# 19. System Calls

- **The syscall opcode enables execution of various system functions via a constant function ID and an aggregate type variable for passing arguments to and from the system function**

**syscall functionID, args**

```
syscall SoundPlayTone, ptArgs
// ptArgs is a struct with input and output args
```

  - **Function identifiers:**

```
FileOpenRead (0), FileOpenWrite (1), FileOpenAppend (2), FileRead
(3), FileWrite (4), FileClose (5), FileResolveHandle (6), FileRename
(7), FileDelete (8), SoundPlayFile (9), SoundPlayTone (10),
SoundGetState (11), SoundSetState (12), DrawText (13), DrawPoint
(14), DrawLine (15), DrawCircle (16), DrawRect (17), DrawGraphic
(18), SetScreenMode (19), ReadButton (20), CommLSWrite (21),
CommLSRead (22), CommLSCheckStatus (23), RandomNumber (24),
GetStartTick (25), MessageWrite (26), MessageRead (27),
```

```
CommBTCheckStatus (28), CommBTWrite (29), KeepAlive (31), IOMapRead
(32), IOMapWrite (33)
```

# 20. Timing

- **The wait opcode suspends the current thread for the number of milliseconds specified by its constant argument**

```
wait 1000  // wait for 1 second
```

- **The waitv opcode acts like wait but it takes a variable argument**

```
waitv iDelay  // wait for the specified number of ms.
```

- **The gettick opcode sets its output argument to the current system tick count**

```
gettick x  // set x to the current system tick count
```

  - NBC implements wait and waitv as thread-specific subroutine calls due to the wait opcode not having been implemented in the NXT firmware
  - If you use a constant argument with waitv the compiler will generate a temporary variable for you
  - If needed, you can implement simple wait loops using gettick:

```
gettick currTick
add endTick, currTick, waitms
Loop:
  gettick currTick
  brcmp LT, Loop, currTick, endTick
```

# 21. Arrays

- **The index opcode returns the arraysrc[index] value in the dest output argument**

**index dest, arraysrc, index**

```
index val, arValues, idx
// extract arValues[idx] and store it in val
```

- **The replace opcode replaces the arraysrc[index] item in arraydest with the newval input argument (arraysrc can be the same variable as arraydest to replace without copying the array; newval can be an array, in which case multiple items are replaced)**

**replace arraydest, arraysrc, index, newval**

```
replace arNew, arValues, idx, x
// replace arValues[idx] with x in arNew
```

- **The arrsize opcode returns the size of the input array in the scalar dest output argument**

**arrsize dest, arraysrc**

```
arrsize nSize, arValues  // nSize == length of array
```

- **The arrinit opcode initializes the output array to the value and size provided**

  **arrinit arraydest, value, size**

```
arrinit arValues, nZero, nSize
// initialize arValues with nSize zeros
```

- **The arrsubset opcode copies a subset of the input array to the output array**

  **arrsubset arraydest, arraysrc, index, length**

```
arrsubset arSub, arValues, NA, x
// copy the first x elements to arSub
```

- **The arrbuild opcode constructs an output array from a variable number of input arrays, scalars, or aggregates**

  **arrbuild arraydest, src1, src2, …, srcN**

```
arrbuild arData, arStart, arBody, arEnd
// build data array from 3 sources
```

# 22. Strings

- **The flatten opcode converts its input argument into its string output argument**

  **flatten strdest, source**

```
flatten strData, args  // copy from the args struct to strData
```

- **The unflatten opcode converts its input string argument to the output argument type. If 'default' does not match the flattened data type exactly (including array sizes) then 'err' will be set to TRUE and 'dest' will contain a copy of 'default'**

  **unflatten dest, err, strsrc, default**

```
unflatten args, bErr, strSource, x  // convert string to cluster
```

- **The numtostr opcode converts its scalar input argument to a string output argument**

  **numtostr deststr, source**

```
numtostr strValue, value  // convert value to a string
```

- **The strtonum opcode parses its input string argument into a numeric output argument, advancing an offset past the numeric string**

  **strtonum dest, offsetNext, str, offsetCur, default**

```
strtonum value, idx, strValue, idx, nZero  // parse string into num
```

- **The strsubset opcode copies a subset of the input string to the output string**

  **strsubset strdest, strsrc, index, length**

```
strsubset strSub, strSource, NA, x
// copy the first x bytes in strSource to strSub
```

- **The strcat opcode constructs an output string from a variable number of input strings**

  **strcat strdest, strsrc1, strsrc2, …, strsrcN**

```
strcat strData, strStart, strBody, strEnd
// build data string from 3 sources
```

- **The arrtostr opcode copies the input byte array into its output string array and adds a null byte at the end**

  **arrtostr strdest, arraysrc**

```
arrtostr strData, arrData  // convert byte array to string
```

- **The strtoarr opcode copies the input string array into its output byte array excluding the last byte (which should be a null)**

  **strtoarr arraydest, strsrc**

```
strtoarr arrData, strData  // convert string to byte array
```

- **The strindex opcode returns the strsrc[index] value in the dest output argument**

  **strindex dest, strsrc, index**

```
strindex val, strVal, idx
// extract strVal[idx] and store it in val
```

- **The strreplace opcode replaces the strsrc[index] item in strdest with the newval input argument (strsrc can be the same variable as strdest to replace without copying the string; newval can be a string, in which case multiple items are replaced)**

  **strreplace strdest, strsrc, index, newval**

```
strreplace strNew, strValues, idx, newStr
// replace strValues[idx] with newStr in strNew
```

- **The strlen opcode returns the size of the input string in the scalar output argument**

  **strlen dest, strsrc**

```
strlen nSize, strMsg  // nSize == length of strMsg
```

# 23. Scheduling

- **The exit opcode finalizes the current thread and schedules zero or more dependant threads by specifying start and end dependency list indices (indices are zero-based and inclusive)**
  **exit [start, end]**

```
exit 0, 2  // schedule this thread's 3 dependants
exit // schedule all this thread's dependants
```

- **The exitto opcode finalizes the current thread and schedules the thread specified by name**

```
exitto worker  // exit now and schedule worker thread
```

- **The acquire opcode acquires the named mutex. If the mutex is already acquired the current thread waits until it becomes available**

```
acquire muFoo  // acquire mutex for subroutine
```

- **The release opcode releases the named mutex allowing other threads to acquire it**

```
release muFoo  // release mutex for subroutine
```

- **The subcall opcode calls into the named thread/subroutine and waits for a return (which might not come from the same thread)**
  - **The second argument is a variable used to store the return address**

```
subcall drawText, retDrawText  // call drawText subroutine
```

- **The subret opcode returns from a thread to the return address value contained in its input argument**

```
subret retDrawText  // return to calling routine
```

- **The call opcode executes the named subroutine and waits for a return. The argument should specify a thread that was declared using the "subroutine" keyword.**

```
call MyFavoriteSubroutine  // call routine
```

- **The return opcode returns from a subroutine.**

```
return  // return to calling routine
```

- **The compiler automatically handles the return address for call and return when they are used with subroutines rather than threads**

# 24. Inputs

- **The setin opcode sets an input field of a sensor on a port to the value specified in its first input argument**

**setin src, port, fieldID**

```
set theType, IN_TYPE_SWITCH  // set type to switch
set theMode, IN_MODE_BOOLEAN  // set mode to boolean
set thePort, IN_1  // set port to #1
setin theType, thePort, Type  // set sensor on port 1 to switch type
setin theMode, thePort, InputMode  // set sensor on port 1 to
boolean mode
```

  - **Input Field Identifiers**

```
Type (0), InputMode (1), RawValue (2), NormalizedValue (3),
ScaledValue (4), InvalidData (5)
```

- **The getin opcode reads a value from an input field of a sensor on a port and writes the value to its dest output argument**

**getin dest, port, fieldID**

```
getin rVal, thePort, RawValue  // read raw sensor value
getin sVal, thePort, ScaledValue  // read scaled sensor value
getin nVal, thePort, NormalizedValue  // read normalized sensor value
```

