

Robodyyssey Mouse

Programming Guide (BasicX)

Version 1.01 - Modified 7/31/2002

Robodyyssey Systems, LLC.
20 Quimby Avenue
Trenton, New Jersey 08610

Phone/Fax: 609-585-8535
Web: www.robodyssey.com
Email: info@robodyssey.com

Copyright 2002 Robodyssey Systems, LLC. All rights reserved.

Basic Express and BasicX are trademarks of NetMedia, Inc.

Basic Stamp is a trademark of Parallax, Inc.

1 Getting Started

This guide is not intended to serve as an introductory guide to programming, and assumes prior programming experience. We will instead show you how to apply your existing programming knowledge to program the Robodyssey Mouse with a BasicX microcontroller.

To get started programming your new Mouse, make sure you have the following items available:

1. Fully Assembled Robodyssey Mouse with Robodyssey Advanced Motherboard
2. BasicX microcontroller installed in Robodyssey Advanced Motherboard
3. Power Pack (7V - 9V)
4. 9 Pin Male - Female Serial Cable
5. BasicX Software (Available on Robodyssey CD or from www.basicx.com)

Install the BasicX Software if you have not done so already. Connect your serial cable to your Mouse and computer, and open the BasicX Development System.

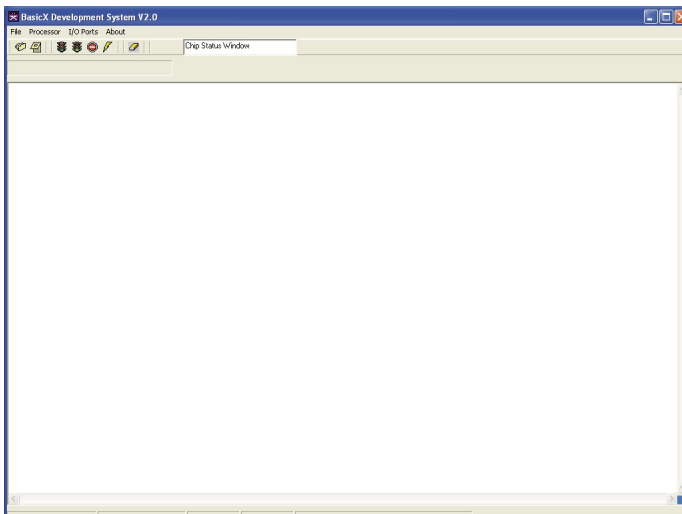


Figure 1 - BasicX Development System

Open the BasicX Editor, click New Project, and enter Project and Module names for your new program.

To compile your program, select Compile from the Compile menu. When you're ready to download to the BasicX, press the lightning bolt symbol in the BasicX Development System window.

To compile, download and run your program with a single command, select Compile and Run from the Compile menu.

More information on the BasicX software can be found in the BasicX documentation.

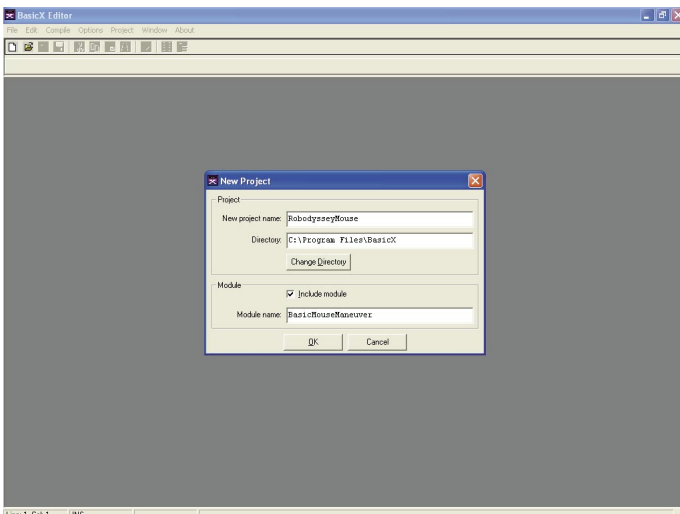


Figure 2 - BasicX Editor - New Project Dialog

2 BasicX Language

The BasicX Language we will be using is very similar to Microsoft's Visual Basic. For a detailed System Library, please see your Robodyssey CD or our web site for the BasicX documentation. Below you will find a quick reference for the available data types, as well as a listing of the System Library.

