



# **Basic Express Compiler User's Guide**

---

Version 2.1

---

© 1998-2003 by NetMedia, Inc. All rights reserved.

Basic Express, BasicX, BX-01, BX-24 and BX-35 are trademarks of NetMedia, Inc.

Microsoft, Windows and Visual Basic are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

2.01S

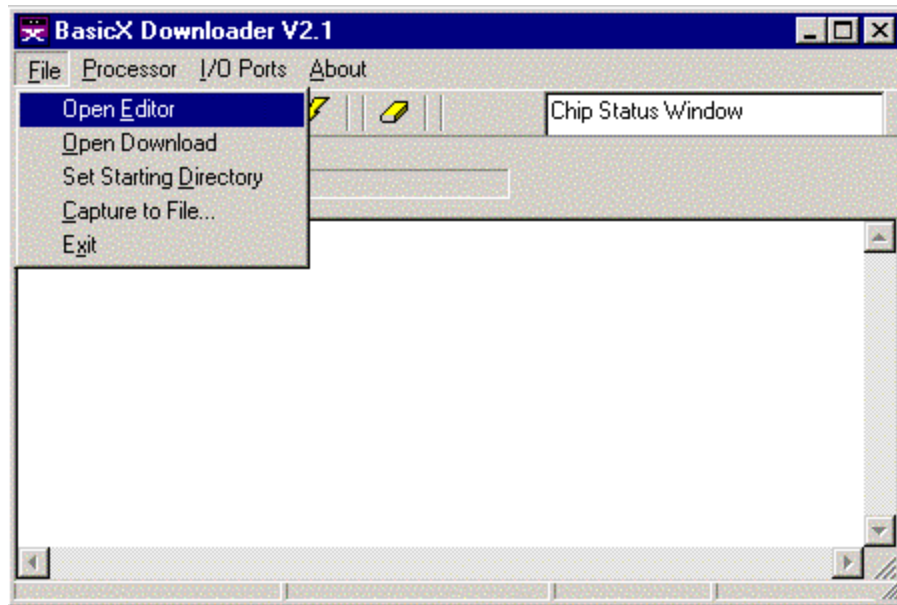
# Contents

1. Getting started .....	4
2. Editor and compiler .....	5
3. Project dialog box .....	10
4. BX-01 chip dialog box .....	11
5. BX-24 and BX-35 chip dialog boxes .....	14
6. Downloader .....	16
7. Watch window .....	20
8. Tutorial on using the watch window .....	22
9. File types .....	25
10. Command line operation .....	28
11. Troubleshooting serial downloads .....	29
12. ATN diagnostic windows .....	30
13. VB project generator .....	31
14. Index .....	37

# Getting Started

When you start the BasicX program, the first thing that appears is the downloader window. This is where programs are downloaded to the BasicX system.

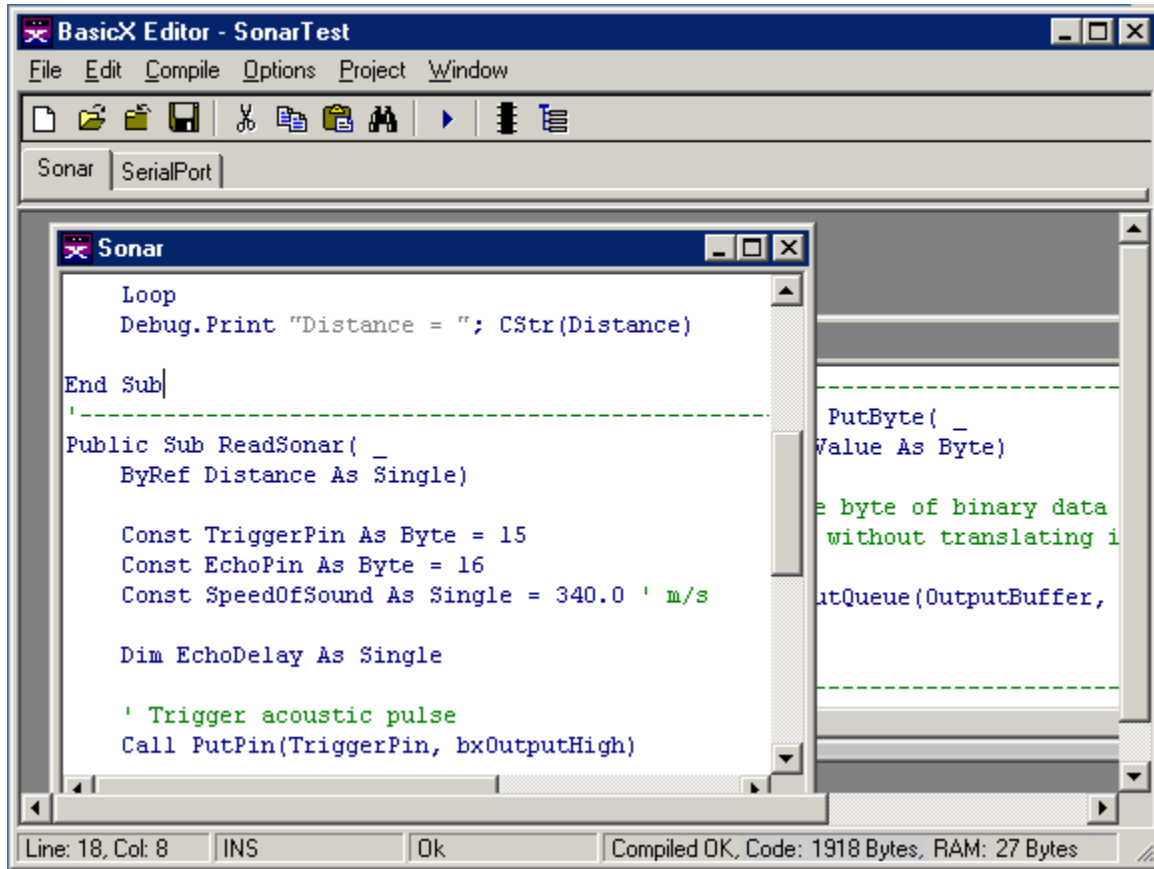
If you want to write a new program, the first step is to open the editor window. In the downloader window, select File / Open Editor:



At this point, the editor window is opened, and you can create, edit and compile a BasicX program. The following section explains the editor and compiler.

# Editor and Compiler

The BasicX Editor/Compiler is where your project and its accompanying files are opened, edited and compiled.



## Editor Window

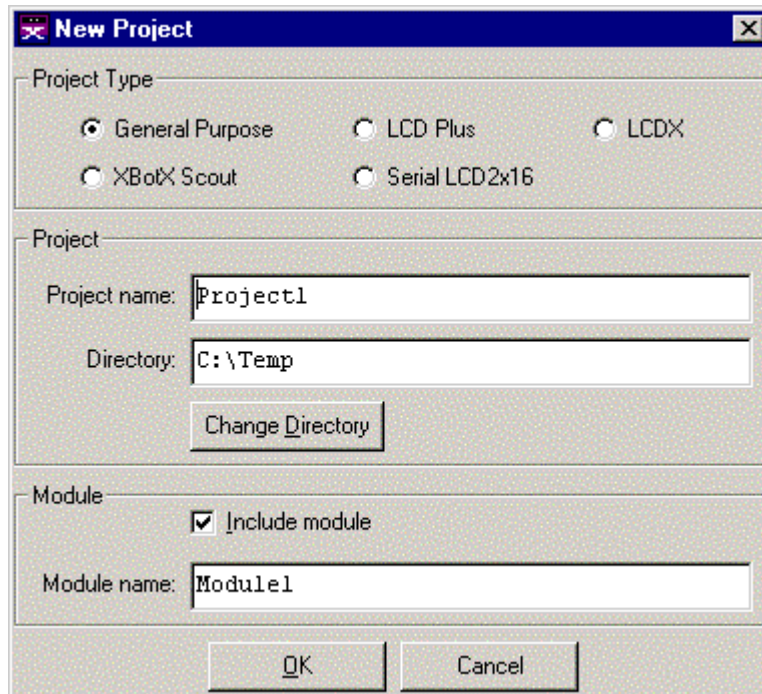
### Title Bar and Tabs

The Title Bar displays the project currently open. The tabs and nested windows display the names of each module being edited.

## File Menu

The File Menu opens and saves project and module files. Each BXP project file keeps track of the collection of modules that make up each program.