# 25. Outputs

- **The setout opcode sets one or more output fields of a motor on one or more ports to the value specified by the coupled input arguments**
- **setout port/portlist, fieldid1, value1, …, fieldidN, valueN**

```
set theMode, OUT_MODE_MOTORON  // set mode to motor on
set rsVal, OUT_RUNSTATE_RUNNING // motor running
set thePort, OUT_A  // set port to #1 (use an array for multiple ports)
set pwr, -75 // negative power means reverse motor direction
// set output values
setout thePort, OutputMode, theMode, RunState, rsVal, Power, pwr
```

  - **Output Field Identifiers**

```
UpdateFlags (0), OutputMode (1), Power (2), ActualSpeed (3), TachoCount
(4), TachoLimit (5), RunState (6), TurnRatio (7), RegMode (8), Overload
(9), RegPValue (10), RegIValue (11), RegDValue (12), BlockTachoCount
(13), RotationCount (14)
```

- **The getout opcode reads a value from an output field of a sensor on a port and writes the value to its dest output argument**

**getout dest, port, fieldID**

```
getout rmVal, thePort, RegMode  // read motor regulation mode
getout tlVal, thePort, TachoLimit  // read tachometer limit value
getout rcVal, thePort, RotationCount // read the rotation count
```

# 26. IO Map Addresses

- **IOMA constants provide a simplified means for accessing input and output field values**
  - **Instead of:**

```
set theType, IN_TYPE_SWITCH  // set type to switch
set theMode, IN_MODE_BOOLEAN  // set mode to boolean
set thePort, IN_1  // set port to #1 (0)
setin theType, thePort, Type  // set sensor on port 1 to switch type
setin theMode, thePort, InputMode  // set sensor on port 1 to
boolean mode
getin rVal, thePort, RawValue  // read raw sensor value
getin sVal, thePort, ScaledValue  // read scaled sensor value
getin nVal, thePort, NormalizedValue  // read normalized sensor
value
getout rmVal, thePort, RegMode // read output regulation mode
```

  - **You can use:**

```
set InputIOType0, IN_TYPE_SWITCH  // set type to switch
set InputIOInputMode0, IN_MODE_BOOLEAN  // set mode to boolean
mov rVal, InputIORawValue0  // read raw sensor value
mov sVal, InputIOScaledValue0 // read scaled value
mov nVal, InputIONormalizedValue0 // read normalized value
set OutputIORunState0, OUT_RUNSTATE_RUNNING // set motor A run state
```

```
set OutputIOPower0, -75 // set motor A power
mov rmVal, OutputIORegMode0 // read motor A regulation mode
mov rcVal, OutputIORotationCount0 // read motor A rotation count
```

# 27. Compile-time Operations

- **Compiler operations**

  **sizeof(arg) // returns the compile-time size of the specified argument**

```
dseg segment
  arg byte
  argsize byte
dseg ends
// ...
set argsize, sizeof(arg) ; argsize == 1
```

  **isconst(arg) // returns 0 or 1 depending on whether arg is a constant or not**

```
dseg segment
  arg byte
  argsize byte
dseg ends
// ...
set argsize, isconst(arg) ; argsize == 0
```

- **Compiler opcodes**

  **compchk cc, const1, const2**

```
compchk EQ, sizeof(arg3), 2
```

  – Reports a compiler error if the expression does not evaluate to true.

  **compif cc, const1, const2**
  **compelse**
  **compend**

  – Used to write conditional blocks of code in conjunction with compelse and compif.

# 28. Expression Evaluator

- **Constant expressions are supported for many opcode arguments**
  - **Expressions are evaluated at compile time (not at run time)**
  - **Expressions must not contain whitespace (4+4, not 4 + 4)**
  - **Supported operators:**
    - +, -, *, / , ^ (power), % (modulo), &, |, <<, and >>

- () for grouping subexpressions
- PI constant

– **Supported compile-time functions:**

- tan, sin, cos, sinh, cosh, arctan, cotan,
  arg, exp, ln, log10, log2, logn, sqr, sqrt,
  abs, trunc, int, ceil, floor, heav, sign, zero,
  ph, rnd, random, max, min, power, intpower

– **Example:**

```
set val, 3+(PI*2)-sqrt(30)
// expression value will be truncated to an integer
```

# 29. Snippets: Inputs

```
dseg segment
  theSensorType byte
  theSensorMode byte
  isInvalid byte
  thePort byte
  SVal sword
  RVal word
  NVal word
dseg ends

thread main
set thePort, IN_1
// set sensor type and mode
set   theSensorType, IN_TYPE_SWITCH
set   theSensorMode, IN_MODE_BOOLEAN
setin theSensorType, thePort, Type
setin theSensorMode, thePort, InputMode

/* when changing the type and/or mode of a sensor the NXT blocks also set the
InvalidData field to true */

set isInvalid, TRUE
setin isInvalid, thePort, InvalidData
// make sure the sensor value is valid
stillInvalid:
      getin isInvalid, thePort, InvalidData
      brtst NEQ, stillInvalid, isInvalid
/* if isInvalid not false (equal to zero) then loop until it is */

/* set sensor values (the NXT blocks reset the ScaledValue to zero before
reading it shortly afterward) */

set   SVal, 0
setin SVal, thePort, ScaledValue

// read sensor values

getin SVal, thePort, ScaledValue
getin RVal, thePort, RawValue
getin NVal, thePort, NormalizedValue
```

```
exit
endt
```

## 30. Snippets: Inputs IOMA

```
dseg segment
  SVal sword
  RVal word
  NVal word
dseg ends

thread main
// set sensor type and mode
set   InputIOType0, IN_TYPE_SWITCH

// reading raw sensor values this time
set   InputIOInputMode0, IN_MODE_RAW

/* when changing the type and/or mode of a sensor the NXT blocks also set the
InvalidData field to true */

set InputIOInvalidData0, TRUE
// make sure the sensor value is valid
stillInvalid:
brtst NEQ, stillInvalid, InputIOInvalidData0
/* if Invalidata is not false (equal to zero) then loop until it is */

/* set sensor values (the NXT blocks reset the ScaledValue to zero before
reading it shortly afterward) */

set InputIOScaledValue0, 0

// read sensor values

mov SVal, InputIOScaledValue0
mov RVal, InputIORawValue0
mov NVal, InputIONormalizedValue0

exit
endt
```

## 31. Snippets: Outputs

```
dseg segment
  theUF byte
  thePower sbyte
```

```
  theOM byte OUT_MODE_MOTORON+OUT_MODE_BRAKE
  theRM byte OUT_REGMODE_IDLE
  theRS byte OUT_RUNSTATE_RUNNING
dseg ends

thread main
      set theUF, UF_UPDATE_MODE ; initialize motor
      setout OUT_A, OutputMode, theOM, RegMode, theRM, RunState, theRS,
UpdateFlags, theUF

Forever:
      set thePower, 90

UpdateSpeed:
      set theUF, UF_UPDATE_SPEED+UF_UPDATE_MODE
      setout OUT_A, Power, thePower, OutputMode, theOM, UpdateFlags, theUF

      jmp Forever
      exit
endt
```

## 32. Snippets: Outputs IOMA

```
thread main
      set OutputIOOutputMode0, OUT_MODE_MOTORON+OUT_MODE_BRAKE
      set OutputIORegMode0, OUT_REGMODE_IDLE
      set OutputIORunState0, OUT_RUNSTATE_RUNNING
      set OutputIOUpdateFlags0, UF_UPDATE_MODE ; initialize motor

Forever:

      set OutputIOPower0, 90
      set OutputIOOutputMode0, OUT_MODE_MOTORON+OUT_MODE_BRAKE
      set OutputIOUpdateFlags0, UF_UPDATE_SPEED+UF_UPDATE_MODE

      jmp Forever
      exit
endt
```

## 33. SysCall: FileOpenRead

```
; define structure
TFileOpen   struct
 Result     word
 FileHandle byte
 Filename   byte[] 'test.txt'
 Length     dword
TFileOpen   ends

; declare variables
```

```
FOR_A TFileOpen
fh  byte

; call the API
syscall FileOpenRead, FOR_A

; copy the filehandle for later use
mov fh, FOR_A.FileHandle
```