Type	Storage	Range
Boolean	8 bits	True .. False
Byte	8 bits	0 .. 255
Integer	16 bits	-32 768 .. 32 767
Long	32 bits	-2 147 483 648 .. 2 147 483 647
Single	32 bits	-3.402 823 E+38 .. 3.402 823 E+38
String	Varies	0 to 64 characters

Math functions

Abs	Absolute value
ACos	Arc cosine
ASin	Arc sine
Atn	Arc tangent
Cos	Cosine
Exp	Raises e to a specified power
Exp10	Raises 10 to a specified power
Fix	Truncates a floating point value
Log	Natural log
Log10	Log base 10
Pow	Raises an operand to a given power
Sin	Sine
Sqr	Square root
Tan	Tangent

Queues

GetQueue	Reads data from a queue
OpenQueue	Defines an array as a queue
PeekQueue	Looks at queue data without removing any data
PutQueue	Writes data to a queue
PutQueueStr	Writes a string to a queue
StatusQueue	Determines if a queue has data available for reading

Real time clock

GetDate	Returns the date
GetDayOfWeek	Returns the day of week
GetTime	Returns the time of day
GetTimestamp	Returns the date and time of day
PutDate	Sets the date
PutTime	Sets the time of day
PutTimestamp	Sets the date, day of week and time of day
Timer	Returns floating point seconds since midnight

Memory-related functions

BlockMove	Copies a block of data in RAM
FlipBits	Generates mirror image of bit pattern
GetBit	Reads a single bit from a variable
GetEEPROM	Reads data from EEPROM
MemAddress	Returns the address of a variable or array
MemAddressU	Returns the address of a variable or array
PersistentPeek	Reads a byte from EEPROM
PersistentPoke	Writes a byte to EEPROM
PutBit	Writes a single bit to a variable
PutEEPROM	Writes data to EEPROM
RAMpeek	Reads a byte from RAM
RAMpoke	Writes a byte to RAM
SerialNumber	Returns the BasicX version number

String functions

Asc	Returns the ASCII code of a character
Chr	Converts a numeric value to a character
LCase	Converts string to lower case
Len	Returns the length of a string
Mid	Copies a substring
Trim	Trims leading and trailing blanks from string
UCase	Converts string to upper case

Type conversions

CBool	Convert Byte to Boolean
CByte	Convert to Byte
CInt	Convert to Integer
CLng	Convert to Long
CSng	Convert to floating point (single)
CStr	Convert to string
CuInt	Convert to UnsignedInteger
CuLng	Convert to UnsignedLong
FixB	Truncates a floating point value, converts to Byte
FixI	Truncates a floating point value, converts to Integer
FixL	Truncates a floating point value, converts to Long
FixUI	Truncates a floating point value, converts to UnsignedInteger
FixUL	Truncates a floating point value, converts to UnsignedLong

Tasking

CallTask	Starts a task
CPUsleep	Puts the processor in various low-power modes
Delay	Pauses task and allows other tasks to run
DelayUntilClockTick	Pauses task until the next tick of the real time clock
FirstTime	Determines whether the program has ever been run since download
LockTask	Locks the task and discourages other tasks from running
OpenWatchdog	Starts the watchdog timer
ResetProcessor	Resets and reboots the processor
Semaphore	Coordinates the sharing of data between tasks
Sleep	Pauses task and allows other tasks to run
TaskIsLocked	Determine whether a task is locked
UnlockTask	Unlocks a task
WaitForInterrupt	Allows a task to respond to a hardware interrupt
Watchdog	Resets the watchdog timer

3 Common Constants

Although you could certainly program your Mouse anyway you like, we've standardized on a common setup of constants to make pin configurations and software sharing a bit easier.

Be sure to connect your servos to your Robodyssey Motherboard as described in your Assembly Instructions. Here's a quick description of how to connect your servos.