The *New Project* selection causes the following window to appear, and allows you to create one of several types of new projects:



These project types are available:

- General Purpose
- LCD Plus
- LCDX
- XBotX Scout
- Serial LCD2x16

The general purpose project creates a new project with a single module, which can be an existing file or a new file. If a new file is generated, it contains a skeleton main procedure that looks like this:

```
Option Explicit  
  
Public Sub Main()  
  
End Sub
```

The other choices (LCD Plus, LCDX, XBotX Scout and Serial LCD2x16) will automatically include specialized modules to give you access to each API (Application Program Interface). The program will also generate a main program with boilerplate code to get you started.

## Edit Menu

The Edit Menu allows various editing operations, such as cut, copy, paste, search and replace. Right-clicking the mouse also displays a popup menu with several editing options.

## Compile Menu

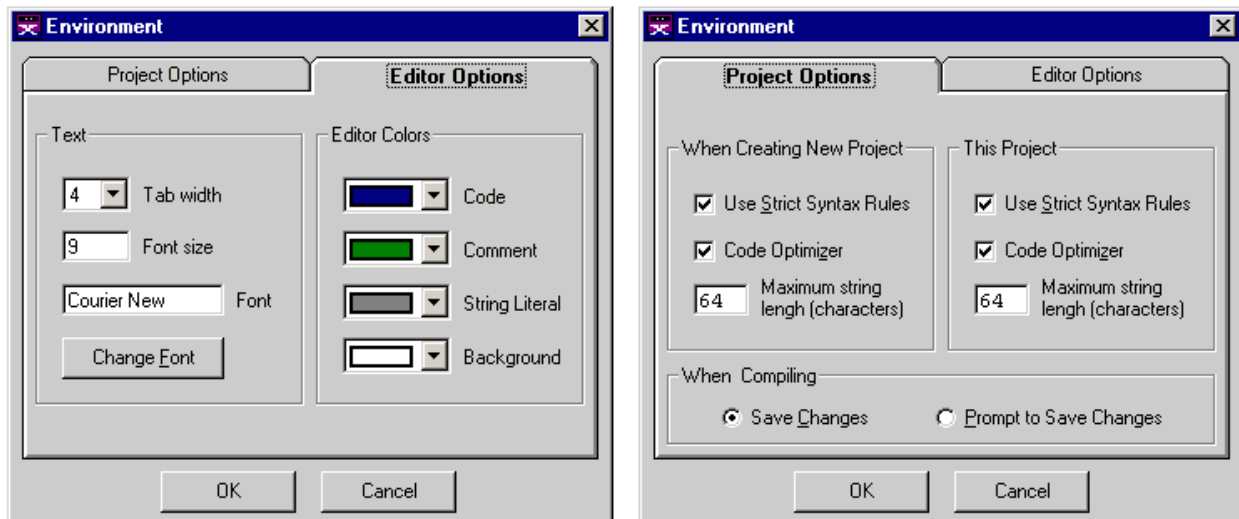
The Compile Menu has options for compiling and running the project. *Compile* creates a BXB executable file. *Compile and Run* creates the file, downloads it to the BasicX and starts the program running. *Compile Status* displays the status of the last compile, including compiler error messages, if any.

## Options Menu

In the Options menu, the *Processor Type* choice allows you to select which BasicX system type to use. Choices are BX-01, BX-24 and BX-35.

The *Environment* choice opens a window (below) that allows you to control various environment settings.

Various editor settings are available in the *Editor Options* tab, including tab width, font size and font type. Syntax highlighting is controlled by user-selectable colors for code, comments and string literals.



## Environment Window

In the *Project Options* tab, you can specify that changes are automatically written to disk whenever you compile a program. Or you can have the compiler ask whether you want to save changes.

You can also specify the following 3 compiler settings:

### ***Use Strict Syntax rules***

This checkbox enforces various coding restrictions intended to make code easier to read and understand. The restrictions affect for-next loop counters, numeric literals and bitwise logical operations. See the language reference manual for more details.

### ***Code Optimizer***

This checkbox allows you to control whether the compiler optimizes the program. Optimization generally reduces the size of a program.

### ***Maximum String Length***

This textbox allows you to specify the maximum allowable length of both variable-length and fixed-length strings. The selectable range is 1 to 64 characters, with a default of 20 characters.

Since all variable-length strings require the same amount of storage, you can reduce overall storage requirements by selecting an optimum value for the string size.

## **Project Menu**

In the Project Menu, *Chip* opens the Chip Dialog Box with options to make common setup routines easier, such as defining the initial states of I/O pins. The *Files* choice opens the Project Dialog Box, which allows you to add or delete modules in the project. *Watch Window* allows you to watch variables in a running BX-24 or BX-35 program.

*Create VB Project* allows you to create a Visual Basic® project from an existing BasicX project. If you have a VB6 compiler (not included), this option allows you to compile and run a BasicX program in a VB environment. For more details, refer to the VB Project Generator section.

## **Window Menu**

The Window menu allows you to specify how nested windows are displayed in the editor. Choices are *Cascade*, *Tile Horizontal* and *Tile Vertical*.

## Edit Window

The Edit Window is where the code for the module files are created and edited. At the bottom of the edit window is the Status Bar:



### Status Bar

The Status Bar indicates the status of various elements of the system. Panel 1 indicates the line and column number of the text cursor. Panel 2 shows whether the text cursor is in insert (INS) or overlay (OVR) mode.

Panel 3 indicates whether errors or warnings were generated by the compiler. Panel 4 shows whether or not the compile was successful.

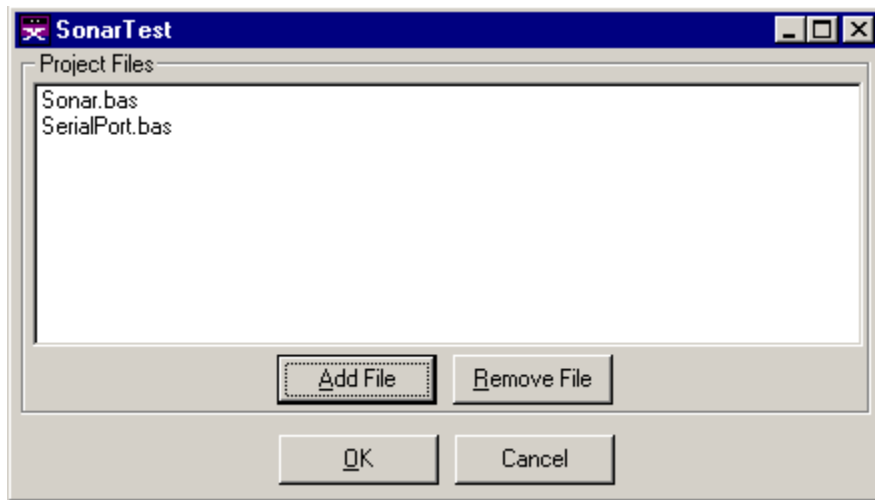
If the compile was successful, panel 4 displays the amount of EEPROM memory used by code, the RAM allocated to static variables, and the nonvolatile memory allocated to persistent variables (abbreviated PRS). If the compile was not successful, a fatal error message appears in this panel.

The above information can also be viewed in a separate window by clicking on the Status Bar or choosing Compile Status under the Compile Menu. You can also see more details of memory allocation by looking at the MPP map file, which is created whenever you compile a program.

**Warning** – the RAM memory in the Status Bar applies only to memory allocated to static variables. This is not the same as the total amount of RAM required by a program. In particular, task stacks need additional RAM, which is not displayed here.

# Project Dialog Box

The Project Dialog Box keeps track of source code that makes up the project. Source code is stored in one or more module files, each of which has a default BAS extension. The collection of modules is compiled into the final BXB executable file.



**Project Dialog Window**

## **Title Bar**

The Title Bar displays the BXP filename of the currently-open project.

## **Project Files**

The Project Files textbox lists all files in the project. Each file contains source code for a single module. You can select which files to include by pressing the Add/Remove Files buttons.

## **Add File**

The Add File button opens a dialog box where module files can be added to the project. You can add existing files or create new files.

To create a new file, you need only specify a file that does not exist, and the program will create the file.