## 34. SysCall: FileOpenWrite

```
; define structure
TFileOpen   struct
 Result     word
 FileHandle byte
 Filename   byte[] 'test.txt'
 Length     dword
TFileOpen   ends

; declare variables
FOW_A TFileOpen
fh byte

; call the API
syscall FileOpenWrite, FOW_A

; copy the filehandle for later use
mov fh, FOW_A.FileHandle
```

## 35. SysCall: FileOpenAppend

```
; define structure
TFileOpen   struct
 Result     word
 FileHandle byte
 Filename   byte[] 'test.txt'
 Length     dword
TFileOpen   ends

; declare variables
FOA_A TFileOpen
fh  byte

; call the API
syscall FileOpenAppend, FOA_A

; copy the filehandle for later use
mov fh, FOA_A.FileHandle
```

## 36. SysCall: FileRead

```
dseg segment
 ; define structure
 TFileReadWrite   struct
  Result    word
  FileHandle      byte
  Buffer    byte[]
  Length    dword
 TFileReadWrite   ends

 ; declare variables
 FR_A TFileReadWrite
 fh  byte
 tmpStr byte[]
 bytesRead dword
dseg ends

thread main
// ...
mov FR_A.FileHandle, fh ; file handle obtained via FileOpenRead
set FR_A.Length, 10 ; specify buffer size and desired number of bytes to read

syscall FileRead, FR_A ; call the API

mov tmpStr, FR_A.Buffer ; copy the Buffer for later use
mov bytesRead, FR_A.Length ; store the actual # of bytes read

endt
```

## 37. SysCall: FileWrite

```
dseg segment
 ; define structure
 TFileReadWrite   struct
  Result    word
  FileHandle      byte
  Buffer    byte[]
  Length    dword
 TFileReadWrite   ends

 ; declare variables
 FW_A TFileReadWrite
 fh  byte
 tmpStr byte[] 'testing'
 bytesWritten dword
dseg ends

thread main
// ...
mov FW_A.FileHandle, fh ; file handle obtained via FileOpenWrite
mov FW_A.Buffer, tmpStr ; copy data into write Buffer
```

```
syscall FileWrite, FW_A ; call the API
mov bytesWritten, FW_A.Length ; store the number of bytes written

endt
```

## 38. SysCall: FileClose

```
dseg segment
 ; define structure
TFileClose  struct
 Result     word
 FileHandle byte
TFileClose  ends

 ; declare variables
 FC_A TFileClose
 fh  byte
dseg ends

thread main
// ...
mov FC_A.FileHandle, fh ; file handle obtained via FileOpen*

syscall FileClose, FC_A ; call the API

endt
```

## 39. SysCall: FileResolveHandle

```
dseg segment
 ; define structure
TFileResolveHandle      struct
 Result     word
 FileHandle byte
 WriteHandle      byte
 Filename   byte[]
TFileResolveHandle      ends

 ; declare variables
 FRH_A TFileResolveHandle
 fh  byte
 bWriteable byte
 theFilename byte[] 'myfile.txt'
dseg ends

thread main
// ...
mov FRH.Filename, theFilename ; pass in a filename

syscall FileResolveHandle, FRH_A ; call the API
```

```
mov fh, FRH_A.FileHandle ; store the resolved handle
mov bWriteable, FRH_A.WriteHandle ; is this a writeable handle?

endt
```

## 40. SysCall: FileRename

```
dseg segment
 ; define structure
TFileRename struct
 Result     word
 OldFilename      byte[]
 NewFilename      byte[]
TFileRename ends

 ; declare variables
 FR_A TFileClose
 fh  byte
dseg ends

thread main
// ...
mov FR_A.OldFilename, 'oldfile.txt' ; old filename
mov FR_A.NewFilename, 'newfile.txt' ; new filename

syscall FileRename, FR_A ; call the API

endt
```

## 41. SysCall: FileDelete

```
dseg segment
 ; define structure
TFileDelete struct
 Result     word
 Filename   byte[]
TFileDelete ends

 ; declare variables
 FD_A TFileClose
 fh  byte
 theFilename byte[] 'delete_me.txt'
dseg ends

thread main
// ...
mov FD_A.Filename, theFilename ; filename to be deleted

syscall FileDelete, FD_A ; call the API
```

```
endt
```

## 42. SysCall: SoundPlayFile

- **Define the record**
- **Call the API**

```
dseg segment
; definition
SoundPF_def struct
 result sbyte
 filename byte[]
 loop byte
 vol byte
SoundPF_def ends

; declarations
PT_A SoundPT_def
 soundFile byte[] 'mysound.rso'
dseg ends

; code
  set PF_A.loop, 0    ; no looping
  set PF_A.vol, 3     ; percent/25
  mov PF_A.filename, soundFile
  syscall SoundPlayFile, PF_A
```

- **Wait for completion**
  - **NXT has no sound queue**

```
dseg segment
; definition
SoundGS_def struct
  result byte
  flags byte
SoundGS_def ends

; declaration
SGS SoundGS_def
dseg ends

; code
stillplaying:
  syscall SoundGetState, SGS
  brtst NEQ, stillplaying, SGS.flags
```

## 43. SysCall: SoundPlayTone

- **Define the record**
- **Call the API**

```
; definition
SoundPT_def struct
```

```
 result sbyte
 freq word
 time word
 loop byte
 vol byte
SoundPT_def ends

; declarations
PT_A SoundPT_def

; code
  set PT_A.freq, 440  ; A
  set PT_A.time, 1000 ; 1 sec
  set PT_A.loop, 0    ; no looping
  set PT_A.vol, 3     ; percent/25
  syscall SoundPlayTone, PT_A
```

- **Wait for completion**
  - **NXT has no sound queue**

```
; definition
SoundGS_def struct
  result byte
  flags byte
SoundGS_def ends

; declaration
SGS SoundGS_def

; code
stillplaying:
  syscall SoundGetState, SGS
  brtst NEQ, stillplaying, SGS.flags
```

## 44. SysCall: SoundGetState

- **Define the record**
- **Call the API**

```
; definition
SoundGS_def struct
  state byte
  flags byte
SoundGS_def ends
; declaration
SGS SoundGS_def

; code
stillplaying:
  syscall SoundGetState, SGS
  brtst NEQ, stillplaying, SGS.flags
```

## 45. SysCall: SoundSetState

- **Define the record**
- **Call the API**

```
dseg segment
; definition
TSoundSetState    struct
 Result     byte
 State      byte
 Flags      byte
TSoundSetState    ends

; declaration
SSS_A TSoundSetState
dseg ends

; code
set SSS.State, 0x04 ; stop playing
set SSS.Flags, 0x01 ; make changes take effect

syscall SoundSetState, SSS ; call the API
```

## 46. SysCall: DrawText

```
dseg segment
TLocation   struct
 X          sword
 Y          sword
TLocation   ends

TDrawText   struct
 Result     sbyte
 Location   TLocation
 Text byte[]
 Options    dword
TDrawText   ends

; declaration
dtArgs TDrawText
value dword
dseg ends

; code
set dtArgs.Location.X, 10
set dtArgs.Location.Y, 8
set dtArgs.Options, 1 ; erase previous text
set value, 13
numtostr dtArgs.Text, value ; set the text to draw

syscall DrawText, dtArgs ; call the API
```

## 47. SysCall: DrawPoint

```
dseg segment
TLocation   struct
 X          sword
 Y          sword
TLocation   ends

TDrawPoint  struct
 Result     sbyte
 Location   TLocation
 Options    dword
TDrawPoint  ends

; declaration
dpArgs TDrawPoint

dseg ends

; code
set dpArgs.Location.X, 10
set dpArgs.Location.Y, 8
set dpArgs.Options, 1 ; erase screen before drawing

syscall DrawPoint, dpArgs ; call the API
```