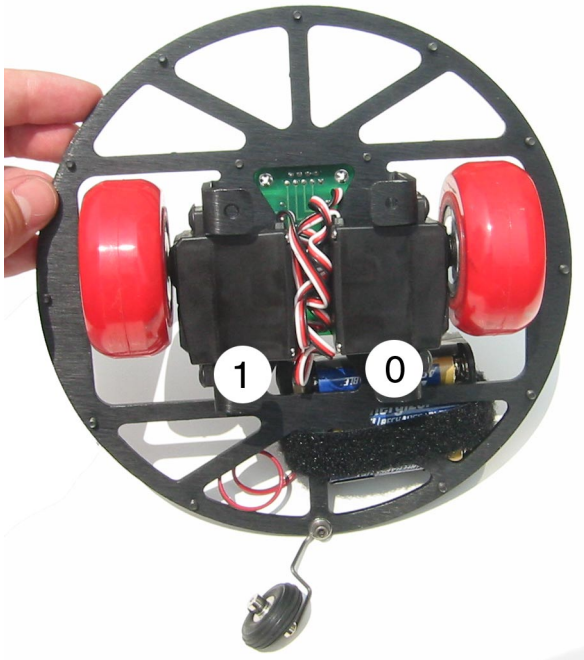


Figure 3 - Proper Servo Connections

```
Option Explicit

Const LeftServoPin As Byte = 5
Const RightServoPin As Byte = 6

Const LeftServoReverse As Single = 0.001
Const LeftServoForward As Single = 0.002
Const RightServoReverse As Single = 0.002
Const RightServoForward As Single = 0.001

Public Sub Main()

End Sub
```

Figure 4 - Servo Constants

As shown in Figure 3, connect the right servo to Pin 0 on the Robodyssey Advanced Motherboard, and the left servo to Pin 1.

Now we can declare two constants, LeftServoPin and RightServoPin, as well as constants for the forward and reverse pulse timings for each servo. See Figure 4 for the code.

The pin numbering is slightly different between the Robodyssey Advanced Motherboard and the BasicX software. The Motherboard I/O pins are numbered starting from zero, while in the BasicX software the pins are referenced by the physical locations on the chip. Therefore, Pin 0 on the Motherboard corresponds to Pin 5 in the BasicX software.

To make sure your physical connections and your software are in sync, always add five to the Motherboard pin number to get the correct number to use in your software.

4 SendServoCommands Function

Robodyssey has also standardized on a function to easily send commands to the servos for movement. By using SendServoCommands, you ensure that each servo receives the proper number of pulses at the proper frequency. Feel free to alter the values as you see fit for increased/decreased speed, wheel turn time, etc.

```
Sub SendServoCommands(ByVal LeftDirection As Single, ByVal RightDirection As Single)
Dim i As Byte

For i=0 to 17      'Change the length of the loop to increase/decrease wheel turn time

    Call PulseOut(LeftServoPin, LeftDirection, 1)
    Call PulseOut(RightServoPin, RightDirection, 1)

    Call Delay(0.02)    'Pulse frequency of approximately 50Hz. Adjust for speed.
Next

End Sub
```

Figure 5 - SendServoCommands Function

To call the function and move the servos, use the command in Figure 6.

```
Call SendServoCommands(LeftServoForward, RightServoForward)
```

Figure 6 - Call SendServoCommands

5 Forward Movement

Execute the following program to move the Mouse continuously forward.

```
Option Explicit

Const LeftServoPin As Byte = 5
Const RightServoPin As Byte = 6
Const LeftServoReverse As Single = 0.001
Const LeftServoForward As Single = 0.002
Const RightServoReverse As Single = 0.002
Const RightServoForward As Single = 0.001

Public Sub Main()
Do
    Call SendServoCommands(LeftServoForward, RightServoForward)
Loop
End Sub

Sub SendServoCommands(ByVal LeftDirection As Single, ByVal RightDirection As Single)
Dim i As Byte
For i=0 to 17
    Call PulseOut(LeftServoPin, LeftDirection, 1)
    Call PulseOut(RightServoPin, RightDirection, 1)
    Call Delay(0.02)
Next
End Sub
```

Figure 7 - Forward Movement

6 Right & Left Turns

Using the Mouse's two wheel differential drive, it's easy to make zero-radius turns just by sending the servos opposite direction commands.

```
Call SendServoCommands (LeftServoReverse, RightServoForward) ` Left Turn
```

```
Call SendServoCommands (LeftServoForward, RightServoReverse) ` Right Turn
```

Figure 8 - Left & Right Turns

Now that we've covered the Mouse's movement, we'll look at interfacing the Mouse to the physical world with various sensors.

7 Switch Sensors

The BasicX must be properly configured to use a switch as a sensor. Directly connecting the switch to the BasicX could result in a short circuit. To prevent this, we will activate the internal pullup resistors of the BasicX for the input pins we wish to use with switches. In the BasicX Editor, go to the Project -> Chip dialog box, as shown in Figure 11. There are four states for each of the 16 I/O pins:

IN Input Configures the corresponding pin as a tristate (high impedance) input. Typically 5 V is logic high (on 5 V systems) and 0 V is logic low.

