Basic Express System Library

Version 2.0

2.00H

System library

The BasicX operating system provides a library of system calls in the following categories:

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
Randomize	Sets the seed for the random number generator
Rnd	Generates a random number
Sin	Sine
Sqr	Square root
Tan	Tangent

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

Memory-related functions

BlockMove	Copies a block of data from one RAM location to another	
FlipBits	Generates mirror image of bit pattern	BX-24, BX-35 only
GetBit	Reads a single bit from a variable	BX-24, BX-35 only
GetEEPROM	Reads data from EEPROM	
GetXIO	Reads data from extended I/O	BX-01 only
GetXRAM	Reads data from XRAM	BX-01 only
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	BX-24, BX-35 only
PutEEPROM	Writes data to EEPROM	
PutXIO	Writes data to extended I/O	BX-01 only
PutXRAM	Writes data to XRAM	BX-01 only
RAMPeek	Reads a byte from RAM	
RAMPoke	Writes a byte to RAM	
SerialNumber	Returns the version number of a BasicX chip	

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

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

Type conversions

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	
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	
ValueS	Convert a string to a float (single) type	

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	

Pin I/O

ADCToCom1	Streams data from ADC to serial port	BX-24, BX-35 only
Com1ToDAC	Streams data from serial port to DAC	BX-24, BX-35 only
CountTransitions	Counts the logic transitions on an input pin	BX-24, BX-35 only
DACPin	Generates a pseudo-analog voltage at an output pin	
FreqOut	Generates dual sinewaves on output pin	BX-24, BX-35 only
GetADC	Returns analog voltage	BX-24, BX-35 only
GetPin	Returns the logic level of an input pin	
InputCapture	Records a pulse train on the input capture pin	
OutputCapture	Sends a pulse train to the output capture pin	
PlaySound	Plays sound from sampled data stored in EEPROM	BX-24, BX-35 only
PulseIn	Measures pulse width on an input pin	
PulseOut	Sends a pulse to an output pin	
PutDAC	Generates a pseudo-analog voltage at an output pin	
PutPin	Configures a pin to 1 of 4 input or output states	
RCTime	Measures the time delay until a pin transition occurs	
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

Communications

Debug.Print	Sends string to Com1 serial port	
DefineCom3	Defines parameters for serial I/O on arbitrary pin	BX-24, BX-35 only
Get1Wire	Receives data bit using Dallas 1-Wire protocol	BX-24, BX-35 only
OpenCom	Opens an RS-232 serial port	
OpenSPI	Opens SPI communications	
Put1Wire	Transmits data bit using Dallas 1-Wire protocol	BX-24, BX-35 only
SPICmd	SPI communications	
X10Cmd	Transmits X-10 data	BX-24, BX-35 only

Network (BX-01 only)

GetNetwork	Reads data from a remote RAM location	BX-01 only
GetNetworkP	Reads data from a remote EEPROM location	BX-01 only
OpenNetwork	Opens the network	BX-01 only
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

The following BX-24 system calls require BX-24 chip version 2.1 or above:

- ADCToCom1
- CBool
- Com1ToDAC
- FlipBits
- GetBit
- PutBit
- ShiftIn
- ShiftOut

The BasicX chip version can be determined by using procedure SerialNumber on all BasicX systems. On BX-24 systems, version 2.1 can be visually identified by a yellow dot on the SPI EEPROM chip.

Abs function

Syntax

F = Abs(*Operand*)

Arguments

Item	Type	Direction	Description
Operand	Any numeric type	Input	Operand
F	Same as operand	Output	Function return

Description

Returns the absolute value of the operand.

Example

```
Dim X As Single
Dim I As Integer

X = Abs(-5.3) ' X is 5.3

I = Abs(-47)  ' I is 47
```

Known Bugs

The Abs function may generate erroneous type mismatch error messages in expressions of the following types:

Long
UnsignedLong
UnsignedInteger

ACos function

Syntax

F = ACos(*Operand*)

Arguments

Item	Type	Direction	Description
Operand	Single	Input	Operand
F	Single	Output	Function return

Description

Calculates the arc cosine. The function return is in units of radians.

Example

```
Dim F As Single

F = ACos(0.707107) ' F is Pi/4 radians (45 Degrees)
```

ADCToCom1 procedure

Syntax

BX-24, BX-35 Only

Call ADCToCom1(*PinNumber, DataRate*)

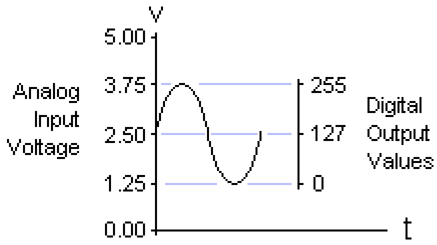
Arguments

Item	Type	Direction	Description
PinNumber	Byte	Input	ADC pin number. Range is 13 to 20.
DataRate	Integer	Input	Data rate. Units are samples per second, at 1 byte per sample. Range is 113 to 11 000.

Description

ADCToCom1 reads the ADC (Analog to Digital Converter) and streams data out the Com1 serial port at *DataRate* samples per second. The baud rate is constant at 115 200 baud (procedure OpenCom is not required since this procedure takes over Com1). To stop the stream, call the procedure using 0 as *PinNumber*.

Analog input: AC signal centered at 2.5 V with maximum range of 2.5 +/- 1.25 V, as shown to the right.



Digital output: stream of bytes in range 0 to 255 and scaled approximately such that a 1.25 V input generates a 0 output, and 3.75 V input generates a 255 output.

Warning

No other ADC readings should be made while ADCToCom1 is active. Also, this procedure uses Timer1, which means it would conflict with anything else that depends on Timer1, such as InputCapture and OutputCapture.

Example

```
' Read ADC pin 16, send to Com1 at 5000 sample/s.
Call ADCToCom1(16, 5000)

' Stop the stream after 1 second.
Call Delay(1.0)
Call ADCToCom1(0, 5000)
```

Asc function

Syntax

F = Asc(*Source*)

Arguments

Item	Type	Direction	Description
Source	String	Input	String source.
F	Byte	Output	ASCII code of the first character of *Source*.