## 48. SysCall: DrawLine

```
dseg segment
TLocation   struct
 X          sword
 Y          sword
TLocation   ends

TDrawLine   struct
 Result     sbyte
 StartLoc   TLocation
 EndLoc     TLocation
 Options    dword
TDrawLine   ends

; declaration
dlArgs TDrawLine

dseg ends

; code
```

```
set dlArgs.StartLoc.X, 10
set dlArgs.StartLoc.Y, 8
set dlArgs.EndLoc.X, 50
set dlArgs.EndLoc.Y, 30
set dlArgs.Options, 1 ; erase screen before drawing

syscall DrawLine, dlArgs ; call the API
```

## 49. SysCall: DrawCircle

```
dseg segment
; definition
TLocation   struct
 X          sword
 Y          sword
TLocation   ends

TDrawCircle struct
 Result     sbyte
 Center     TLocation
 Size byte
 Options    dword
TDrawCircle ends

; declaration
dcArgs TDrawCircle

dseg ends

; code
set dcArgs.Center.X, 30
set dcArgs.Center.Y, 20
set dcArgs.Size, 10
set dcArgs.Options, 0 ; don't clear the screen

syscall DrawCircle, dcArgs ; call the API
```

## 50. SysCall: DrawRect

```
dseg segment
TLocation   struct
 X          sword
 Y          sword
TLocation   ends

TSize struct
 Width      sword
 Height     sword
```

```
TSize ends

TDrawRect   struct
 Result     sbyte
 Location   TLocation
 Size TSize
 Options    dword
TDrawRect   ends

drArgs TDrawRect
dseg ends

; code
set drArgs.Location.X, 30
set drArgs.Location.Y, 20
set drArgs.Size.Width, 20
set drArgs.Size.Height, 10
set drArgs.Options, 0 ; don't clear the screen

syscall DrawRect, drArgs ; call the API
```

## 51. SysCall: DrawGraphic

```
dseg segment
; definition
TLocation   struct
 X          sword
 Y          sword
TLocation   ends

TDrawGraphic      struct
 Result     sbyte
 Location   TLocation
 Filename   byte[]
 Variables  sdword[]
 Options    dword
TDrawGraphic      ends

; declaration
dgArgs TDrawGraphic
dseg ends

; code
set dgArgs.Location.X, 30
set dgArgs.Location.Y, 20
mov dgArgs.Filename, 'picture.ric' ; let compiler create temporary string
set dgArgs.Options, 0 ; do not clear screen

syscall DrawGraphic, dgArgs ; call the API
```

## 52. SysCall: SetScreenMode

- **Define the record**
- **Call the API**

```
TSetScreenMode struct
 Result byte
 ScreenMode   dword
TSetScreenMode ends

; declaration
SSM_A TSetScreenMode
dseg ends

; code

set SSM_A.ScreenMode, 0 ; erase the screen
syscall SetScreenMode, SSM_A ; call the API

/*
 calling SetScreenMode with any ScreenMode value other than zero has
 no effect. 0 == RESTORE_NXT_SCREEN
*/
```

## 53. SysCall: ReadButton

```
; declarations
TReadButton struct
 Result     sbyte
 Index      byte
 Pressed    byte
 Count      byte
 Reset      byte
TReadButton ends

  rbArgs TReadButton

  bPressed2 byte

; code

set rbArgs.Index, BTN3
syscall ReadButton, rbArgs ; call the API
mov bPressed2, rbArgs.Pressed

set rbArgs.Index, BTN3
set rbArgs.Reset, TRUE
syscall ReadButton, rbArgs ; reset the button
```

## 54. SysCall: CommLSWrite

```
; declarations
TCommLSWrite      struct
 Result     sbyte
 Port byte
 Buffer     byte[]
 ReturnLen  byte
TCommLSWrite      ends

 lswArgs TCommLSWrite
 bufLSWrite1 byte[] 0x2, 0x42

; code

// first configure sensor port as IN_TYPE_LOWSPEED_9V and IN_MODE_RAW
// (not shown)
mov   lswArgs.Port, thePort
set   lswArgs.ReturnLen, 1
mov   lswArgs.Buffer, bufLSWrite1
syscall CommLSWrite, lswArgs

// now you can check status and read ...
```

## 55. SysCall: CommLSRead

```
; declarations
TCommLSRead struct
 Result     sbyte
 Port byte
 Buffer     byte[]
 BufferLen  byte
TCommLSRead ends

 lsrArgs TCommLSRead
 readBuf byte[]
 lsBytesRead byte
 RLSBRetVal byte
; code

// first configure sensor port as IN_TYPE_LOWSPEED_9V and IN_MODE_RAW
// (not shown)
// then write to the LS sensor port using CommLSWrite

mov   lsrArgs.Port, thePort
set   lsrArgs.BufferLen, 1
syscall CommLSRead, lsrArgs

mov   readBuf, lsrArgs.Buffer
arrsize lsBytesRead, readBuf

index RLSBRetVal, readBuf, NA
```

## 56. SysCall: CommLSCheckStatus

```
; declarations
TCommLSCheckStatus      struct
 Result                 sbyte
 Port                   byte
 BytesReady             byte
TCommLSCheckStatus      ends

 lscsArgs TCommLSCheckStatus

; code

thread CheckLSStat
      mov   lscsArgs.Port, thePort
      syscall     CommLSCheckStatus, lscsArgs
      tst   LT, bLSStatLT0, lscsArgs.Result
      tst   EQ, bLSStatEQ0, lscsArgs.Result
      subret      chkls_ret
endt
```

## 57. SysCall: RandomNumber

```
; declarations
TRandomNumber     struct
 Result     sword
TRandomNumber     ends

 rnArgs TRandomNumber
 value sword

; code

      syscall     RandomNumber, rnArgs
      mov   value, rnArgs.Result
```

## 58. SysCall: GetStartTick

```
; declarations
TGetStartTick     struct
 Result     dword
TGetStartTick     ends

 gstArgs TGetStartTick
 curTick dword

; code

      syscall     GetStartTick, gstArgs
      mov   curTick, gstArgs.Result
```

## 59. SysCall: MessageWrite

```
; declarations
TMessageWrite     struct
 Result     sbyte
 QueueID    byte
 Message    byte[]
TMessageWrite     ends

 mwArgs TMessageWrite

; code

      syscall     MessageWrite, mwArgs
```

## 60. SysCall: MessageRead

```
; declarations
TMessageRead      struct
 Result     sbyte
 QueueID    byte
 Remove     byte
 Message    byte[]
TMessageRead      ends

 mrArgs TMessageRead

; code

      syscall     MessageRead, mrArgs
```

## 61. SysCall: CommBTCheckStatus

```
; declarations
TCommBTCheckStatus      struct
 Result           sbyte
 Connection byte
TCommBTCheckStatus      ends

 bcsArgs TCommBTCheckStatus

; code

      syscall     CommBTCheckStatus, bcsArgs
```

## 62. SysCall: CommBTWrite

```
; declarations
```

```
TCommBTWrite      struct
 Result     sbyte
 Connection byte
 Buffer     byte[]
TCommBTWrite      ends

 btwArgs TCommBTWrite

; code

      syscall     CommBTWrite, btwArgs
```

## 63. SysCall: KeepAlive

```
; declarations
TKeepAlive  struct
 Result     dword
TKeepAlive  ends

 kaArgs TKeepAlive
 val dword

; code

      syscall     KeepAlive, kaArgs
      mov   val, kaArgs.Result // current sleep time
```

## 64. SysCall: IOMapRead

```
TIOMapRead  struct
 Result     sbyte
 ModuleName byte[]
 Offset     word
 Count      word
 Buffer     byte[]
TIOMapRead  ends

iomrArgs TIOMapRead
blevel sdword
b1 byte
b2 byte

mov iomrArgs.ModuleName, 'Ui.mod'
set iomrArgs.Offset, 4 // UIOffsetBatteryLevel
set iomrArgs.Count, 2  // Battery level is 2 bytes
syscall IOMapRead, iomrArgs

// two bytes go into blevel
index b1, iomrArgs.Buffer, NA // 0
index b2, iomrArgs.Buffer, 1
mul blevel, b2, 256
add blevel, blevel, b1
```