## **Remove File**

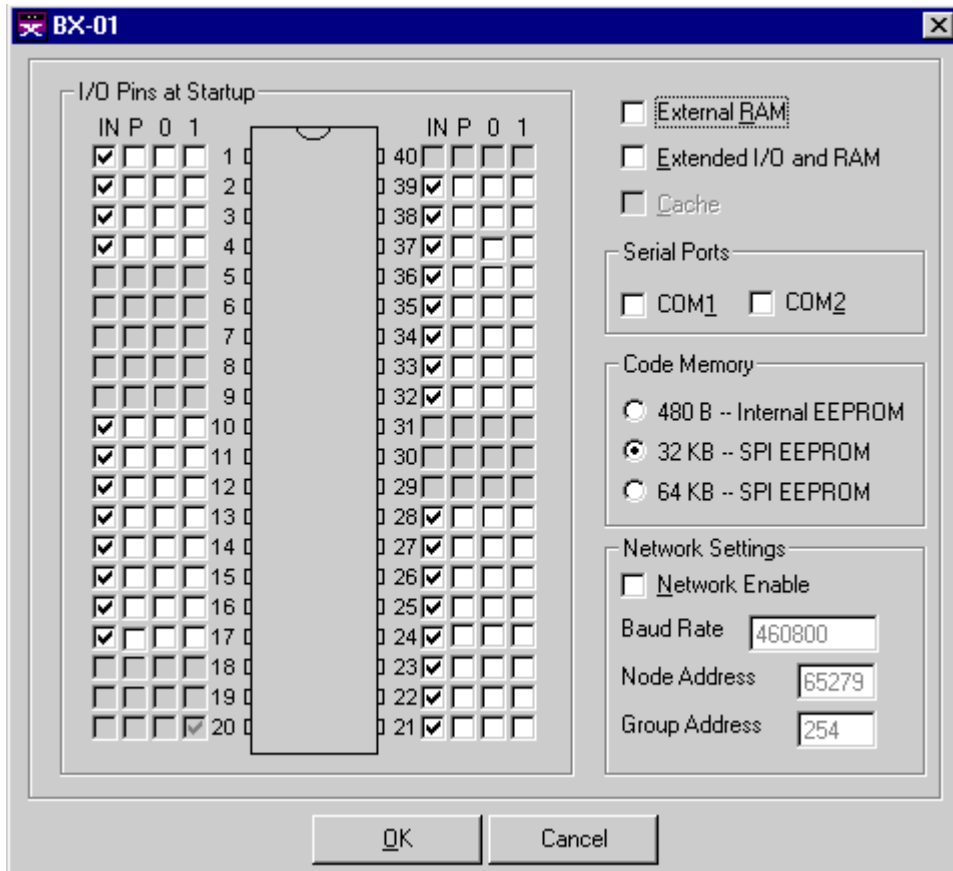
The Remove File button removes the highlighted file from the project.

## **OK**

The OK button exits after saving any changes made within the Project Dialog Box.

# BX-01 Chip Dialog Box

The Chip Dialog Box allows you to control BasicX chip options such as external RAM, Com1/Com2 serial ports, code memory and extended I/O. You can set network options such as baud rate and node/group address. You can also control the initial state of each I/O pin (note that some of these options can also be controlled by the BasicX program itself).



## External RAM

The External RAM option configures BasicX to utilize the additional RAM of the RAMSandwich™. External RAM requires pins 2, 16, 17, 21-28, and 32-39.

## Cache

The Cache option can be enabled when External RAM is utilized. When checked, it loads your program into memory where it will run faster.

## Com1

The Com1 option configures BasicX to utilize the built-in Com1 device. This disables the network. Com1 requires pins 10 and 11.

## Com2

The Com2 option configures BasicX to utilize the built-in Com2 device, which requires pins 1 and 12.

## Code Memory

This section allows you to specify where the program code is stored. Code can be stored in 480 bytes of EEPROM memory internal to the CPU chip, or code can be stored in an external SPI EEPROM, which can be either 32 KB or 64 KB.

Note that using internal EEPROM frees pins 6, 7 and 8.

The code memory choice defines the maximum allowable program space available in the system. The compiler checks to make sure your program fits in this memory.

**Warning** – system library procedure PutEEPROM will work only with a 32 KB EEPROM chip. PutEEPROM does not work for internal EEPROM or external 64 KB EEPROM.

## Network Enable

The Network Enable option configures BasicX to utilize the built-in network feature. The network requires the COM1 device and pins 10, 11, and 14.

## Baud Rate

The Baud Rate sets the network communication rate. The maximum (and default) is 460 800 baud.

## Node Address

The Node Address sets the network address node for the processor. Each chip on a particular network must have a unique address. Valid node addresses are in range 0 to 65 279.

## Group Address

This option allows you to define the group address of the chip. Multiple nodes on a network are allowed to share a common group address, and *groupcasting* allows you to send a network packet simultaneously to all members of a particular group. Valid group addresses are in range 0 to 254.

## OK Button

The OK Button exits after saving any changes made within the Chip Dialog Box.

## Cancel Button

The Cancel Button exits and disregards any changes made within the Chip Dialog Box.

## I/O Pin Grid

The I/O Pin Grid allows you to define the initial state of each available I/O pin whenever the processor is rebooted. Each pin can be set to one of four states -- tristate, pullup, output-high and output-low:

**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.

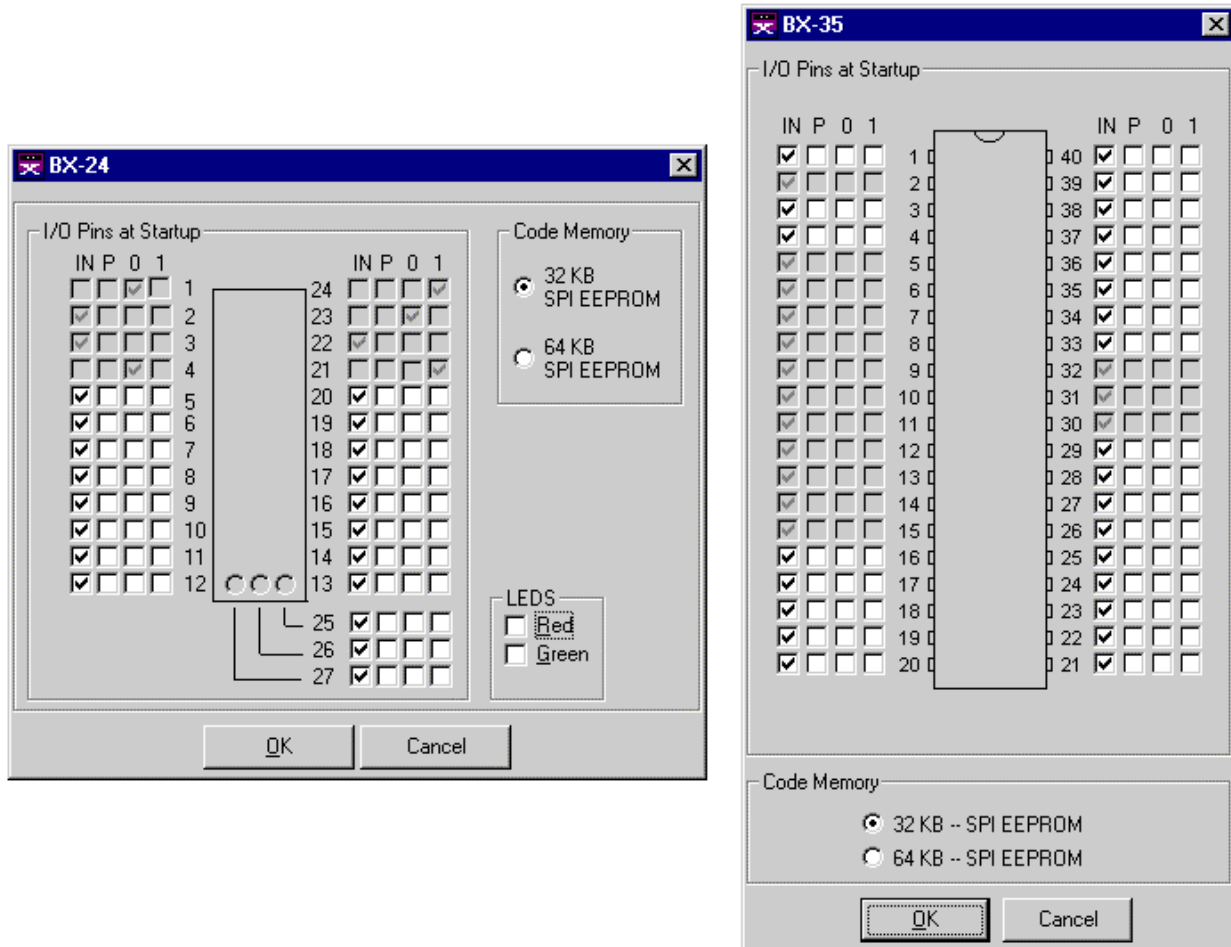
The I/O Pin Grid defines only the initial states of each pin. After startup, you can also have the program configure each pin independently using procedure PutPin or other system calls.

## Known Bugs

If you enable the cache option, the program will occasionally fail to start after download or reset. A workaround is to disable this option.