P Input w/Pullup Configures the pin as an input with pullup. This state is typically used to sense the state of passive devices such as switches.

0 Output Low Configures the corresponding pin as a logic low output, which is 0 V.

1 Output High Configures the pin as logic high output, which is typically 5 V.

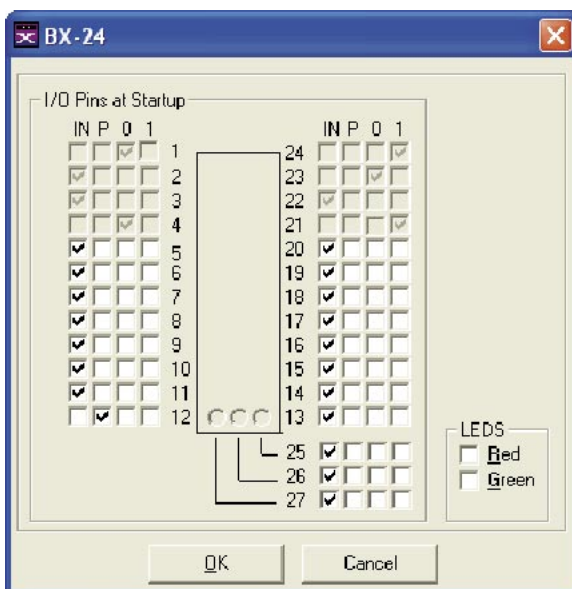


Figure 9 - BX24 Chip Settings Dialog

To use a simple switch with the BasicX, we're going to configure our input pin as "Input with Pullup." The switch will be used to connect the input pin to ground when closed. In the switch's open state, the pin will be high. In the switch's closed state, the pin will be low.

The program in Figure 10 demonstrates how to make the Mouse move forward until the switch is closed.

```

Option Explicit

Const LeftServoPin As Byte = 5
Const RightServoPin As Byte = 6
Const LeftServoReverse As Single = 0.001
Const LeftServoForward As Single = 0.002
Const RightServoReverse As Single = 0.002
Const RightServoForward As Single = 0.001

Const SwitchSensorPin As Byte = 12

Public Sub Main()
Do
    If GetPin(SwitchSensorPin) = 1 Then
        Call SendServoCommands(LeftServoForward, RightServoForward)
    Else
        Sleep(1.0)
    End If
Loop
End Sub

Sub SendServoCommands(ByVal LeftDirection As Single, ByVal RightDirection As Single)
Dim i As Byte
For i=0 to 17
    Call PulseOut(LeftServoPin, LeftDirection, 1)
    Call PulseOut(RightServoPin, RightDirection, 1)
    Call Delay(0.02)
Next
End Sub

```

Figure 10 - Forward Movement Until Switch is Closed

8 Digital Sensors

To use digital sensors, configure each I/O pin in the Chip Settings Dialog box to IN (Tristate High Impedance Input). Connect your sensor to the Robodyssey Advanced Motherboard, and simply use `GetPin(PinNumber)` to retrieve the binary state of the pin. Any I/O pins on the Motherboard may be used for digital input. Pins 0 - 7 have the same V+ as the power supply, while pins 8 - 15 are connected to a regulated 5V V+.

For information on using I2C and other serial sensors, see the BasicX documentation.

9 Analog Sensors

The BasicX microcontroller has 8 10-bit Analog to Digital Converters (ADC) that allow voltage measurements from 0V to 5V. The ADC's are available on Pins 8 - 15 on the Motherboard, or 13 - 20 in the BasicX software. Each pin to be used should be configured as a regular Input pin in the Chip Settings dialog.

To connect a Sharp GP2D12 Infrared Range Finder, follow the Wire Assembly Instructions on the Robodyssey CD for creating the connector for the Robodyssey Advanced Motherboard.

With your connector complete, connect the sensor to any of the ADC pins on the Motherboard. The Sharp GP2D12 outputs an analog voltage inversely proportional to the distance of the measured object. Basically, the greater the voltage measured from the sensor, the closer the object and vice versa. See Figure 11 for a graph of distance versus measured voltage.