## 65. SysCall: IOMapWrite

```
// Put the NXT into boot mode
dseg  segment
TIOMapWrite struct
 Result           sbyte
 ModuleName byte[]
 Offset           word
 Buffer           byte[]
TIOMapWrite ends

iomwArgs TIOMapWrite

dataBuf byte[] 0x5A, 0xA5 // boot

dseg  ends
; -------------- program code --------------
thread main

  mov iomwArgs.ModuleName, 'IOCtrl.mod'
  set iomwArgs.Offset, 0 ; IOCtrlOffsetPowerOn
  mov iomwArgs.Buffer, dataBuf
  syscall IOMapWrite, iomwArgs
  exit
endt
```

## 66. Predefined Constants

- **See NBCCommon.h for a complete list of predefined constants**
  - NA (0xFFFF)
    - Indicates a non-available argument
    - Can be used in certain operations to allow for a default behavior
      - stop – if the stop? argument is NA use true
      - index – if the index argument is NA use 0
      - replace – if the index argument is NA use 0
      - arrinit – if the size argument is NA use 0
      - arrsubset – if the index argument is NA use 0, if the length argument is NA use the length of the source array
      - strsubset – if the index argument is NA use 0, if the length argument is NA use the length of the source string
      - strtonum – if the offset argument is NA use 0, if the default argument is NA use 0
  - **Boolean Values**
    - TRUE (1), FALSE (0)
  - **Output Ports**
    - OUT_A (0x00), OUT_B (0x01), OUT_C (0x02)
  - **Input Ports**
    - IN_1 (0x00), IN_2 (0x01), IN_3 (0x02), IN_4 (0x03)

- **Comparison Codes**
  - LT (0x00), GT (0x01), LTEQ (0x02), GTEQ (0x03), EQ (0x04), NEQ (0x05)
- **Update Flags**
  - UF_UPDATE_MODE (0x01), UF_UPDATE_SPEED (0x02), UF_UPDATE_TACHO_LIMIT (0x04), UF_UPDATE_RESET_COUNT (0x08), UF_UPDATE_PID_VALUES (0x10), UF_UPDATE_RESET_BLOCK_COUNT (0x20), UF_UPDATE_RESET_ROTATION_COUNT (0x40), UF_PENDING_UPDATES (0x80)
- **Output Mode**
  - OUT_MODE_COAST (0x00), OUT_MODE_MOTORON (0x01), OUT_MODE_BRAKE (0x02), OUT_MODE_REGULATED (0x04), OUT_MODE_REGMETHOD (0xF0)
- **Output Run State**
  - OUT_RUNSTATE_IDLE (0x00), OUT_RUNSTATE_RAMPUP (0x10), OUT_RUNSTATE_RUNNING (0x20), OUT_RUNSTATE_RAMPDOWN (0x40)
- **Output Regulation Mode**
  - OUT_REGMODE_IDLE (0), OUT_REGMODE_SPEED (1), OUT_REGMODE_SYNC (2)
- **Input Types**
  - IN_TYPE_NO_SENSOR (0x0), IN_TYPE_SWITCH (0x1), IN_TYPE_TEMPERATURE (0x2), IN_TYPE_REFLECTION (0x3), IN_TYPE_ANGLE (0x4), IN_TYPE_LIGHT_ACTIVE (0x5), IN_TYPE_LIGHT_INACTIVE (0x6), IN_TYPE_SOUND_DB (0x7), IN_TYPE_SOUND_DBA (0x8), IN_TYPE_CUSTOM (0x9), IN_TYPE_LOWSPEED (0xA), IN_TYPE_LOWSPEED_9V (0xB), IN_TYPE_HISPEED (0xC)
- **Input Modes**
  - IN_MODE_RAW (0x00), IN_MODE_BOOLEAN (0x20), IN_MODE_TRANSITIONCNT (0x40), IN_MODE_PERIODCOUNTER (0x60), IN_MODE_PCTFULLSCALE (0x80), IN_MODE_CELSIUS (0xA0), IN_MODE_FAHRENHEIT (0xC0), IN_MODE_ANGLESTEP (0xE0), IN_MODE_SLOPEMASK (0x1F), IN_MODE_MODEMASK (0xE0)
- **IO Map Address Constants**
  - IO_BASE (0xc000), MOD_INPUT (0x0000), MOD_OUTPUT (0x0200), IO_IN_FPP (6), IO_OUT_FPP (15)
- **IO Map Addresses for Inputs**
  - InputIOType0 (0xc000), InputIOInputMode0 (0xc001), InputIORawValue0 (0xc002), InputIONormalizedValue0 (0xc003), InputIOScaledValue0 (0xc004), InputIOInvalidData0 (0xc005), InputIOType1 (0xc006), InputIOInputMode1 (0xc007), InputIORawValue1 (0xc008), InputIONormalizedValue1 (0xc009), InputIOScaledValue1 (0xc00a), InputIOInvalidData1 (0xc00b), InputIOType2 (0xc00c), InputIOInputMode2 (0xc00d), InputIORawValue2 (0xc00e), InputIONormalizedValue2 (0xc00f), InputIOScaledValue2 (0xc010), InputIOInvalidData2 (0xc011), InputIOType3 (0xc012), InputIOInputMode3 (0xc013), InputIORawValue3 (0xc014), InputIONormalizedValue3 (0xc015), InputIOScaledValue3 (0xc016), InputIOInvalidData3 (0xc017)
- **IO Map Addresses for Outputs**
  - OutputIOUpdateFlags0 (0xc200), OutputIOOutputMode0 (0xc201), OutputIOPower0 (0xc202), OutputIOActualSpeed0 (0xc203), OutputIOTachoCount0 (0xc204), OutputIOTachoLimit0 (0xc205), OutputIORunState0 (0xc206), OutputIOTurnRatio0 (0xc207), OutputIORegMode0 (0xc208), OutputIOOverload0 (0xc209), OutputIORegPValue0 (0xc20a), OutputIORegIValue0 (0xc20b),

OutputIORegDValue0 (0xc20c), OutputIOBlockTachoCount0 (0xc20d), OutputIORotationCount0 (0xc20e), OutputIOUpdateFlags1 (0xc20f), OutputIOOutputMode1 (0xc210), OutputIOPower1 (0xc211), OutputIOActualSpeed1 (0xc212), OutputIOTachoCount1 (0xc213), OutputIOTachoLimit1 (0xc214), OutputIORunState1 (0xc215), OutputIOTurnRatio1 (0xc216), OutputIORegMode1 (0xc217), OutputIOOverload1 (0xc218), OutputIORegPValue1 (0xc219), OutputIORegIValue1 (0xc21a), OutputIORegDValue1 (0xc21b), OutputIOBlockTachoCount1 (0xc21c), OutputIORotationCount1 (0xc21d), OutputIOUpdateFlags2 (0xc21e), OutputIOOutputMode2 (0xc21f), OutputIOPower2 (0xc220), OutputIOActualSpeed2 (0xc221), OutputIOTachoCount2 (0xc222), OutputIOTachoLimit2 (0xc223), OutputIORunState2 (0xc224), OutputIOTurnRatio2 (0xc225), OutputIORegMode2 (0xc226), OutputIOOverload2 (0xc227), OutputIORegPValue2 (0xc228), OutputIORegIValue2 (0xc229), OutputIORegDValue2 (0xc22a), OutputIOBlockTachoCount2 (0xc22b), OutputIORotationCount2 (0xc22c)

## 67. Motor API Macros

- **OnFwd**(ports, power): start the motors turning at the specified power level.
- **OnFwdEx**(ports, power, reset): Same as OnFwd but allows specifying whether to reset one or more of the tachometer counters.
- **OnRev**(ports, power): same as OnFwd but in the opposite direction.
- **OnRevEx**(ports, power, reset): same as OnFwdEx but in the opposite direction.
- **OnFwdReg**(ports, power, regmode): Same as OnFwd but allows for choosing the regulation mode
- **OnFwdRegEx**(ports, power, regmode, reset): Same as OnFwdEx but allows for choosing the regulation mode.
- **OnRevReg**(ports, power, regmode): Opposite direction of OnFwdReg.
- **OnRevRegEx**(ports, power, regmode): Opposite direction of OnFwdRegEx.
- **OnFwdSync**(ports, power, turnpct): Same as OnFwd but allows synchronizing multiple motors.
- **OnFwdSyncEx**(ports, power, turnpct, reset): Same as OnFwdEx but allows synchronizing multiple motors.
- **OnRevSync**(ports, power, turnpct): Opposite direction of OnFwdSync.
- **OnRevSyncEx**(ports, power, turnpct, reset): Opposite direction of OnFwdSyncEx.
- **RotateMotor**(ports, power, angle): Rotate motors at power for specified angle. Negative power settings turn the opposite direction.
- **RotateMotorPID**(ports, power, angle, p, i, d): Same as RotateMotor but with the additional ability to tune the PID parameters.