## BX-24 and BX-35 Chip Dialog Boxes

The BX-24 and BX-35 Chip Dialog Boxes allow you to control the initial state of the I/O pins and on-board LEDs on a BX-24. These options can also be controlled by the BasicX program.



### I/O Pin Grid

The I/O Pin Grid allows you to define the initial state of each available I/O pin whenever the processor is rebooted. Each pin can be set to one of four states -- tristate, pullup, output-high and output-low:

- 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.

The I/O Pin Grid defines only the initial states of each pin. After startup, you can also have the program configure each pin independently using procedure PutPin or other system calls.

### **Code Memory**

This section allows you to specify where the program code is stored. You can choose an SPI EEPROM chip with either 32 KB or 64 KB of memory.

The code memory choice defines the maximum allowable program space available in the system. The compiler checks to make sure your program fits in this memory.

**Warning** – system library procedure PutEEPROM will work only with the 32 KB EEPROM chip. PutEEPROM does not work with a 64 KB EEPROM chip.

### **OK Button**

The OK Button exits after saving any changes made within the Chip Dialog Box.

### **Cancel Button**

The Cancel Button exits and disregards any changes made within the Chip Dialog Box.

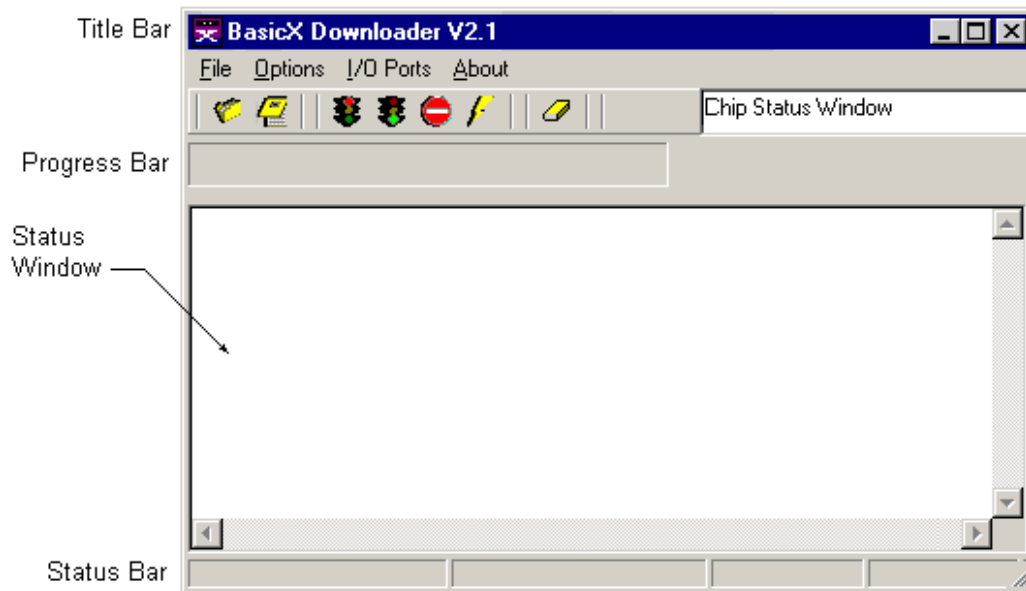
### **LEDS Box (BX-24 only)**

The LEDS box allows you to define the initial state of each of the two LEDs on the BX-24. You can also have your program control the state of each LED through pin I/O functions -- the red LED is connected to pin 25 and the green LED is connected to pin 26. To turn on an LED, you need to set the pin to logic *low*. Raising the pin turns *off* the LED.

# Downloader

The BasicX Downloader is where executable files are downloaded and run on the BasicX system. You can also use the Downloader to open the Editor/Compiler when creating or editing the BXP project or BAS source code files.

The Downloader displays the results of the download process in the Status Bar. The Status Window allows interactive two-way RS-232 communication with the BasicX system.



**Downloader Window**

## **Open Download File Button**

The Open Download File Button loads a BXB and PRF file into the BasicX. The button will ask you for a BXB filename.

## **Open Editor Button**

The Editor Button opens the BasicX Editor. This is where your project is opened, edited and compiled.

## **Stop Processor Button**

On BX-01 systems, the Stop Processor Button turns off the power derived from the parallel port. The button is useful when changing components on the Developer Board. **Warning** -- this button has no effect on power supplied externally through the Power Port on the Developer Board.

On BX-24 and BX-35 systems, the Stop Processor Button halts the processor but has no effect on power.



### **Execute Button**

The Execute Button tells the BasicX program to boot up and start running. This is indicated when the Status Bar says RUNNING. The Execute Button has the same effect as physically releasing the Reset Switch on the Developer Board.



### **Reset Processor Button**

The Reset Processor Button halts BasicX program. This is indicated when the Status Bar says RESET. The button has the same effect as physically pressing the Reset Switch on the Developer Board, but the (software) button takes precedence over the physical switch.



### **Download Program Button**

The Download Program Button loads a BXB file into the BasicX system.



### **Clear Serial Window Button**

The Clear Serial Window Button clears the contents of the Status Window.

## **Downloader Menus**

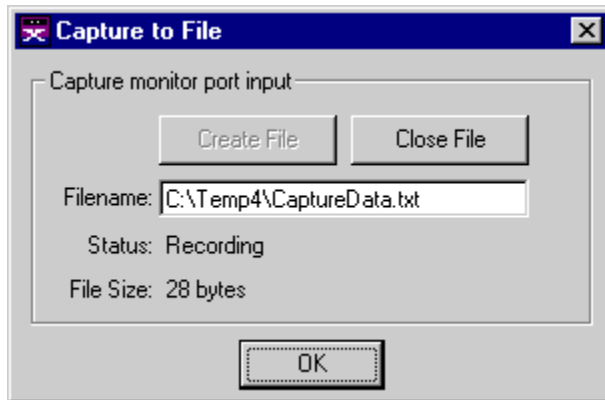
### **File Menu**

The File Menu allows you to open BXB and PRF files, or exit the BasicX Development Environment. The BXB files are BasicX binary executables created by the compiler, and PRF files contain additional information needed by the BXB file.

*Open Editor* opens the BasicX editor and compiler.

*Open Download* opens user-specified BXB and PRF files. After you open the files, you use the Download button to load the program into the development system.

*Set Starting Directory* allows you to specify the directory where the program initially looks for project files to open.



### Capture File Window

The *Capture to File* menu choice allows you to record data received by the monitor port on the PC. The data is stored in a user-specified file, which has a length limited only by available disk space. Recording continues until you close the file or exit the program.

### Processor Menu

The *Processor Type* menu allows you to select which BasicX system type to use. Choices are BX-01, BX-24 and BX-35. You can also download programs and control the processor with *Download Program*, *Stop Processor*, *Execute* and *Reset Processor* choices.

### I/O Ports Menu

The *I/O Ports* menu allows you to configure communication ports. *Download Port* configures the port used for downloading programs to the BasicX. The download port may be either a parallel or serial port, depending on the BasicX system being used.

If a serial port is used for downloading, you select only the com port number. Other settings, such as baud rate and parity, are fixed and cannot be changed by the user.

*Monitor Port* configures the serial port that is connected to the status window. Here you can specify the PC's com port number, baud rate, parity, number of data bits and number of stop bits. Note that the monitor port is bidirectional -- data received from a BasicX is displayed in the status window, and data you type into the status window is transmitted to the BasicX.

Note that for systems that use serial downloading, the monitor port can be the same as the download port. The program switches between the 2 modes automatically.

The *Rescue* function (BX-24 and BX-35 only) forces the processor to accept a new program. *ATN Diagnostic* is used for troubleshooting (see page 26 for more detail).

The *Clear Window* clears the contents of the Status Window.

### Progress Bar

The Progress Bar displays the percentage of completion when downloading your program into the BasicX system.

## **Status Window**

The Status Window displays serial data received from the BasicX system as received by the monitor port on the PC. This window also accepts input from the PC keyboard for serial transmission to the BasicX.

**Warning** -- the status window is intended for relatively simple communications with BasicX. The window does not incorporate any handshaking protocol when it communicates with a BasicX serial port. In practice, this is not usually a problem unless the BasicX is sending data at a high rate and the PC is loaded down with other processing tasks.

In some cases the status window is able to detect communication errors, such as overflow of its input buffer. If an error is detected, the message "Serial Data Error Detected" is displayed to the right of the Progress Bar.