Description

Returns the ASCII code of the first character of a string.

Example

```
Dim Tx As String * 3, Code As Byte
Tx = "ABC"
Code = Asc(Tx) ' Code is 65 (ASCII "A")
```

ASin function

Syntax

F = ASin(*Operand)*

Arguments

Item	Type	Direction	Description
Operand	Single	Input	Operand
F	Single	Output	Function return

Description

Calculates arc sine. The function return is in units of radians

Example

```
Dim F As Single

F = ASin(1.0) ' F is Pi/2 radians (90 Degrees)
```

Atn function

Syntax

F = Atn(*Operand)*

Arguments

Item	Type	Direction	Description
Operand	Single	Input	Operand
F	Single	Output	Function return

Description

Calculates arc tangent. The function return is in units of radians.

Example

```
Dim Y As Single

F = Atn(1.0) ' F is Pi/4 radians (45 degrees).
```

BlockMove procedure

Syntax

Call BlockMove(*NumberOfBytes, SourceAddress, DestinationAddress*)

Arguments

Item	Type	Direction	Description
NumberOfBytes	UnsignedInteger	Input	Number of bytes to copy. Legal range is 1 to 65 535.
SourceAddress	UnsignedInteger	Input	Address of source.
DestinationAddress	UnsignedInteger	Input	Address of destination.

Description

Copies a block of memory from source to destination in RAM. BlockMove can copy an arbitrarily large block of memory in a single operation, and the block is allowed to span multiple variables in memory.

Example

```
Sub Main()

    Dim UI As New UnsignedInteger
    Dim B(1 To 2) As Byte

    UI = &h9ABC&

    ' Copy the 16-bit unsigned integer variable to the
    ' two byte array.
    Call BlockMove( 2, MemAddress(UI), MemAddress(B) )

    ' At this point, B(1) is BCh and B(2) is 9Ah (note
    ' that B(2) is the most significant byte).

End Sub
```

CallTask procedure

Syntax

CallTask *"TaskName", TaskStack*

Arguments

Item	Type	Direction	Description
TaskName	Task name	Input	Name of procedure to be used as a task. The name must be in quotes.
TaskStack	Byte array	Input/Output	Stack memory to be used by task – must be a module-level byte array

Description

CallTask starts a procedure as a parallel running task. The *TaskName* procedure is just like any other procedure except it's not allowed to have formal parameters.

The task must be allocated memory to be used as workspace. That's what *TaskStack* is for. This array is used as memory workspace for processing expressions, math functions, calling subprograms, etc.

Multiple tasks can be run at the same time, up to the limit of available stack memory. Each task is executed on a sequential basis. Multiple copies of the same task can be run at the same time using different stacks.

Tasks can start other tasks, and a task can call other subprograms. With the exception of the main program, whenever a task exits, either through an End Sub statement or an Exit Sub statement, the task is completed and ceases to run. The stack used by the task is then free to be used by another task.

The main program is an exception. It never terminates as long as the processor keeps running.

Warning

If a task has insufficient stack space, it will cause the whole BasicX chip to become unreliable. It is better to err on the side of too much stack space.

Note that the *TaskStack* array requires 15 bytes of overhead for its internal task frame. This means the array needs to be at least 15 bytes long, plus enough room for the actual task stack.

Example

See CallTaskExample.bas example file in the BX01_Docs\Examples\Doc_Examples subdirectory.

CBool function

Syntax

BX-24, BX-35 Only

F = CBool(*Operand*)

Arguments

Item	Type	Direction	Description
Operand	Byte	Input	Operand
F	Boolean	Output	Function return

Description

Converts a Byte type to a Boolean type.

If the operand is zero, the function returns False. If the operand is nonzero, the function returns True.

Example

```
Dim B As Boolean

B = CBool(255) ' B is True.

B = CBool(127) ' B is True.

B = CBool(0)   ' B is False.
```

CByte function

Syntax

F = CByte(*Operand*)

Arguments

Item	Type	Direction	Description
Operand	Any numeric	Input	Operand
F	Byte	Output	Function return

Description

Converts any numeric type to Byte type.

Example

```
Dim X As Single
Dim B As Byte

X = 2.4
B = CByte(X) ' B is 2
```

Chr function

Syntax

F = Chr(*Code*)

Arguments

Item	Type	Direction	Description
Code	Byte	Input	ASCII code of character.
F	String	Output	Character in string.

Description

Converts an ASCII code to a character in a string. If the destination string is larger than one character, the string is left justified and blank filled.

Example

```
Dim Tx as String * 1
Tx = Chr(65) ' Tx is "A" (ASCII 65).
```

CInt function

Syntax

F = CInt(*Operand*)

Arguments

Item	Type	Direction	Description
Operand	Any numeric	Input	Operand
F	Integer	Output	Function return

Description

Converts any numeric type to Integer type.

Example

```
Dim X As Single
Dim Y As Integer

X = 1.5
Y = CInt(X) ' Y is 2
```

CLng function

Syntax

F = CLng(*Operand*)

Arguments

Item	Type	Direction	Description
Operand	Any numeric	Input	Operand
F	Long	Output	Function return

Description

Converts any numeric type to Long type.

Example

```
Dim X As Single
Dim L as Long

X = 1.5
L = CLng(X) ' L = 2
```

Com1ToDAC procedure

Syntax

BX-24, BX-35 Only

Call Com1ToDAC(*PinNumber*)

Arguments

Item	Type	Direction	Description
PinNumber	Byte	Input	I/O pin number.

Description

This procedure streams data from the Com1 serial port to a DAC on *PinNumber* output pin. The data source should be a remote BasicX system running procedure ADCToCom1.

As bytes arrive at the serial port, the processor will change the value of the DAC to correspond. The Com1 baud rate is automatically set to 115 200 baud (it is not necessary to call OpenCom). The DAC is updated at a constant 10 000 updates per second, and the output voltage range is 0 V to 5 V (on 5 V systems).