- **RotateMotorEx**(ports, power, angle, turnpct, bsync, bstop): Same as RotateMotor but with additional ability to synchronize motors, specify a turn ratio, and tell the motors whether to brake at the end of the angle of rotation.
- **RotateMotorExPID**(ports, power, angle, turnpct, bsync, bstop, p, i, d): Same as RotateMotorEx but with the additional ability to tune the PID parameters.
- **Coast**(ports): Stop motors without braking.
- **Float**(ports): An alias for Coast.
- **CoastEx**(ports): Same as Coast but allows specifying whether to reset one or more of the tachometer counters.
- **Off**(ports): Stop motors with braking.
- **OffEx**(ports): Same as Off but allows specifying whether to reset one or more of the tachometer counters.
- **ResetTachoCount**(ports): Reset the tachometer count for the specified motors.
- **ResetBlockTachoCount**(ports): Reset the block tachometer count for the specified motors.
- **ResetRotationCount**(ports): Reset the rotation count for the specified motors.
- **ResetAllTachoCounts**(ports): Reset all the counters for the specified motors.

## 68. Sensor API Macros

- **Port can be either a variable or a constant**
  - **SetSensorType**(port, type): set sensor type.
  - **SetSensorMode**(port, mode): set sensor mode.
  - **ClearSensor**(port): clear the sensor (set to zero).
  - **SetSensorTouch**(port): set type and mode as touch sensor.
  - **SetSensorLight**(port): set type and mode as light sensor.
  - **SetSensorSound**(port): set type and mode as sound sensor.
  - **SetSensorUltrasonic**(port): set type and mode as ultrasonic sensor.
  - **SetSensorLowspeed**(port): set type and mode as an I2C (aka lowspeed) sensor.
  - **ResetSensor**(port): wait for sensor value to become valid.
  - **ReadSensor**(port, value): read sensor into value.
  - **ReadSensorUS**(port, value): read ultrasonic sensor into value.
  - **ReadI2CBytes**(port, inbuf, count, outbuf, result): read data from an I2C sensor.

## 69. Drawing API Macros

- **TextOu**t(x, y, text): draw the text at (x,y). Do not clear the screen.

- **TextOu**tEx(x, y, text, cls): draw the text at (x,y). Optionally clear the screen before drawing based on the value of the boolean "cls" argument.
- **NumOut**(x, y, numeric): draw the number at (x, y). Do not clear the screen.
- **NumOutEx**(x, y, numeric, cls): draw the number at (x, y). Optionally clear the screen before drawing based on the value of the boolean "cls" argument.
- **PointOut**(x, y): draw a point at (x,y). Do not clear the screen.
- **PointOutEx**(x, y): draw a point at (x,y). Optionally clear the screen before drawing based on the value of the boolean "cls" argument.
- **LineOut**(x1, y1, x2, y2): draw a line from (x1,y1) to (x2,y2). Do not clear the screen.
- **LineOutEx**(x1, y1, x2, y2, cls): draw a line from (x1,y1) to (x2,y2). Optionally clear the screen before drawing based on the value of the boolean "cls" argument.
- **RectOut**(x, y, width, height): draw a rectangle at (x,y) with specified width and height. Do not clear the screen.
- **RectOutEx**(x, y, width, height, cls): draw a rectangle at (x,y) with specified width and height. Optionally clear the screen before drawing based on the value of the boolean "cls" argument.
- **CircleOut**(x, y, radius): draw a circle centered at (x,y) with specified radius. Do not clear the screen.
- **CircleOutEx**(x, y, radius, cls): draw a circle centered at (x,y) with specified radius. Optionally clear the screen before drawing based on the value of the boolean "cls" argument.
- **GraphicOut**(x, y, filename): draw the RIC icon file specified by filename at (x,y). Do not clear the screen.
- **GraphicOutEx**(x, y, filename, vars, cls): draw the RIC icon file specified by filename at (x,y). Use the included 16 element array of values as a transformation function for the commands in the RIC file. Optionally clear the screen before drawing based on the value of the boolean "cls" argument.

# 70. Sound API Macros

- **PlayTone**(frequency, duration): play a tone. frequency is in Hz. duration is in milliseconds.
- **PlayToneEx**(frequency, duration, volume, loop): play a tone at the specified volume. loop is a boolean value indicating whether to repeat the tone.
- **PlayFile**(filename): play a sound (.rso) or melody (.rmd) file.
- **PlayFileEx**(filename, volume, loop): play a sound or melody file at the specified volume. loop is a boolean value indicating whether to repeat the file.

- **GetSoundState**(state, flags): get the current state of the sound module.
- **SetSoundState**(state, flags, result): set the current state of the sound module.

## 71. Miscellaneous API Macros

- **ReadButtonEx**(idx, reset, pressed, count, result): read the specified button state and return pressed, count, and result. Optionally reset the button state.
- **Random**(variable, maximum): read a random number into variable. The value will be between 0 and maximum.
- **SignedRandom**(variable): read a signed random number into variable. The value will be a signed 16-bit value.
- **ResetSleepTimer**: reset the sleep timer to its initial value. Use this to keep the brick from powering down when the sleep timer has elapsed.
- **GetFirstTick**(value): read the tick from program startup (4 bytes).

## 72. Messaging API Macros

- **SendMessage**(queue, msg, result): send a message to the specified queue.
- **ReceiveMessage**(queue, clear, msg, result): receive a message from the specified queue. Optionally clear the message from the queue.
- **ReceiveRemoteBool**(queue, clear, bval, result): receive a boolean value from the specified queue. Optionally clear the message from the queue.
- **ReceiveRemoteNumber**(queue, clear, val, result): receive a numeric value from the specified queue. Optionally clear the message from the queue.
- **ReceiveRemoteString**(queue, clear, msg, result): Same as ReceiveMessage.
- **ReceiveRemoteMessageEx**(queue, clear, str, val, bval, result): receive a message from the specified queue. This routine attempts to convert the message into a string, a numeric value, and a boolean value. Optionally clear the message from the queue.
- **SendResponseBool**(queue, bval, result): send a response containing a boolean value to the specified queue.
- **SendResponseNumber**(queue, val, result): send a response containing a numeric value to the specified queue.
- **SendResponseString**(queue, msg, result): send a response containing a string to the specified queue.
- **BluetoothStatus**(conn, result): check the status of the specified Bluetooth connection.
- **BluetoothWrite**(conn, buffer, result): write the data in the buffer to the specified Bluetooth connection.