## **Status Bar**

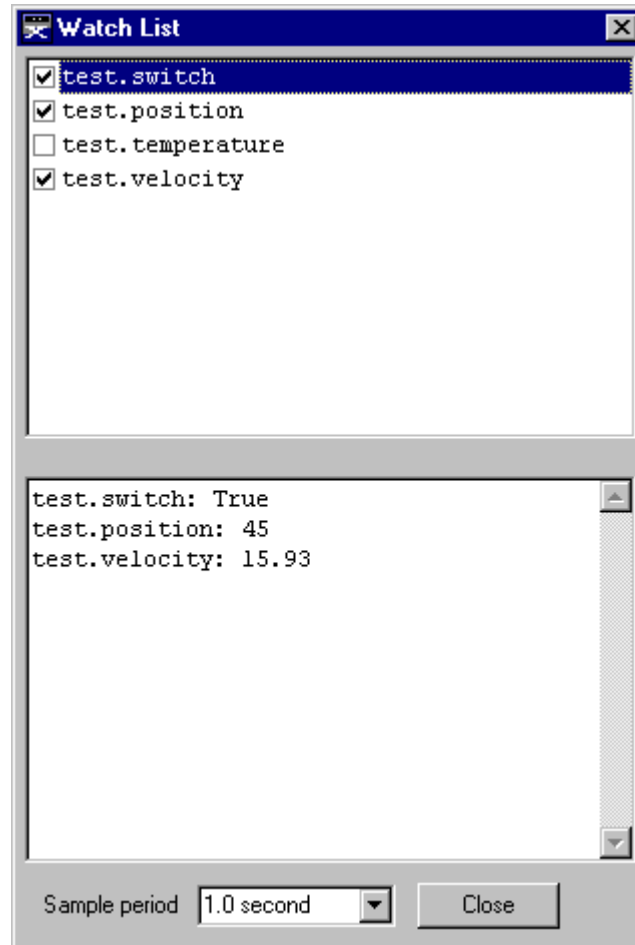
The Status Bar indicates the status of various elements of the system. These include power (on/off), processor state (running/reset) and program size.

## **Title Bar**

The Title Bar indicates the BasicX Development System version number.

## Watch Window (BX-24 and BX-35 only)

The Watch Window allows you to watch variables in a running BasicX program. The Watch Window can handle static (module-level) variables in RAM memory. Variables can be numeric or Boolean types. If an array is chosen, the first element of the array is displayed:



**Watch Window**

The Watch Window is chosen from the Project menu in the editor.

The upper window lists RAM-based module-level variables in the program. The checkboxes allow you to select which variables to watch. The lower window displays the current values of the selected variables.

The Watch Window periodically polls the BasicX processor to update each variable. The combo box at the bottom of the window allows you to tailor the sample rate by selecting from sample periods ranging from 1 second to 5 seconds.

The Watch Window reads the BXM file, which is generated whenever a program is compiled. See the next section for details about the contents of the BXM file.

---

**Warning** -- the Watch Window requires the BasicX processor's Com1 serial port in order to communicate. The program should not use Com1 while the Watch Window is active, and the monitor port should be closed.

Also, anything that interferes with serial communications, such as turning off interrupts, will interfere with the Watch Window.

The following variables and objects are not displayed in the watch window:

- Local variables
- Persistent variables
- String variables
- Register objects
- Block data objects

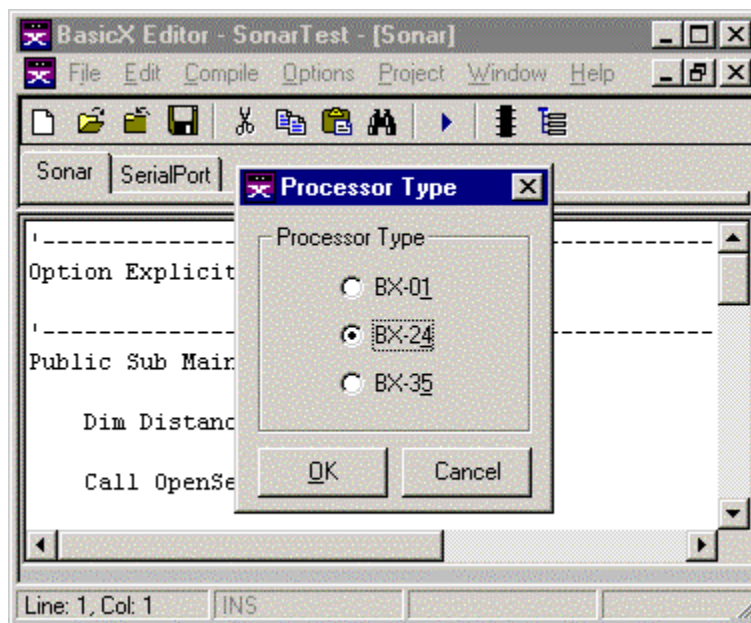
# Tutorial on using the Watch Window

## Procedure

To illustrate the Watch Window, we'll use the following program as an example. The objective is to display the value of static variable *i* while the program is running:

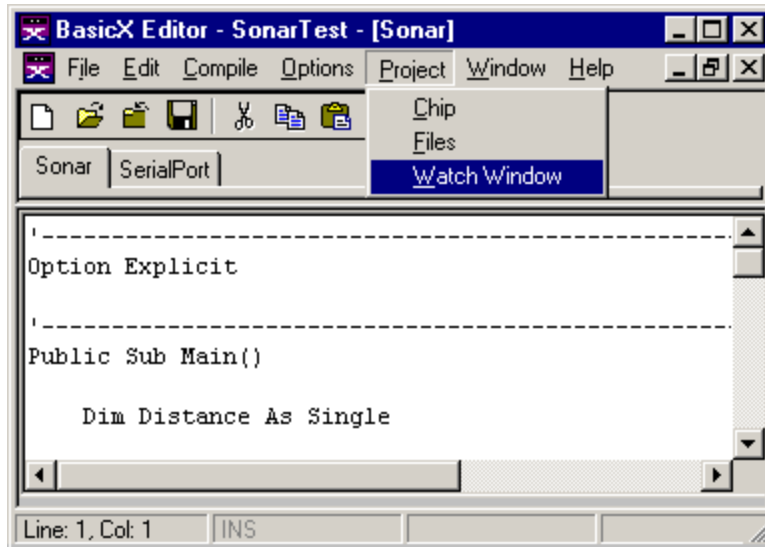
```
Dim i As Integer
Sub Main()
    i = 1
    Do
        i = i + 1
        If (i > 5) Then
            i = 1
        End If
        Call Delay(2.0)
    Loop
End Sub
```

To start the Watch window, the first step is to open the editor and make sure BX-24 or BX-35 is selected in the Options-Processor menu:



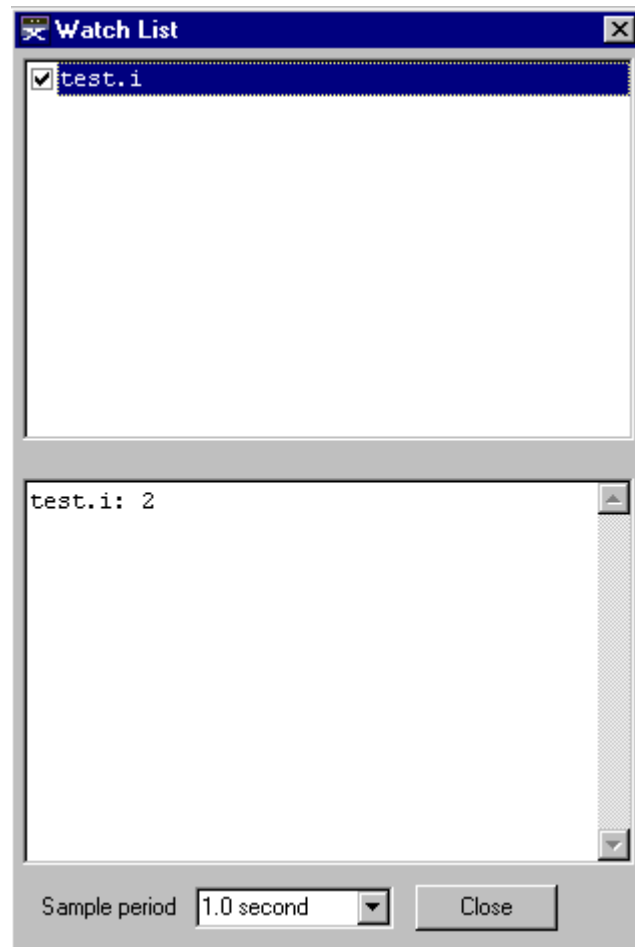
Processor Type window

The next step is to compile and download your program as usual. Once the program starts running, select the Project-Watch Window menu:



**Project-Watch Window menu**

The Watch Window should appear. In the top window, you can choose one or more module-level variables to watch, and the compiler will dynamically update the values of each variable on the bottom window. Note that the variable type is displayed next to each identifier in the top window:



**Watch Window**

In this case, the variable *test.i* is updated about once per second. The update rate can be chosen from the combo box at the lower edge of the window.