There are 2 ways to terminate Com1ToDAC. First, you can call the procedure with a *PinNumber* of 0. Second, the remote system can terminate its ADCToCom1.

Example

```
' Stream data from Com1 to a DAC on pin 17.
Call Com1ToDAC(17)

' Turn off the stream after 1.5 seconds.
Call Delay(1.5)
Call Com1ToDAC(0)
```

Cos function

Syntax

F = Cos(*Operand)*

Arguments

Item	Type	Direction	Description
Operand	Single	Input	Operand
F	Single	Output	Function return

Description

Calculates cosine. The operand is in units of radians

Example

```
Dim F As Single

' Cos(Pi/4)
F = Cos(0.785398) ' F is 0.707 107
```

CountTransitions function (float version)

Syntax

BX-24, BX-35 Only

F = CountTransitions(*PinNumber, TimeInterval*)

Arguments

Item	Type	Direction	Description
PinNumber	Byte	Input	Pin number.
TimeInterval	Single	Input	Time interval over which to count. Units are in seconds. Range is about 2.441 µs to 4800.0 s. Resolution is about 2.441 µs.
F	Long	Output	Number of transitions.

Description

This function counts the number of logic transitions that occur during the specified time interval. Both rising edges and falling edges are counted as transitions. The maximum sample rate is 409 600 sample/s.

Counting starts as soon as the function is called. If no transitions occur within the specified time interval, the function returns 0.

Warning

This procedure halts all multitasking for the duration of the call. The real time clock (RTC), task switching and network traffic are suspended during this time. If *TimeInterval* is comparable in size or greater than the RTC tick period (about 1.95 milliseconds), the RTC will lose time.

CountTransitions function (integer version)

Syntax

BX-24, BX-35 Only

F = CountTransitions(*PinNumber, TimeInterval*)

Arguments

Item	Type	Direction	Description
PinNumber	Byte	Input	Pin number.
TimeInterval	Long	Input	Time interval over which to count. Units are (1 / 409 600) s (about 2.441 µs).
F	Long	Output	Number of transitions.

Description

This function counts the number of logic transitions that occur during the specified time interval. Both rising edges and falling edges are counted as transitions. The maximum sample rate is 409 600 sample/s.

Counting starts as soon as the function is called. If no transitions occur within the specified time interval, the function returns 0.

Warning

This procedure halts all multitasking for the duration of the call. The real time clock (RTC), task switching and network traffic are suspended during this time. If *TimeInterval* is comparable in size or greater than the RTC tick period (about 1.95 milliseconds), the RTC will lose time.

CPUSleep procedure

Syntax

Call CPUSleep( )

Arguments

None.

Description

This procedure causes the processor to execute a special machine-language SLEEP instruction, which can put the processor in various low power modes depending on how internal registers are configured. Refer to Atmel documentation on the details of using the SLEEP instruction.

CStr function

Syntax

F = CStr(*Operand*)

Arguments

Item	Type	Direction	Description
Operand	Boolean or numeric	Input	Operand
F	String	Output	Function return

Description

Converts Boolean and numeric types to String type.

Example

```
Dim Tx As String, B As Boolean

Tx = "V = " & CStr(-193) & " m/s"   ' Tx is "V = -193 m/s"

B = True
Tx = "State = " & CStr(B)           ' Tx is "State = True"
```

CSng function

Syntax

F = CSng(*Operand*)

Arguments

Item	Type	Direction	Description
Operand	Any numeric	Input	Operand
F	Single	Output	Function return

Description

Converts any numeric type to Single type.

Example

```
Dim Y As Single
Dim I As Integer

I = 3
Y = CSng(I) ' Y is 3.0
```

CuInt function

Syntax

F = CuInt(*Operand*)

Arguments

Item	Type	Direction	Description
Operand	Any discrete numeric	Input	Operand
F	UnsignedInteger	Output	Function return

Description

Converts any discrete (non-float) numeric type to UnsignedInteger type.

Example

```
Dim L As Long
Dim U As New UnsignedInteger

L = 65535
U = CuInt(L)  ' U is 65 535
```

CuLng function

Syntax

F = CuLng(*Operand*)

Arguments

Item	Type	Direction	Description
Operand	Any discrete numeric	Input	Operand
F	UnsignedLong	Output	Function return

Description

Converts any discrete (non-float) numeric type to UnsignedLong type.

Example

```
Dim U As New UnsignedLong
Dim B As Byte

B = 255
U = CuLng(B) ' Type conversion -- U is now 255
```

DACPin procedure

Syntax

Call DACPin(*Pin, DACvalue, DACcounter*)

Arguments

Item	Type	Direction	Description
Pin	Byte	Input	Pin number.
DACvalue	Byte	Input	Voltage output, range 0 to 255 units. Unit conversions: 0 = 0 volts 255 = 5 volts (on 5 V systems) **Example:** Converting volts to DACvalue 1.6 volts = 1.6 V * 256 / 5.0V = DACvalue = 82 **Example:** Converting DACvalue to volts 167 counts = 167 * 5.0 V / 256 = 3.26 volts
DACcounter	Byte	Input/Output	DACcounter is a value that must be returned each time the routine is called so that the DAC remains in sync. If you have multiple DACs running concurrently, then you must have a different DACcounter for every pin.

Description

DACPin generates an 8-bit pseudo-analog voltage on an I/O pin. On 5 volt systems, the voltage range is 0.0 V to 5.0 V, with a resolution of about 19.6 mV.

A rapid set of pulses is precisely timed to produce the desired voltage. A simple low pass filter circuit is needed externally to filter the output. PutDAC produces this "blast" of pulses for a short time, then places the pin in a high impedance state before returning.

The external filter circuit is relied upon to maintain the voltage between calls. DACPin should be called periodically to refresh the pin and keep the voltage within tolerances. The optimum refresh rate depends on the characteristics of the circuit to which the pin is connected. You might consider calling DACPin in a separate task if you need to refresh the pin continuously.

See **PutDAC** for the floating point equivalent of DACPin.