- **SendRemoteBool**(conn, queue, bval, result): send a message containing a boolean value to the specified queue over the specified Bluetooth connection.
- **SendRemoteNumber**(conn, queue, val, result): send a message containing a numeric value to the specified queue over the specified Bluetooth connection.
- **SendRemoteString**(conn, queue, msg, result): send a message containing a string to the specified queue over the specified Bluetooth connection.
- **RemoteMessageRead**(conn, queue, result): sends a MessageRead direct command to the device on the specified connection.
- **RemoteMessageWrite**(conn, queue, msg, result): sends a MessageWrite direct command to the device on the specified connection.
- **RemoteStartProgram**(conn, filename, result): sends a StartProgram direct command to the device on the specified connection.
- **RemoteStopProgram**(conn, result): sends a StopProgram direct command to the device on the specified connection.
- **RemotePlaySoundFile**(conn, filename, bloop, result): sends a PlaySoundFile direct command to the device on the specified connection.
- **RemotePlayTone**(conn, frequency, duration, result): sends a PlayTone direct command to the device on the specified connection.
- **RemoteStopSound**(conn, result): sends a StopSound direct command to the device on the specified connection.
- **RemoteKeepAlive**(conn, result): sends a KeepAlive direct command to the device on the specified connection.
- **RemoteResetScaledValue**(conn, port, result): sends a ResetScaledValue direct command to the device on the specified connection.
- **RemoteResetMotorPosition**(conn, port, brelative, result): sends a ResetMotorPosition direct command to the device on the specified connection.
- **RemoteSetInputMode**(conn, port, type, mode, result): sends a SetInputMode direct command to the device on the specified connection.
- **RemoteSetOutputState**(conn, port, speed, mode, regmode, turnpct, runstate, tacholimit, result): sends a SetOutputState direct command to the device on the specified connection.

## 73. Math API Macros

- **Sqrt**(value, result): sets result to be the square root of value.
- **Sin**(value, result): sets result to be the 100 times the sine of value.
- **Cos**(value, result): sets result to be the 100 times the cosine of value.
- **Asin**(value, result): sets result to be the arc-sine of value. The value must be between -100 and 100.
- **Acos**(value, result): sets result to be the arc-cosine of value. The value must be between -100 and 100.

# 74. File API Macros

- **CreateFile(fname ,** fsize, handle, result): creates a new file with the specified name and size. Returns the write handle and a result code.
- **OpenFileAppend**(fname , fsize, handle, result): opens the specified file for appending if it already exists. Returns the size, a write handle, and a result code.
- **OpenFileRead**(fname , fsize, handle, result): opens the specified file for reading if it already exists. Returns the size, a read handle, and a result code.
- **CloseFile**(handle, result): closes the specified file handle and returns a result code.
- **ResolveHandle**(fname, handle, writeable, result): returns a handle for the specified file if the file is already open for reading or writing/appending. writeable is set to TRUE if the handle is a write handle. It also returns a result code.
- **RenameFile**(oldname, newname, result): renames the specified file handle and returns a result code.
- **DeleteFile**(fname, result): deletes the specified file handle and returns a result code.
- **Read**(handle, n, result): read a binary numeric value from the file associated with the specified file handle. The number of bytes read is based on the size of the type of the variable "n" (i.e., byte = 1, int = 2, long = 4). Returns a result code.
- **ReadLn**(handle, n, result): Reads a binary numeric value from the file followed by a CRLF 2 byte sequence and returns a result code.
- **ReadBytes**(handle, len, buf, result): Reads the specified number of bytes from the file into the buffer provided and returns a result code.
- **ReadLnString**(handle, output, result): Reads a string followed by a CRLF 2 byte sequence into the output string provided and returns a result code.
- **Write**(handle, n, result):
- **WriteLn**(handle, n, result):
- **WriteString**(handle, str, cnt, result):
- **WriteLnString**(handle, str, cnt, result):
- **WriteBytes**(handle, buf, cnt, result):
- **WriteBytesEx**(handle, len, buf, result):

# 75. IOMap Writing API Macros

- **SetIOMapBytes(modName**, offset, cnt, arrIn): write cnt bytes to the specified module at the specified offset.

- **SetIOMapValue(modName**, offset, n): write the value 'n' to the specified module at the specified offset.
- The following are module-specific versions of the generic SetIOMapValue macro
  - **SetUIModuleValue(**offset, n)
  - **SetIOCtrlModuleValue**(offset, n)
  - **SetLoaderModuleValue**(offset, n)
  - **SetCommandModuleValue**(offset, n)
  - **SetSoundModuleValue**(offset, n)
  - **SetButtonModuleValue**(offset, n)
  - **SetInputModuleValue**(offset, n)
  - **SetOutputModuleValue**(offset, n)
  - **SetLowSpeedModuleValue**(offset, n)
  - **SetDisplayModuleValue**(offset, n)
  - **SetCommModuleValue**(offset, n)

## 76. IOMap Reading API Macros

- **GetIOMapBytes**(modName, offset, cnt, arrOut): read cnt bytes from the specified module at the specified offset.
- **GetIOMapValue(modName**, offset, n): read the value 'n' from the specified module at the specified offset.
- The following are module-specific versions of the generic GetIOMapValue macro
  - **GetUIModuleValue**(offset, n)
  - **GetIOCtrlModuleValue**(offset, n)
  - **GetLoaderModuleValue**(offset, n)
  - **GetCommandModuleValue**(offset, n)
  - **GetSoundModuleValue**(offset, n)
  - **GetButtonModuleValue**(offset, n)
  - **GetInputModuleValue**(offset, n)
  - **GetOutputModuleValue**(offset, n)
  - **GetLowSpeedModuleValue**(offset, n)
  - **GetDisplayModuleValue**(offset, n)
  - **GetCommModuleValue**(offset, n)

## 77. UI Module API Macros

- **GetBatteryLevel**(value)
- **GetCommandFlags**(value)
- **GetUIState**(value)
- **GetUIButton**(value)
- **GetVMRunState**(value)
- **GetBatteryState**(value)
- **GetBluetoothState**(value)
- **GetUsbState**(value)
- **GetSleepTimeout**(value)
- **GetSleepTimer**(value)

- **GetRechargeableBattery**(value)
- **GetVolume**(value)
- **GetOnBrickProgramPointer**(value)

- **SetCommandFlags**(n)
- **SetUIState**(n)
- **SetUIButton**(n)
- **SetVMRunState**(n)
- **SetBatteryState**(n)
- **SetBluetoothState**(n)
- **SetUsbState**(n)
- **SetSleepTimeout**(n)
- **SetSleepTimer**(n)

- **SetVolume**(value)
- **SetOnBrickProgramPointer**(value)
- **ForceOff**(n)

## 78. IOCtrl Module API Macros

- **PowerDown**: immediately power down the NXT brick.
- **RebootInFirmwareMode**: immediately boot the NXT into firmware download mode.

## 79. Loader Module API Macros

- **GetFreeMemory**(variable): read the amount of free flash memory into the provided variable (4 bytes).

## 80. Sound Module API Macros

- **GetSoundFrequency**(value)
- **GetSoundDuration**(value)
- **GetSoundSampleRate**(value)
- **GetSoundFlags**(value)
- **GetSoundState**(value)
- **GetSoundMode**(value)
- **GetSoundVolume**(value)

- **SetSoundFrequency**(n)
- **SetSoundDuration**(n)
- **SetSoundSampleRat**e(n)
- **SetSoundFlag**s(n)
- **SetSoundState**(n)
- **SetSoundMode**(n)
- **SetSoundVolume**(n)

# 81. Button Module API Macros

- **GetButtonPressCount**(b, value)
- **GetButtonLongPressCount**(b, value)
- **GetButtonShortReleaseCount**(b, value)
- **GetButtonLongReleaseCount**(b, value)
- **GetButtonReleaseCount**(b, value)
- **GetButtonState**(b, value)

- **SetButtonPressCount**(b, n)
- **SetButtonLongPressCount**(b, n)
- **SetButtonShortReleaseCount**(b, n)
- **SetButtonLongReleaseCount**(b, n)
- **SetButtonReleaseCount**(b, n)
- **SetButtonState**(b, n)

# 82. Input Module API Macros

- **GetInCustomZeroOffset**(p, n)
- **GetInSensorBoolean**(p, n)
- **GetInDigiPinsDirection**(p, n)
- **GetInDigiPinsStatus**(p, n)
- **GetInDigiPinsOutputLevel**(p, n)
- **GetInCustomPercentFullScale**(p, n)
- **GetInCustomActiveStatus**(p, n)