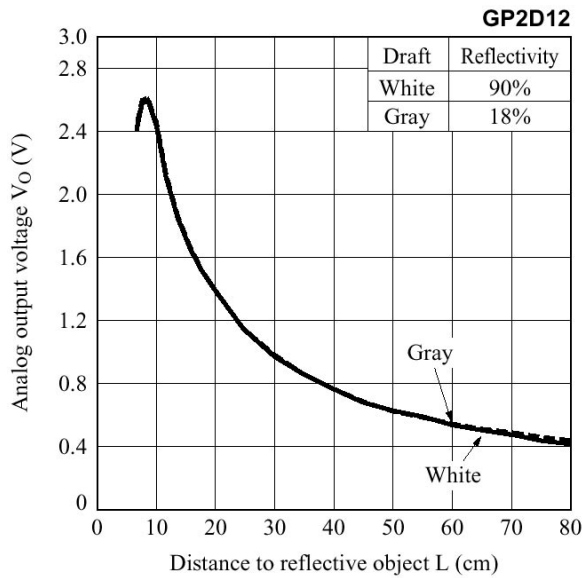


Figure 11 - Sharp GP2D12 Distance versus Voltage

Option Explicit

```
Const LeftServoReverse As Single = 0.001
Const RightServoReverse As Single = 0.002
```

```
Const LeftServoForward As Single = 0.002
Const RightServoForward As Single = 0.001
```

```
Const FrontSensorPin As Byte = 20
Const LeftSensorPin As Byte = 19
Const RightSensorPin As Byte = 18
```

```
Const LeftServoPin As Byte = 5
Const RightServoPin As Byte = 6
```

```
Const ObjectDetectionThreshold As Integer = 250
```

```
Dim ContinueMoving As Boolean
Dim SensorMonitorStack (1 To 128) As Byte
```

```
Public Sub Main()
```

```
ContinueMoving = true
CallTask "SensorMonitor", SensorMonitorStack
```

```
Do
```

```
    If ContinueMoving = true Then
        Call SendServoCommands(LeftServoForward, RightServoForward)
    End If
```

```
Loop
```

```
End Sub
```

```
\Continued on next page
```

Figure 12 contains a program that uses three Sharp GP2D12 sensors to allow a Mouse to navigate a desktop without colliding into any objects or falling off the edge. A five sensor setup is much more reliable, with two forward looking sensors (one left, one right), two downward looking sensors in the front (one left, one right), and a single downward looking sensor in the rear.

The program in Figure 12 also utilizes the multitasking capabilities of the BasicX to continuously monitor all three sensors. For more information, see the BasicX documentation.

Figure 12 - Table Navigation (Continued on Next Page)

```

Public Sub SensorMonitor()
Dim LeftIR As Integer
Dim RightIR As Integer
Dim FrontIR As Integer
    Do
        LeftIR = GetADC(LeftSensorPin)
        RightIR = GetADC(RightSensorPin)
        FrontIR = GetADC(FrontSensorPin)

        If LeftIR > ObjectDetectionThreshold Then
            If RightIR > ObjectDetectionThreshold Then
                ContinueMoving = false
                AvoidFront
            Else
                ContinueMoving = false
                AvoidLeft
            End If
        ElseIf RightIR > ObjectDetectionThreshold Then
            If LeftIR > ObjectDetectionThreshold Then
                ContinueMoving = false
                AvoidFront
            Else
                ContinueMoving = false
                AvoidRight
            End If
        End If
        If FrontIR < 300 Then
            ContinueMoving = false
            AvoidEdge
        End If
        Sleep(0.1)
    Loop
End Sub

Sub AvoidLeft()
    Call SendServoCommands(LeftServoForward, RightServoReverse)
    ContinueMoving = true
End Sub

Sub AvoidRight()
    Call SendServoCommands(LeftServoReverse, RightServoForward)
    ContinueMoving = true
End Sub

Sub AvoidFront()
    Call SendServoCommands(LeftServoReverse, RightServoReverse)
    Call SendServoCommands(LeftServoForward, RightServoReverse)
    ContinueMoving = true
End Sub

Sub AvoidEdge()
    LockTask
    Call SendServoCommands(LeftServoReverse, RightServoReverse)
    Call SendServoCommands(LeftServoReverse, RightServoForward)
    ContinueMoving = true
    UnlockTask
End Sub

Sub SendServoCommands(ByVal LeftDirection As Single, ByVal RightDirection As Single)
Dim i As Byte
For i=0 to 17
    Call PulseOut(LeftServoPin, LeftDirection, 1)
    Call PulseOut(RightServoPin, RightDirection, 1)
    Call Delay(0.014)
Next
End Sub

```

10 Additional Sensors

Robodyssey carries various additional sensors that are easy to interface to your Mouse. Each sensor comes with an application guide on how to operate and write the necessary software. For detailed information on the sensors we carry, visit our web site at <http://www.robodyssey.com>.