Warning

DACPin turns the selected pin into an output pin independent of any other setting. Also, if the output pin is not refreshed periodically, the analog output voltage will not be maintained

Example

```
Dim DACcounter As Byte

' Set pin 17 to 3.26 volts = (167 * 5.0V / 256)
Call DACPin(17, 167, DACcounter)
```

Debug.Print method

Syntax

Debug.Print *Operand*$_1$ [; *Operand*$_2$; ... *Operand*$_N$] [;]

Arguments

Item	Type	Direction	Description
Operand$_N$	String	Input	Operand

Description

Debug.Print sends one or more strings out the Com1 serial port at 19 200 baud. Multiple string parameters must be separated by semicolons.

A carriage-return/linefeed pair is automatically appended unless an optional semicolon terminates the line. An empty Debug.Print outputs a carriage return/linefeed only.

Debug.Print automatically sets up Com1 for output. OpenCom is not needed.

Example

```
Debug.Print "Velocity = "; CStr(193);

Debug.Print "m/s"

Debug.Print ' Outputs only <CR><LF>

' Output is "Velocity = 193 m/s<CR><LF><CR><LF>
```

DefineCom3 procedure

Syntax

BX-24, BX-35 Only

Call DefineCom3(*InputPin, OutputPin, ParameterMask*)

Arguments

Item	Type	Direction	Description
InputPin	Byte	Input	Input pin number.
OutputPin	Byte	Input	Output pin number.
ParameterMask	Byte	Input	Communication parameters (see below).

Allowable values for the internal bit pattern in *ParameterMask:*

Parameter	Value	Bit pattern (x = don't care)
Inverted logic	&H80	1 0 x x x x x x
Non-inverted logic	&H00	0 0 x x x x x x
Even parity	&H30	x x 1 1 x x x x (7 bit only)
Odd parity	&H20	x x 1 0 x x x x (7 bit only)
No parity	&H00	x x 0 0 x x x x
7 data bits	&H07	x x x x 0 1 1 1
8 data bits	&H08	x x x x 1 0 0 0

Description

DefineCom3 defines parameters for serial port Com3. This procedure is used in conjunction with OpenCom to define the port, which can be routed to any pair of I/O pins. Com3 always uses 1 stop bit.

If you want to open a port with a single pin (as input-only or output-only), you can use pin 0 as one of the pin numbers. Pin 0 is treated as a dummy pin.

Warning

Parity is not supported for 8-bit data.

Example

```
' Define port 3 to use pin 16 as input, 17 as output. Also use
' inverted logic, even parity, 7 data bits. Implicit 1 stop bit.
Call DefineCom3(16, 17, bx1011_0111)

' Define baud rate and buffers for port 3.
Call OpenCom(3, 19200, InputBuffer, OutputBuffer)
```

Delay procedure

Syntax

Call Delay(*Interval*)

Arguments

Item	Type	Direction	Description
Interval	Single	Input	Delay period. Range is 0.0 to 127.0 s. Resolution is about 1.95 ms.

Description

Suspends the current task for at least the specified time interval. At the end of the delay, the task will become ready again. How soon the task actually resumes execution depends on how busy the system is with other tasks.

A delay of 0.0 is a useful way to allow other tasks to execute, while allowing immediate resumption if no other tasks are eligible to run.

The actual time delay is guaranteed to be at least the specified delay period, as long as *Interval* meets range constraints. A negative delay is treated as equivalent to a delay of 0.0.

If the task is locked, Delay will unlock the task (see procedure LockTask).

Example

```
' Set pin 16 high
Call PutPin(16, bxOutputHigh)

' Pause this task for a minimum of 1/2 s, then wake up
Call Delay(0.5)

'Set pin 16 low
Call PutPin(16, bxOutputLow)
```

DelayUntilClockTick procedure

Syntax

Call DelayUntilClockTick

Arguments

None.

Description

Suspends the current task until the next tick of the real time clock (RTC). Other tasks are allowed to run during the suspension. How soon the task actually resumes execution depends on how busy the system is with other tasks.

If the task is locked, this procedure will unlock the task (see procedure LockTask).

Example

```
' Toggle pin 17 at the RTC tick rate.
Do
    Call DelayUntilClockTick
    Call PutPin(17, 1)
    Call DelayUntilClockTick
    Call PutPin(17, 0)
Loop
```

Exp function

Syntax

F = Exp(*Operand*)

Arguments

Item	Type	Direction	Description
Operand	Single	Input	Operand
F	Single	Output	Function return

Description

Raises *e* to the power specified by the operand. The constant e (natural logarithm base) is approximately 2.718 282.

Example

```
Dim F As Single

F = Exp(1.0) ' F is equal to "e"
```

Exp10 function

Syntax

F = Exp10(*Operand)*

Arguments

Item	Type	Direction	Description
Operand	Single	Input	Operand
F	Single	Output	Function return

Description

Raises 10 to the power specified by the operand.

Example

```
Dim Y As Single

Y = Exp10(3.0) ' Y is 1000.0
```

FirstTime function

Syntax

F = FirstTime( )

Arguments

Item	Type	Direction	Description
F	Boolean	Output	Whether the function has ever been called since the program was download.

Description

Returns a Boolean value that indicates whether this is the first time the function has been called since the program was downloaded.

FirstTime is useful if you want a program to behave differently the first time it is run. For example, you may want to set persistent variables to initial values that apply only when the program is first executed. Whenever the processor reboots, you can avoid re-initializing those variables, or you can set them to other values.

This is what happens behind the scenes -- whenever a program is downloaded, a special variable in nonvolatile EEPROM memory is set to a nonzero value. When you call FirstTime, the function looks at the variable. If it's nonzero, the variable is cleared and the function returns *true*. Otherwise the function returns *false*. Any subsequent calls to FirstTime will return *false*, even after the system reboots.

Example

```
Dim Setpoint As New PersistentSingle

Sub Initialize()

    If (FirstTime) Then

        ' This is the first time the program has been run.
        ' Initialize the default setpoint value.
        Setpoint = 72.0
    End If

End Sub
```

Fix function

Syntax

F = Fix(*Operand)*

Arguments

Item	Type	Direction	Description
Operand	Single	Input	Operand
F	Single	Output	Function return

Description

Truncates a floating point value without changing the data type. Truncation is toward 0.0.