- **SetInCustomZeroOffset**(p, n)
- **SetInSensorBoolean**(p, n)
- **SetInDigiPinsDirection**(p, n)
- **SetInDigiPinsStatus**(p, n)
- **SetInDigiPinsOutputLevel**(p, n)
- **SetInCustomPercentFullScale**(p, n)
- **SetInCustomActiveStatus**(p, n)

# 83. Output Module API Macros

- **GetOutPwnFreq**(v)

- **SetOutPwnFreq**(v)

# 84. LowSpeed Module API Macros

- **GetLSInputBuffer**(p, offset, cnt, data)
- **GetLSInputBufferInPtr**(p, n)
- **GetLSInputBufferOutPtr**(p, n)
- **GetLSInputBufferBytesToRx**(p, n)

- **GetLSOutputBuffer**(p, offset, cnt, data)
- **GetLSOutputBufferInPtr**(p, n)
- **GetLSOutputBufferOutPtr**(p, n)
- **GetLSOutputBufferBytesToRx**(p, n)
- **GetLSMode**(p, v)
- **GetLSChannelState**(p, v)
- **GetLSErrorType**(p, v)
- **GetLSState**(v)
- **GetLSSpeed**(v)

- **SetLSInputBuffer**(p, offset, cnt, data)
- **SetLSInputBufferInPtr**(p, n)
- **SetLSInputBufferOutPtr**(p, n)
- **SetLSInputBufferBytesToRx**(p, n)
- **SetLSOutputBuffer**(p, offset, cnt, data)
- **SetLSOutputBufferInPtr**(p, n)
- **SetLSOutputBufferOutPtr**(p, n)
- **SetLSOutputBufferBytesToRx**(p, n)
- **SetLSMode**(p, v)
- **SetLSChannelState(**p, v)
- **SetLSErrorType**(p, v)
- **SetLSState**(v)
- **SetLSSpeed**(v)

## 85. Display Module API Macros

- **GetDisplayEraseMask**(n)
- **GetDisplayUpdateMask**(n)
- **GetDisplayDisplay**(n)
- **GetDisplayFlags**(n)
- **GetDisplayTextLinesCenterFlags**(n)
- **GetDisplayNormal**(x, line, cnt, data)
- **GetDisplayPopup**(x, line, cnt, data)

- **SetDisplayEraseMask**(n)
- **SetDisplayUpdateMask**(n)
- **SetDisplayDisplay**(n)
- **SetDisplayFlags**(n)
- **SetDisplayTextLinesCenterFlags**(n)
- **SetDisplayNormal**(x, line, cnt, data)
- **SetDisplayPopup**(x, line, cnt, data)

## 86. BT Comm Module API Macros

- **GetBTDeviceName**(p, str)
- **GetBTDeviceClass**(p, v)
- **GetBTDeviceAddress**(p, addr)
- **GetBTDeviceStatus**(p, v)
- **GetBTConnectionName**(p, str)

- **GetBTConnectionClass**(p, v)
- **GetBTConnectionPinCode**(p, str)
- **GetBTConnectionAddress**(p, addr)
- **GetBTConnectionHandleNum**(p, v)
- **GetBTConnectionStreamStatus**(p, v)
- **GetBTConnectionLinkQuality**(p, v)
- **GetBrickDataName**(str)
- **GetBrickDataBluecoreVersion**(v)
- **GetBrickDataAddress**(addr)
- **GetBrickDataBtStateStatus**(v)
- **GetBrickDataBtHardwareStatus**(v)
- **GetBrickDataTimeoutValue**(v)
- **GetBTInputBuffer**(data)
- **GetBTInputBufferInPtr**(n)
- **GetBTInputBufferOutPtr**(n)
- **GetBTOutputBuffer**(data)
- **GetBTOutputBufferInPtr**(n)
- **GetBTOutputBufferOutPtr**(n)
- **GetBTDeviceCount**(n)
- **GetBTDeviceNameCount**(n)

- **SetBTDeviceName**(p, str)
- **SetBTDeviceClass**(p, v)
- **SetBTDeviceAddress**(p, addr)
- **SetBTDeviceStatus**(p, v)
- **SetBTConnectionName**(p, str)
- **SetBTConnectionClass**(p, v)
- **SetBTConnectionPinCode**(p, str)
- **SetBTConnectionAddress**(p, addr)
- **SetBTConnectionHandleNum**(p, v)
- **SetBTConnectionStreamStatus**(p, v)
- **SetBTConnectionLinkQuality**(p, v)
- **SetBrickDataName**(str)
- **SetBrickDataBluecoreVersion**(v)
- **SetBrickDataAddress**(addr)
- **SetBrickDataBtStateStatus**(v)
- **SetBrickDataBtHardwareStatus**(v)
- **SetBrickDataTimeoutValue**(v)
- **SetBTInputBuffer**(data)
- **SetBTInputBufferInPtr**(n)
- **SetBTInputBufferOutPtr**(n)
- **SetBTOutputBuffer**(data)
- **SetBTOutputBufferInPtr**(n)
- **SetBTOutputBufferOutPtr**(n)
- **SetBTDeviceCount**(n)
- **SetBTDeviceNameCount**(n)

## 87. USB/HS Comm Module API Macros

- **GetUSBInputBuffer**(data)

- **GetUSBInputBufferInPtr**(n)
- **GetUSBInputBufferOutPtr**(n)
- **GetUSBOutputBuffer**(data)
- **GetUSBOutputBufferInPtr**(n)
- **GetUSBOutputBufferOutPtr**(n)
- **GetUSBPollBuffer**(data)
- **GetUSBPollBufferInPtr**(n)
- **GetUSBPollBufferOutPtr**(n)
- **GetUSBState**(n)

- **GetHSInputBuffer**(data)
- **GetHSInputBufferInPtr**(n)
- **GetHSInputBufferOutPtr**(n)
- **GetHSOutputBuffer**(data)
- **GetHSOutputBufferInPtr**(n)
- **GetHSOutputBufferOutPtr**(n)
- **GetHSFlags**(n)
- **GetHSSpeed**(n)
- **GetHSState**(n)

- **SetUSBInputBuffer**(data)
- **SetUSBInputBufferInPtr**(n)
- **SetUSBInputBufferOutPtr**(n)
- **SetUSBOutputBuffer**(data)
- **SetUSBOutputBufferInPtr**(n)
- **SetUSBOutputBufferOutPtr**(n)
- **SetUSBPollBuffer**(data)
- **SetUSBPollBufferInPtr**(n)
- **SetUSBPollBufferOutPtr**(n)
- **SetUSBState**(n)

- **SetHSInputBuffer**(data)
- **SetHSInputBufferInPtr**(n)
- **SetHSInputBufferOutPtr**(n)
- **SetHSOutputBuffer**(data)
- **SetHSOutputBufferInPtr(n)**
- **SetHSOutputBufferOutPtr**(n)
- **SetHSFlags**(n)
- **SetHSSpeed**(n)
- **SetHSState**(n)

# 88. Compiler Usage

- **Extract compiler to the directory of your choice**
- **Open a command prompt or terminal window**
- **Execute NBC**
  - D:\Bricxcc\>nbc
    Next Byte Codes Compiler version 1.0.1.b30 (1.0.1.30, built Jun 05 2007 06:25:36)
    Copyright (c) 2006, John Hansen
    Use "nbc -help" for more information.

- D:\Bricxcc\>nbc –help
  Next Byte Codes Compiler version 1.0.1.b30 (1.0.1.30, built Jun 05 2007 06:25:36)
  Copyright (c) 2006, John Hansen
  Syntax: nbc [options] filename [options]

  -S=<portname>: specify port name (COMn or usb), resource name, or alias
  -BT: use bluetooth
  -d: download program
  -b: treat input file as a binary file (don't compile it)
  -q: no sound upon download completion
  -x: decompile program
  -O=<outfile> : specify output file
  -E=<filename> : write compiler errors to <filename>
  -I=<path>: search <path> for include files
  -L=<filename> : generate code listing to <filename>
  -help : display command line options
- D:\Bricxcc\>nbc –S=usb –d testinc.nbc
- The previous command produced no output because there were no program errors and the download succeeded.