# File Types

The compiler/downloader reads and generates the following files:

<b>File Extension</b>	<b>Description</b>
BAS	Module file -- contains source code for the program.
BXP	Project file -- lists all modules in program.
PRF	Preferences file -- contains initialization and configuration information.
BXB	BasicX program in binary format -- similar to EXE file on a PC.
MPP	Map file of entire program, in human-readable format.
BXM	Map file of static module-level variables, in machine-readable format.
MPX	Map file of module level variables, in source code format for network.
OBJ	Object file -- contains object code for each subprogram
ERR	Error log file, produced by command line mode

## **BAS file**

This file contains the source code for a single module. Each module must be in a separate file, with one and only one module per file. File extensions can be arbitrary, with BAS as the default.

The module name is taken from the filename without the extension. The module name must be a legal Basic identifier, which means the name must start with a letter, and all other characters must be letters, digits or underscores.

## **BXP file**

This is the project file, which contains a list of the filenames of all modules in the program. Typical format:

```
C:\Dir1\MainModule.bas
C:\Dir3\SubDir\Irrigation.bas
SerialPort.bas
```

Filenames are prefixed by optional pathnames. If a pathname is absent, the compiler assumes the file is located in the same subdirectory as the BXP file.

## **PRF file**

This is the preferences file, which contains information required by the BasicX system. Information includes the initial states of I/O pins, as well as memory configuration data.

The combination of PRF and BXB file is required whenever you download a program to a BasicX system.

## **BXB file**

The BXB file contains the executable code that is run by the BasicX chip. This file, combined with the PRF file, is similar to the EXE file on a PC.



Decoded values:

```
    ModuleName: "serialport"  
    VariableName: "inputbuffer"  
    MemoryLocation: 400  
        DataType: Byte (18)  
    IsPersistent: False  
    Dimensions: 1  
    LowerBound(1): 1  
    UpperBound(1): 32  
    LowerBound(2..8): 0  
    UpperBound(2..8): 0
```

### MPX file

This file contains information about module-level variables in source code format. The primary function of the MPX file is to provide information to a remote node on the network in order for the remote node to get read/write access to static variables on the local node.

In the above serial port example, the MPX file contains the following lines pertaining to array InputBuffer:

```
' serialport_inputbuffer data type: Byte  
Public Const serialport_inputbuffer As Integer = 400 ' &H190  
Public Const serialport_inputbuffer_IsPersistent As Boolean = False
```

You can include the MPX file directly as a module in the remote program. Whenever you change the local program, the MPX file is updated automatically when you compile the program, which means you can recompile the remote program to update it.

**Note** -- the address of each variable has a numerical range of 0 to 65 535. However, the MPX file uses 16-bit signed integer types in which to store these numbers, and addresses between 32 768 and 65 535 are actually represented as negative numbers in two's complement format. The internal bit patterns correspond to the actual addresses.

As an example, the apparent address -1 corresponds to real address 65 535 (&HFFFF).

### OBJ file

These files contain the compiled object code for each subprogram. The files are written to the <Install>\Temp subdirectory, where <Install> is where the BasicX.exe program is installed.

Whenever a BasicX program is finished being compiled, the OBJ files are no longer needed. You can erase all the files in the Temp subdirectory whenever the compiler is done.

**Warning** – you should not run 2 or more copies of the compiler at the same time. This will cause the compilers to interfere with each other, since all object files are written to a common directory.

## Command Line Operation

The compiler and downloader may be invoked from a command line. This is useful when you're using a third-party editor for writing your BasicX code. The syntax is as follows:

**BasicX Project /c /d**

**BasicX** - invokes the BasicX.exe file. If nothing follows this parameter, the BasicX Environment will start without a project. The pathname may also be required if the current directory is other than the installation directory.

**Project** (optional) -- the name of your .BXP project file without the .BXP extension. You must set the starting directory to the location of this file. If nothing follows this parameter, the BasicX Environment will start with the specified project loaded into the editor.

**/c** (optional) -- switch for compiling the project. This will create the .BXB file for your project. The BasicX Environment itself will not start when this switch is used.

**/d** (optional) -- switch for downloading the project. This switch will download the .BXB file to your chip. The .BXB file must exist already if this switch is used without /c. The BasicX Environment itself will not start when this switch is used.

The command line is not case sensitive. Each argument must be separated by 1 or more spaces or tabs.

The project must be completely created as usual within the BasicX Environment. Once the source filenames and chip preferences have been established and saved in the project, you may then edit your source files in another ASCII editor and use the command line to compile and download the project. The starting directory, processor type (BX-01, BX-24, BX-35, etc.) and download port must be properly set for the current project.

A message file named *BasicX.err* is created in the starting directory when you use the command line option. The file contains the processing results. If errors are encountered, the error number and description will be given along with any compile time errors, generally including the line number and filename of the source file causing the error.

**Example** -- in the following example, file DemoProgram.bxp is located in the starting directory:

```
C:\Program Files\BasicX\>BasicX DemoProgram /c /d
```

This line causes the program to be compiled and downloaded. Note that BasicX.exe is assumed to be located in subdirectory \Program Files\BasicX in this example. This location can vary, depending on where the program is installed.

# Troubleshooting Serial Downloads

**Symptom** – the error message “Unable to halt BasicX” appears when you try to download a new program, or when you use the IDE to start and stop the processor (BX-24 or BX-35 only).

**Action** -- press and release the hardware reset button while the downloader is going through its retry cycles. This is when "Retry *N*" appears in the chip status window.

Here's what's happening -- in order to download a new program, the PC first needs to halt the BasicX chip. The PC does this by rapidly toggling the ATN line, which is pin 4 of the DB-9 connector.

Sometimes a legal BasicX program can interfere with this process. Example code:

```
Sub Main()  
  Register.TCCR0 = 0  
End Sub
```

This program shuts down the real time clock, which has the side effect of preventing the processor from recognizing the ATN pin.

To get around this problem, the BasicX Operating System allows a brief time period after a hardware reset during which time the chip listens to the ATN line. The user-written program doesn't start until the end of this period.

The idea behind pressing the reset line is to get the BasicX chip to reboot during the time the PC is toggling the ATN line.

## Rescue Function

If the above procedure doesn't work, another alternative is to use the *Rescue* function. Here the PC continuously toggles the ATN line during a period of about 10 seconds, during which you can press the hardware reset line. Here is the procedure:

- (1) Click on the Rescue menu.
- (2) Within the next 10 seconds, press and release the hardware reset button on the BasicX.
- (3) After the 10 second period is up, the system should be ready for a new download.

Again, the idea is to get the BasicX chip to reboot during the time the PC is toggling the ATN line.

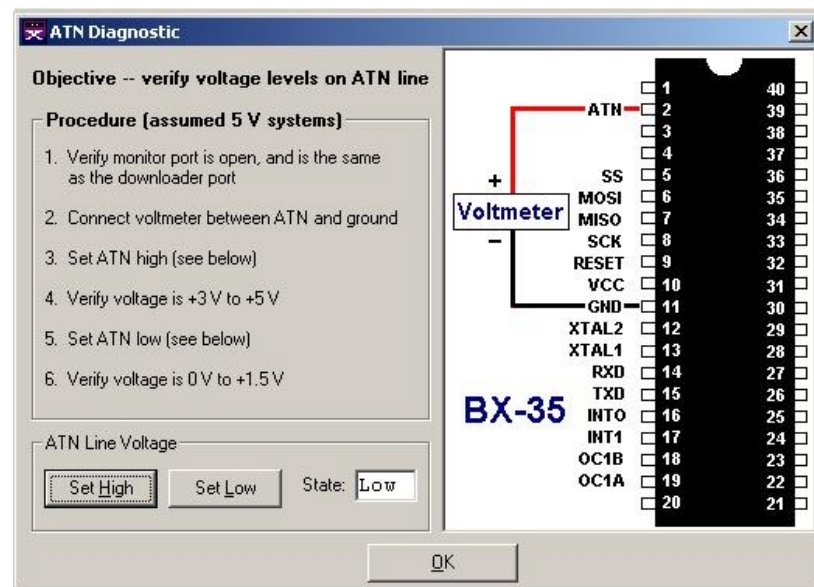
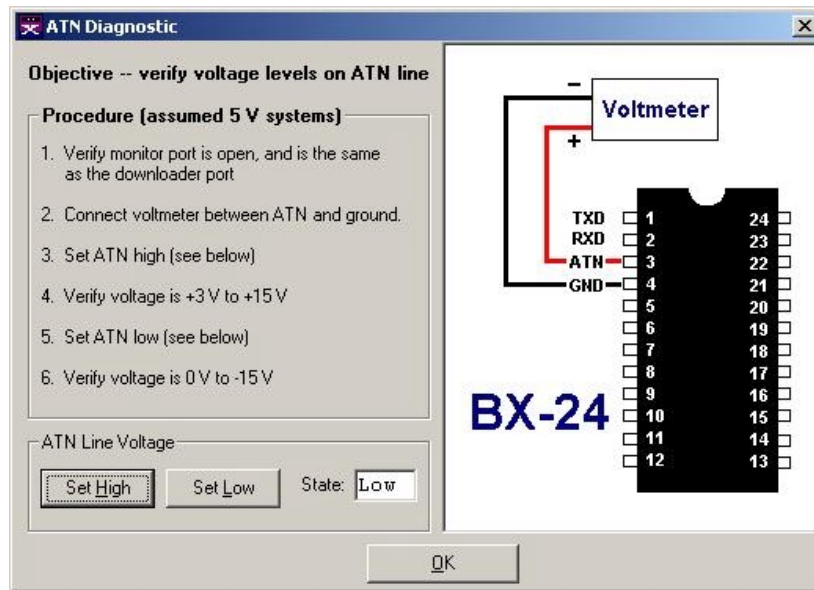
If the PC is still unable to halt the processor, you may want to verify that the ATN line is functioning correctly. See the following section for ATN diagnostics.