Example

```
Dim Y1 As Single
Dim Y2 As Single

Y1 = Fix(1.1)   ' Y1 is 1.0

Y2 = Fix(-4.9) ' Y2 is -4.0
```

FixB function

Syntax

F = FixB(*Operand)*

Arguments

Item	Type	Direction	Description
Operand	Single	Input	Operand
F	Byte	Output	Function return

Description

Truncates the floating point operand and converts the result to Byte type. Truncation is toward 0.

Example

```
Dim B1 As Byte
Dim B2 As Byte

B1 = FixB(6.4)   ' B1 is 6

B2 = FixB(-9.8) ' B2 is -9
```

FixI function

Syntax

F = FixI(*Operand)*

Arguments

Item	Type	Direction	Description
Operand	Single	Input	Operand
F	Integer	Output	Function return

Description

Truncates the floating point operand and converts the result to Integer type. Truncation is toward 0.

Example

```
Dim I As Integer

I = FixI(-1.5) ' I is -1
```

FixL function

Syntax

F = FixL(*Operand)*

Arguments

Item	Type	Direction	Description
Operand	Single	Input	Operand
F	Long	Output	Function return

Description

Truncates the floating point operand and converts the result to Long type. Truncation is toward 0.

Example

```
Dim L As Long

L = FixL(12.9) ' L is 12
```

FixUI function

Syntax

F = FixUI(*Operand)*

Arguments

Item	Type	Direction	Description
Operand	Single	Input	Operand
F	UnsignedInteger	Output	Function return

Description

Truncates the floating point operand and converts the result to UnsignedInteger type. Truncation is toward 0.

Example

```
Dim I As New UnsignedInteger

I = FixUI(-1.5) ' I is -1
```

FixUL function

Syntax

F = FixUL(*Operand*)

Arguments

Item	Type	Direction	Description
Operand	Single	Input	Operand
F	UnsignedLong	Output	Function return

Description

Truncates the floating point operand and converts the result to UnsignedLong type. Truncation is toward 0.

Example

```
Dim L As New UnsignedLong

L = FixUL(5.9) ' L is 5
```

FlipBits function

Syntax

BX-24, BX-35 Only

F = FlipBits(*Operand*)

Arguments

Item	Type	Direction	Description
Operand	Byte	Input	Operand
F	Byte	Output	Function return

Description

FlipBits generates the mirror image of the operand's bit pattern. LSbit becomes MSbit and vice versa.

Example

```
Dim A As Byte, B As Byte

A = bx11110100

B = FlipBits(A) ' B is bx00101111.
```

FreqOut procedure (float version)

Syntax

BX-24, BX-35 Only

Call FreqOut(*Pin, Freq1, Freq2, Duration*)

Arguments

Item	Type	Direction	Description
Pin	Byte	Input	Output pin number.
Freq1	Integer	Input	Frequency 1. Units are in Hz.
Freq2	Integer	Input	Frequency 2. Units are in Hz.
Duration	Single	Input	Duration of signal. Units are in seconds. Range is about 1.0 ms to 2.56 s.

Description

Generates an analog signal that consists of two superimposed sine waves. The signal is generated for the specified duration, where the time units are in seconds.

Warning

This procedure halts all multitasking for the duration of the call. The real time clock (RTC), task switching and network traffic are suspended during this time. If *Duration* is greater than 1.95 milliseconds, the RTC will lose time.

FreqOut procedure (integer version)

Syntax

BX-24, BX-35 Only

Call FreqOut(*Pin, Freq1, Freq2, Duration*)

Arguments

Item	Type	Direction	Description
Pin	Byte	Input	Output pin number.
Freq1	Integer	Input	Frequency 1. Units are in Hz.
Freq2	Integer	Input	Frequency 2. Units are in Hz.
Duration	Integer	Input	Duration of signal. Units are in ms. Range is 1 ms to 2560 ms.

Description

Generates an analog signal that consists of two sine waves. The signal is generated for the specified duration, where the time units are in milliseconds.

Warning

This procedure halts all multitasking for the duration of the call. The real time clock (RTC), task switching and network traffic are suspended during this time. If *Duration* is greater than 1 unit, the RTC will lose time.

Get1Wire function

Syntax

BX-24, BX-35 Only

F = Get1Wire(*PinNumber*)

Arguments

Item	Type	Direction	Description
PinNumber	Byte	Input	Pin number.
F	Byte	Output	Bit value. Range is 0 to 1.

Description

Receives a single bit using the Dallas 1-Wire protocol. The bit is input on the specified pin number.

GetADC procedure (float version)

Syntax

BX-24, BX-35 Only

Call GetADC(*PinNumber, NondimVolt)*

Arguments

Item	Type	Direction	Description
PinNumber	Byte	Input	Pin number.
NondimVolt	Integer	Output	Nondimensional voltage. Range is 0.0 to 1.0. Resolution is about 0.0978 %.

Description

GetADC returns 10-bit analog voltage. The returned value is nondimensional, with a range of 0.0 to 1.0. For 5 V systems, the range corresponds to 0.0 V to 5.0 V, with a resolution of about 4.89 mV (5 / 1023).

ADC pin numbers depend on the system:

BX-24 ADC pins: 13 to 20
BX-35 ADC pins: 33 to 40

Note that GetADC automatically configures the pin for analog input. You don't need a separate call to configure the pin to input mode.

Example

```
Dim NondimVolt As Single
Const PinNumber As Byte = 13

Call GetADC(PinNumber, NondimVolt)
```

GetADC function (integer version)

Syntax

BX-24, BX-35 Only

Voltage = GetADC(*PinNumber)*

Arguments

Item	Type	Direction	Description
PinNumber	Byte	Input	Pin number.
F	Integer	Output	Voltage. Range is 0 to 1023. For 5 V systems, units are in 5/1023 volts (about 4.89 mV).

Description

GetADC returns a 10-bit analog voltage. ADC pin numbers depend on the system:

BX-24 ADC pins: 13 to 20
BX-35 ADC pins: 33 to 40

Note that GetADC automatically configures the pin for analog input. You don't need a separate call to configure the pin to input mode.

Example

```
Dim Voltage As Integer
Const PinNumber As Byte = 13

Voltage = GetADC(PinNumber)
```

GetBit function

Syntax

BX-24, BX-35 Only

F = GetBit(*Operand, BitNumber*)

Arguments

Item	Type	Direction	Description
Operand	Any variable or array	Input	Operand
BitNumber	Byte	Input	Bit number (numbering starts at 0). Range is 0 to 255.
F	Byte	Output	Function return

Description

GetBit returns the value of the specified bit. Bit numbering starts at 0. If the operand is an array, GetBit can be used to read any of the first 256 bits of the array.

Example

```
' This illustrates GetBit for a single byte.

    Dim A As Byte, B As Byte, C As Byte

    A = bx00100000

    B = GetBit(A, 5) ' B is 1.
    C = GetBit(A, 6) ' C is 0.

' This illustrates GetBit for a 32-bit Long array.

    Dim L(1 To 2) As Long

    L(1) = 0
    L(2) = 1

    B = GetBit(L, 31) ' B is 0.
    C = GetBit(L, 32) ' C is 1 (1st bit in 2nd element of array).
```

GetDate procedure

Syntax

Call GetDate(*Year, Month, Day*)

Arguments

Item	Type	Direction	Description
Year	Integer	Output	Year. Range is 1999 to 2177.
Month	Byte	Output	Month.
Day	Byte	Output	Day of month.

Description

GetDate returns the date.

GetDayOfWeek function

Syntax

F = GetDayOfWeek( )

Arguments

Item	Type	Direction	Description
F	Byte	Output	Day of week. Range is 1 to 7 (bxSunday, bxMonday .. bxSaturday).

Description

Returns the day of week. Range is bxSunday to bxSaturday.

Warning

The day of week is undefined until the calendar date is defined. See procedures PutDate or PutTimestamp to define the calendar date.

GetEEPROM procedure

Syntax

Call GetEEPROM(*Address, Value, Length*)

Arguments

Item	Type	Direction	Description
Address	Long	Input	Starting location of the source in EEPROM
Value	Any type	Input/Output	Starting location of the destination in RAM.
Length	UnsignedInteger	Input	Number of bytes to transfer from EEPROM to RAM

Description

GetEEPROM transfers data from EEPROM to RAM. The EEPROM memory is where the BasicX program is stored. Since a particular program may not use all the available memory, this procedure allows you to use leftover space for nonvolatile data storage.

GetEEPROM can transfer an arbitrarily large block of memory in a single operation, and the block is allowed to span multiple variables in RAM.

Example

```
' Each of these strings requires 22 bytes of storage
' (20 characters plus 2 bytes overhead).
Dim Name As String * 20
Dim Address As String * 20
Dim Phone As String * 20

Sub Main()

    ' Read data from the EEPROM into RAM variables.
    Call GetEEPROM(1000, Name, 22)
    Call GetEEPROM(1022, Address, 22)
    Call GetEEPROM(1044, Phone, 22)

End Sub
```

GetNetwork procedure

Syntax

BX-01 Only

Call GetNetwork(*NodeAddress, MemoryAddress, Value, Result*)

Arguments

Item	Type	Direction	Description
NodeAddress	UnsignedInteger	Input	Node address of the remote system.
MemoryAddress	UnsignedInteger	Input	RAM address of the data to be copied. See the discussion of MPX map files for more information about variable locations.
Value	Any scalar type	Output	Destination of the copy.
Result	Byte	Output	Result of the network operation. See below for allowable values.

Allowable values for *Result:*

bxNetOk	= 0	No errors
bxNetNoResponse	= 1	No response from remote system
bxNetBusy	= 255	Network command in progress

Description

GetNetwork copies a scalar variable from a remote BasicX system over the network.

The task that executes the GetNetwork procedure will suspend until the data transfer is either acknowledged by the remote system, or a number of retries has been attempted. The task is then awakened and a result value is returned.

Known Bugs

See procedure PutNetwork.

GetNetworkP procedure

Syntax

BX-01 Only

Call GetNetworkP(*NodeAddress, MemoryAddress, Value, Result*)

Arguments

Item	Type	Direction	Description
NodeAddress	UnsignedInteger	Input	Node address of the remote system.
MemoryAddress	UnsignedInteger	Input	EEPROM (persistent) address of the data to be copied. See the discussion of MPX map files for more information about variable locations.
Value	Any scalar type	Output	Destination of the copy.
Result	Byte	Output	Result of the network operation. See below for allowable values.

Allowable values for *Result:*

bxNetOk	= 0	No errors
bxNetNoResponse	= 1	No response from remote system
bxNetBusy	= 255	Network command in progress

Description

GetNetworkP copies a persistent scalar variable from a remote BasicX system over the network.

The task that executes the GetNetworkP procedure will suspend until the data transfer is either acknowledged by the remote system, or a number of retries has been attempted. The task is then awakened and a result value is returned.

Known Bugs

See procedure PutNetwork.

GetPin function

Syntax

F = GetPin(*Pin*)

Arguments

Item	Type	Direction	Description
Pin	Byte	Input	Pin number
F	Byte	Output	Logic level (0 or 1).

Description

GetPin reads the state of an I/O pin. GetPin is typically used in conjunction with procedure PutPin, which configures the pin.

Warning

If you call GetPin without previously configuring the pin as input, results are undefined. The pin direction can be set using PutPin, or you can use the chip dialog boxes in the compiler to configure each pin.

Example

```
Dim PinLogicLevel As Byte

' Define pin 16 as input.
Call PutPin(16, bxInputPullup)

' Read the value of pin 16.
PinLogicLevel = GetPin(16)
```

Known Bugs

On the BX-01, if a pin is set to input-pullup, GetPin erroneously changes the pin to input-tristate when the function returns. **Software workaround** -- just after GetPin, add a call to PutPin in order to restore the pin state to input-pullup. **Hardware workaround** -- add an external pullup resistor to the pin.