# ATN Diagnostic Windows

The ATN diagnostic window gives you a simple way of testing the ATN line. The ATN line is pin 4 of the DB-9 connector, and is used for serial downloading of new programs for BX-24 and BX-35 systems.

Whenever a new program is downloaded, the PC first needs to get the attention of the BasicX processor. This is done by rapidly toggling the ATN line. The ATN diagnostic window lets you manually control the line level, which makes it easy to use a voltmeter to determine that the connection and voltage levels are acceptable.

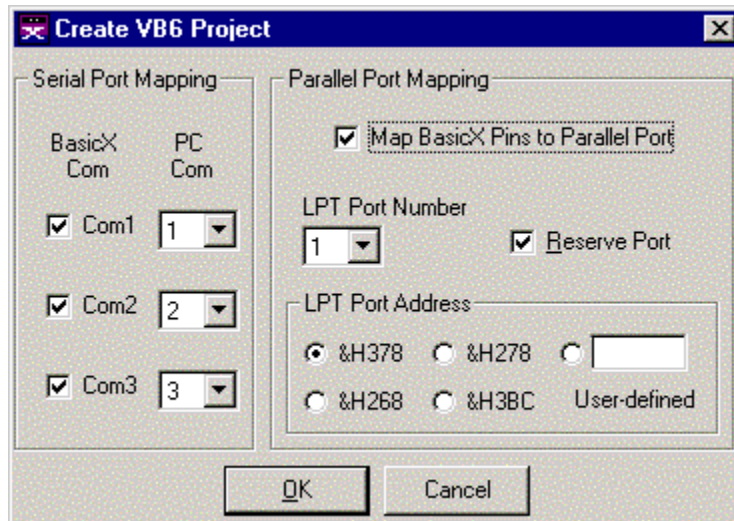
The diagnostic windows are shown below for BX-24 and BX-35 systems:



## VB Project Generator

The BasicX IDE allows you to create a Visual Basic® project from an existing BasicX project. If you have a VB6 compiler (not included), this option makes it easy to use a VB environment to compile and run a BasicX program on a PC. Since most of the BasicX language is subset-compatible with VB, this can be helpful in developing and debugging a BasicX program.

To generate a VB project, the first step is to use the BasicX editor to open an existing BasicX project. Next, go to the *Project / Create VB Project* menu. This window appears:



This window allows you to map BasicX I/O to PC I/O. In particular you can map BasicX serial ports to PC serial ports, and BasicX pin numbers to PC parallel port pin numbers.

When you click the OK button, a VBP project file is created, which includes all BasicX modules listed in the currently-open BXP file, as well as various VB6 support files that are required for BasicX emulation. The VBP file has the same name as the BXP file and is in the same directory.

The subdirectory VB\_Support is also created in this directory. The subdirectory typically contains the following OS emulation and support files:

- bxBase2Constants.bas
- bxBOSAPI.bas
- bxMain.frm
- bxRegisterBX24.cls            Depends on BasicX system type
- bxStubs.bas
- bxUtilities.bas
- UnsignedInteger.cls
- UnsignedLong.cls

The VB program starts from the form file bxMain.frm, which includes setup and initialization code that defines the I/O map. In addition, the file BasicXAPI.dll is required by the emulator, and if you use the parallel port, one or more parallel port drivers are used, depending on the version of Windows you're using. These files are installed in appropriate Windows OS subdirectories as required.

## Operating the BasicX emulator

The following notes apply to BasicX OS emulation in a VB environment:

1. A high performance timer/counter is required for proper operation. The WIN32API functions QueryPerformanceCounter and QueryPerformanceFrequency can be used to determine the presence of the timer.
2. The emulator uses the PC's built in clock to simulate the BasicX real time clock (RTC), but the 1.95 ms granularity of the RTC is not reproduced.
3. If you map BasicX I/O pins to the PC parallel port, all pin directions are fixed as input-only or output-only. DB-25 pin directions are shown below:

<u>Pin</u>	<u>Input</u>	<u>Output</u>	<u>Remarks</u>
1		x	OutputCapture pin
2		x	
3		x	
4		x	
5		x	
6		x	
7		x	
8		x	
9		x	
10	x		InputCapture pin
11	x		
12	x		
13	x		
14		x	
15	x		
16		x	
17		x	

The emulator will ignore attempts to configure output pins to input modes, or input pins to output modes.

4. Several pins in the parallel port are inverted. This is handled automatically by the emulator. As an example, when you write code to raise an output pin, if the pin is inverted, the emulator reverses the logic in order to physically raise the pin. Similar logic applies to code that reads input pins.
5. If you try to add data to a full queue, or extract data from an empty queue, the emulator will raise an exception. This behavior is different from a BasicX system, which simply blocks on either of the two events, and resumes as soon as the queues are ready.

In the VB environment, this can affect serial I/O in particular, and you should avoid trying to add data to a full serial queue, or extract data from an empty serial queue, unless you use the timeout version of GetQueue.

6. For serial output queues, the emulator contains an additional internal output buffer that is 1024 bytes long. As soon as you write data to a queue, the data is immediately copied to the internal buffer. This can alter the timing of code that depends on waiting until a queue is empty, or code that would otherwise block on a full output queue.

7. For serial input queues, the emulator contains an additional internal input buffer that is 512 bytes long. This can alter the behavior of code that may be sensitive to overruns of input data.
8. Several data types have different sizes in VB compared to BasicX. Examples are Boolean, UnsignedInteger and enumeration types. This can affect lower level code that depends on data type sizes, such as BlockMove, RAMPeek, RAMPoke and queue operations.  
  
GetQueue and PutQueue in particular will handle only Byte types, which can be scalars or arrays. Any other data types will raise an exception.
9. The Register class is stubbed. You are allowed to read and write to registers, but any behavior beyond that is not implemented. For example, setting or clearing an interrupt bit in Register.SREG has no effect on interrupts.
10. BasicX tends to be more permissive than VB regarding project and module names. Unlike BasicX, VB does not allow a module to have the same name as the project to which it belongs.
11. Timing in Windows is less deterministic than a BasicX system, which should be taken into account when you call procedures such as PulseIn and PulseOut.
12. For additional information about running BasicX programs in a VB environment, refer to the “Portability Issues” section in the *Basic Express Language Reference*.
13. Not all BasicX OS functions are emulated. See the following section for more details.

# BasicX Operating System Functions

Of approximately 120 functions provided by the BasicX language and operating system, 27 are built into VB6, 43 are emulated and 50 are stubbed.

Three BasicX classes are not implemented (block data, bounded string and persistent variable classes), and some kinds of enumeration type conversions need special handling (see below).