GetQueue procedure

Syntax

Call GetQueue(*Queue, Variable, Count*)

Arguments

Item	Type	Direction	Description
Queue	Byte array	Input/Output	Queue from which data is removed.
Variable	Any type	Output	Destination of extracted data.
Count	Integer	Input	Number of bytes to be extracted.

Description

GetQueue removes data from a queue and places the data into one or more RAM variables. GetQueue can cross boundries between variables to retrieve multiple pieces of data in a single operation. Variables do not have to be the same type going in as going out (see example)

If there is nothing in the queue, GetQueue will suspend the current task until the correct amount of data is placed into the queue.

Queues are a convenient way to pass data between tasks or to store data for future processing.

Warning

If there is nothing in the queue, and no task ever places anything in the queue, this command will not return and the task will halt indefinitely.

Example

```
Dim Oven(1 To 50) As Byte
Dim Pi As Single
Dim Fridge(1 To 4) As Byte

Sub Main()

    Call OpenQueue(Oven, 50)
    Pi = 3.14159

    ' Put some Pi in the oven.
    Call PutQueue(Oven, Pi, 4)

    ' Put four byte-size pieces of Pi in the Fridge.
    Call GetQueue(Oven, Fridge, 4)

End Sub
```

GetTime procedure

Syntax

Call GetTime(*Hour, Minute, Second*)

Arguments

Item	Type	Direction	Description
Hour	Byte	Output	Hours. Range is 0 to 23.
Minute	Byte	Output	Minutes after the hour.
Second	Single	Output	Seconds. Resolution is about 1.95 ms.

Description

Returns time of day in 24-hour format. Floating point seconds are returned, with a resolution of 1 / 512 seconds (about 1.95 ms). Resolution is independent of time-of-day.

GetTimestamp procedure

Syntax

Call GetTimestamp(*Year, Month, Day, Hour, Minute, Second*)

Arguments

Item	**Type**	**Direction**	**Description**
Year	Integer	Output	Year. Range is 1999 to 2177.
Month	Byte	Output	Month.
Day	Byte	Output	Day.
Hour	Byte	Output	Hours. Range is 0 to 23.
Minute	Byte	Output	Minutes.
Second	Single	Output	Seconds.

Description

Returns the date and time of day. Time is in 24-hour format.

GetXIO function

Syntax

BX-01 only

F = GetXIO(*Address*)

Arguments

Item	Type	Direction	Description
Address	UnsignedInteger	Input	I/O address, range 607 to 65 535
F	Byte	Output	I/O port value

Description

GetXIO receives data from an eXtended I/O port. BasicX supports up to 65 536 of these I/O ports.

Using the same pins as RAM for addressing (the RD line, the WR line and the IO Request line), BasicX addresses the 65 536 ports.

Warning

For the *Address* argument, do not use values below 607 (&H25F).

This command enables the RAM/XIO pins. If you have any other functions or data on these pins, then they will be overridden.

Example

```
Dim Value As Byte
Dim Address As New UnsignedInteger

Address = &H700

' Get port data.
Value = GetXIO(Address)
```

GetXRAM procedure

Syntax

BX-01 only

Call GetXRAM(*Address, Buffer, Count*)

Arguments

Item	Type	Direction	Description
Address	UnsignedInteger	Input	Starting address in extended RAM. Legal range is 608 to 65 535.
Buffer	Any type	Input/Output	Variable or array in RAM to which data is copied
Count	UnsignedInteger	Input	Number of bytes to transfer. Legal range is 1 to 64 928.

Description

GetXRAM copies data from extended RAM into local RAM variables. The lengths of both local and extended RAM are 64 KB.

GetXRAM can transfer an arbitrarily large block of memory in a single operation, and the block is allowed to span multiple variables in RAM.

Warning

If the copy operation overflows RAM memory, the system may crash.

Internal RAM in the BasicX chip occupies addresses in range 0 to 607 (&H25F). **Any transfers into RAM used by the BasicX operating system may crash the system.** Please see BasicX RAM for more information about this subject.

Example

```
Sub Main()

    Dim LocalData(1 To 20) As Single

    ' Write the array to XRAM, starting at location
    ' 4096 (&H1000). Use four bytes per element
    ' for floating point type.
    Call PutXRAM( &H1000, LocalData, 20*4 )

    ' Retrieve the array from XRAM. Syntax is similar.
    Call GetXRAM( &H1000, LocalData, 20*4 )

End Sub
```

InputCapture procedure

Syntax

Call InputCapture(*CaptureArray, NumberOfPulses, EdgeTrigger*)

Arguments

Item	Type	Direction	Description
CaptureArray	Array of UnsignedInteger	Output	Array of pulse widths. Units are (1 / 7 372 800) seconds Pulse width range is 1 to 65 535 (about 136 ns to 8.89 ms).
NumberOfPulses	Integer	Input	Number of pulses to capture.
EdgeTrigger	Byte	Input	Trigger mechanism -- 0 means a falling edge starts the capture, 1 means a rising edge starts the capture.

Description

InputCapture captures a pulsetrain from the input capture pin (see pin definitions). By utilizing special hardware within the BasicX chip, the procedure measures pulse widths to very precise tolerances -- values are in units of 1 / 7 372 800 seconds (about 135.6 nanoseconds).

InputCapture suspends the calling task until *CaptureArray* is filled. The procedure does not tie up the machine waiting for input – other tasks are allowed to run while InputCapture is waiting.

On the BX-01 and BX-35, the input capture pin is always a tristate (high impedance) input. On the BX-24, the input capture pin is shared with I/O pin 12, which means pin 12 should be set to either input-tristate or input-pullup before calling InputCapture (see procedure PutPin).

Note – once captured, the same *CaptureArray* pulsetrain can be output through the OutputCapture procedure.