## Functions built into VB6

Abs	Absolute value	
Asc	Returns the ASCII code of a character	
Atn	Arc tangent	
CBool	Convert Byte to Boolean	BX-24, BX-35 only
CByte	Convert to Byte	
CInt	Convert to Integer	
CLng	Convert to Long	
CSng	Convert to floating point (single)	
CStr	Convert to string	
Chr	Converts a numeric value to a character	
Cos	Cosine	
Debug.Print	Sends string to Com1 serial port (BasicX) or debug window (VB)	
Exp	Raises e to a specified power	
Fix	Truncates a floating point value	
LCase	Converts string to lower case	
Len	Returns the length of a string	
Log	Natural log	
Mid	Copies a substring	
Randomize	Sets the seed for the random number generator	
Rnd	Generates a random number	
Sin	Sine	
Sqr	Square root	
Tan	Tangent	
Timer	Returns floating point seconds since midnight	
Trim	Trims leading and trailing blanks from string	
UCase	Converts string to upper case	
VarPtr	Returns the address of a variable or array	

## Emulated functions

ACos	Arc cosine	
ASin	Arc sine	
BlockMove	Copies a block of data from one RAM location to another	
ClearQueue	Clears data from queue	
CloseCom	Closes serial port	
CountTransitions	Counts the logic transitions on an input pin	BX-24, BX-35 only
CuInt	Convert to UnsignedInteger	
CuLng	Convert to UnsignedLong	
DefineCom3	Defines parameters for serial I/O on arbitrary pin	BX-24, BX-35 only
Delay	Pauses task and allows other tasks to run	
Exp10	Raises 10 to a specified power	
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	
FlipBits	Generates mirror image of bit pattern	BX-24, BX-35 only
Fmt	Converts a floating point value to a formatted string	
GetDate	Returns the date	
GetDayOfWeek	Returns the day of week	
GetPin	Returns the logic level of an input pin	
GetQueue	Reads data from a queue	
GetQueueBufferSize	Returns the queue's buffer size	
GetQueueCount	Returns the number of valid bytes in the queue	
GetTime	Returns the time of day	
GetTimestamp	Returns the date and time of day	
InputCapture	Records a pulse train on the input capture pin	
Log10	Log base 10	
OpenCom	Opens an RS-232 serial port	
OpenQueue	Defines an array as a queue	
OutputCapture	Sends a pulse train to the output capture pin	
PeekQueue	Looks at queue data without removing any data	
Pow	Raises an operand to a given power	
PulseIn	Measures pulse width on an input pin	
PulseOut	Sends a pulse to an output pin	
PutPin	Configures a pin to 1 of 4 input or output states	
PutQueue	Writes data to a queue	
PutQueueStr	Writes a string to a queue	
RAMPeek	Reads a byte from RAM	
RAMPoke	Writes a byte to RAM	
Sleep	Pauses task and allows other tasks to run	
StatusQueue	Determines if a queue has data available for reading	
ValueS	Convert a string to a float (single) type	

### Stubbed functions

ADCToCom1	Streams data from ADC to serial port	BX-24, BX-35 only
CPUSleep	Puts the processor in various low-power modes	
CallTask	Starts a task	
Com1ToDAC	Streams data from serial port to DAC	BX-24, BX-35 only
DACPin	Generates a pseudo-analog voltage at an output pin	
DelayUntilClockTick	Pauses task until the next tick of the real time clock	
FirstTime	Determines whether the program has ever been run since download	
FreqOut	Generates dual sinewaves on output pin	BX-24, BX-35 only
Get1Wire	Receives data bit using Dallas 1-Wire protocol	BX-24, BX-35 only
GetADC	Returns analog voltage	BX-24, BX-35 only
GetBit	Reads a single bit from a variable	BX-24, BX-35 only
GetEEPROM	Reads data from EEPROM	
GetNetwork	Reads data from a remote RAM location	BX-01 only
GetNetworkP	Reads data from a remote EEPROM location	BX-01 only
GetXIO	Reads data from extended I/O	BX-01 only
GetXRAM	Reads data from XRAM	BX-01 only
LockTask	Locks the task and discourages other tasks from running	
MemAddress	Returns the address of a variable or array	
MemAddressU	Returns the address of a variable or array	
OpenNetwork	Opens the network	BX-01 only
OpenSPI	Opens SPI communications	
OpenWatchdog	Starts the watchdog timer	
PersistentPeek	Reads a byte from EEPROM	

PersistentPoke	Writes a byte to EEPROM	
PlaySound	Plays sound from sampled data stored in EEPROM	BX-24 only
Put1Wire	Transmits data bit using Dallas 1-Wire protocol	BX-24, BX-35 only
PutBit	Writes a single bit to a variable	BX-24, BX-35 only
PutDAC	Generates a pseudo-analog voltage at an output pin	
PutDate	Sets the date	
PutEEPROM	Writes data to EEPROM	
PutNetwork	Sends data to a remote RAM location	BX-01 only
PutNetworkP	Sends data to a remote EEPROM location	BX-01 only
PutNetworkPacket	Sends a special packet over the network	BX-01 only
PutNetworkQueue	Sends data to a remote queue	BX-01 only
PutTime	Sets the time of day	
PutTimestamp	Sets the date, day of week and time of day	
PutXIO	Writes data to extended I/O	BX-01 only
PutXRAM	Writes data to XRAM	BX-01 only
RCTime	Measures the time delay until a pin transition occurs	
ResetProcessor	Resets and reboots the processor	
SPICmd	SPI communications	
Semaphore	Coordinates the sharing of data between tasks	
SerialNumber	Returns the version number of a BasicX chip	
ShiftIn	Shifts bits from an I/O pin into a byte variable	BX-24, BX-35 only
ShiftOut	Shifts bits out of a byte variable to an I/O pin	BX-24, BX-35 only
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	
X10Cmd	Transmits X-10 data	BX-24, BX-35 only

### Not implemented

Block data classes  
Bounded string classes  
Persistent data classes

### Enumeration type conversions

In both BasicX and VB, if you want to convert an enumeration type to a numeric type, you can use the usual type conversion functions CByte, CInt, CLng and similar functions.

For type conversions in the opposite direction -- numeric to enumeration -- BasicX provides the CType function, and also provides functions of the form ToEnum, where Enum is the type name.

CType is not compatible with VB6 (it is compatible with VB.NET), and you would need to write your own versions of ToEnum for the VB6 environment. The emulator provides a function called CEnum to assist with this. As an example, if you have an enumeration type called ColorType, a type conversion function would look like this:

```
Public Function ToColorType( _
    ByVal Value As Variant) As Byte

    ToColorType = CEnum(Value)

End Function
```

# Index

ATN diagnostic window 30  
ATN line toggling 30  
ATN line, serial downloads 29  
BAS file 25  
BX-01 chip dialog box 11  
    Baud rate 12  
    Cache 11  
    Code memory 12  
    Com1 12  
    Com2 12  
    External RAM 11  
    Group address 12  
    I/O pin grid 13  
    Network enable 12  
    Node address 12  
BX-24 chip dialog box 14  
    Code memory 15  
    I/O pin grid 14  
    LEDs box 15  
BX-35 chip dialog box 14  
    Code memory 15  
    I/O pin grid 14  
BXB file 25  
BXM file 26  
BXP file 25  
BasicX emulator, operation 32  
BasicXAPI.dll 31  
CType 36  
Chip dialog box 11  
    BX-01 11  
    BX-24 14  
    BX-35 14  
Code optimizer 8  
Command line operation 28  
Compiler 5  
Create VB project 8  
Data type enumerations, BXM file 26  
Downloader 16  
Downloader buttons 16  
    Clear serial window 17  
    Download program 17  
    Execute 17  
    Open download file 16  
    Open editor 16  
    Reset processor 17  
    Stop processor 16

Downloader menus 17  
   Capture to file 18  
   File 17  
   I/O ports 18  
   Monitor port 18  
   Open download 17  
   Open editor 17  
   Processor 18  
   Rescue 18  
   Set starting directory 17  
   Status bar 19  
   Status window, downloader 19  
 ERR file 28  
 Edit window 9  
 Editor 5  
 Editor menus 6  
   Compile 7  
   Edit 7  
   Editor options 7  
   Environment 7  
   File 6  
   New project 6  
   Options 7  
   Project options 7  
 Enumeration type conversions, BasicX emulator 36  
 File types 25  
 High performance timer/counter 32  
 Inverted pins, parallel port 32  
 MPP file 26  
 MPX file 27  
 Maximum string length 8  
 New project menu 6  
   General purpose 6  
   LCD Plus 6  
   LCDX 6  
   Serial LCD2x16 6  
   XBotX Scout 6  
 OBJ file 27  
 PRF file 25  
 Parallel port, BasicX emulator 32  
 Project dialog box 10  
   Add file 10  
   Project files 10  
   Remove file 10  
 RTC, BasicX emulator 32  
 Real time clock, BasicX emulator 32  
 Real time clock, BasicX emulator 32  
 Rescue function 29  
 Status bar, editor window 9  
 Strict syntax rules 8  
 ToEnum 36

---

Troubleshooting serial downloads 29  
VB project generator 31  
Watch window 20