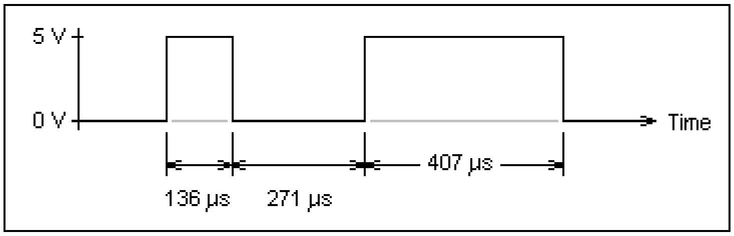
Warning

InputCapture does not start recording anything until the specified edge trigger (rising or falling) is detected. If the edge never occurs, the procedure never returns.

Timeouts return 65 535 (&HFFFF). That is, if a capture starts, and if a timeout occurs during one or more pulses, the timed-out pulses return values of 65 535.

InputCapture takes over Timer1. If any other task or device is using Timer1, there will be a conflict. The Com2 serial port is an example of a device that use Timer1.

Example



In this example, it is assumed that the above pulse train is received at the input capture pin:

```
Sub Main()

    Dim PulseTrain(1 To 3) As New UnsignedInteger

    ' Get 3 samples, where the first sample starts with a rising edge.
    Call InputCapture(PulseTrain, 3, 1)

    ' After the capture, the array contains approximately these values:
    '
    '     PulseTrain(1) = 1000 => 136 us
    '     PulseTrain(2) = 2000 => 271 us
    '     PulseTrain(3) = 3000 => 407 us

End Sub
```

Note that both high and low pulse widths are recorded in the *PulseTrain* array.

Pin numbers:

BX-01 InputCapture pin: 31 (PDIP)

BX-24 InputCapture pin: 12 (shared with I/O pin)

BX-35 InputCapture pin: 20 (PDIP)

LCase function

Syntax

F = LCase(*StringVar*)

Arguments

Item	Type	Direction	Description
StringVar	String	Input	Input string
F	String	Output	Output string

Description

Converts a string to lower case.

Example

```
Dim Tx1 As String
Dim Tx2 As String

Tx1 = "ABC"
Tx2 = LCase(Tx1)   ' Tx2 is "abc"
```

Len function

Syntax

F = Len(*StringVar*)

Arguments

Item	Type	Direction	Description
StringVar	String	Input	String variable
F	Integer	Output	Length of string

Description

Finds the length of a string.

Example

```
Dim Length As Integer
Dim Tx1 As String
Dim Tx2 As String * 10

Tx1 = "ABC"

Length = Len(Tx1) ' Length of Tx1 is 3.
Tx2 = "ABC"       ' Tx2 is left-justified, blank-filled.
Length = Len(Tx2) ' Length of Tx2 is (constant) 10.

Tx1 = ""
Length = Len(Tx1) ' Now length of Tx1 is zero.
```

LockTask procedure

Syntax

Call LockTask( )

Arguments

None.

Description

Locktask prevents any other tasks from running (with some exceptions -- see below). BasicX will only execute the current task. Other tasks won't run until a call to UnlockTask, Delay, Sleep or any other call that would cause the current task to switch, such as queue or networking system calls, or if another task is triggered by a hardware interrupt.

It is permissible to call LockTask if a task is already locked -- multiple calls to LockTask have the same effect as a single call if a task is already locked. For example, you don’t need 2 calls to UnlockTask in order to undo 2 calls to LockTask, generally speaking.

Warning

If other time critical tasks are also running when the LockTask command is executed, the other tasks generally will not run. Care must be taken to cooperate with other tasks as required.

All tasks generally have the same priority, although if another task is blocked and waiting for a hardware interrupt (see WaitForInterrupt), the interrupt event has priority. The locked task becomes temporarily unlocked and the task scheduler resumes normal task switching. As soon as the previously-locked task resumes running, it becomes locked again.

Log function

Syntax

F = Log(*Operand)*

Arguments

Item	Type	Direction	Description
Operand	Single	Input	Operand
F	Single	Output	Function return

Description

Calculates the natural logarithm (base *e*). The value of *e* is approximately 2.718 282.

Example

```
Dim F As Single

F = Log(20.08554) ' F is 3.0 (i.e. 20.085 54 is approximately e^3)
```

Log10 function

Syntax

F = Log10(*Operand)*

Arguments

Item	**Type**	**Direction**	**Description**
Operand	Single	Input	Operand
F	Single	Output	Function return

Description

Calculates the logarithm base 10.

Example

```
Dim F As Single

F = Log10(100.0) ' F is 2.0.
```

MemAddress function

Syntax

F = MemAddress(*Variable*)

Arguments

Item	Type	Direction	Description
Variable	Any type	Input	Variable or array
F	Integer	Output	Address of argument

Description

MemAddress returns the RAM address of the argument. When used in conjunction with RAMPeek or RAMPoke, these functions allow you to modify data directly, while bypassing the restrictions normally imposed by the language.

If the variable is an array, string or multi-byte variable, MemAddress returns the address of the first byte or least significant byte..

Warning

MemAddress should not be used for addresses beyond 32 767, which is the maximum legal value for the function return type. See MemAddressU if you need to handle higher addresses.

Example

See MemAddressU for example.

MemAddressU function

Syntax

F = MemAddressU(*Variable*)

Arguments

Item	Type	Direction	Description
Variable	Any type	Input	Variable or array
F	UnsignedInteger	Output	Address of argument

Description

MemAddressU returns the RAM address of the argument. When used in conjunction with RAMPeek or RAMPoke, these functions allow you to modify data directly, while bypassing the restrictions normally imposed by the language.

If the variable is an array, string or multi-byte variable, MemAddressU returns the address of the first byte or least significant byte.

Example

```
Sub Main()

    Dim B(1 to 5) As Byte, I As Integer
    Dim Value As Byte

    ' Fill byte array with even numbers.
    For I = 1 to 5
        B(I) = 2 * CByte(I)
    Next

    ' Read element 3 of the array, which is actually
    ' offset 2 bytes after the beginning of the array
    ' in memory.
    Value = RAMPeek( MemAddressU(B)+2 )

    ' At this point, Value is 6.

End Sub
```

Mid function

Syntax

F = Mid(*StringVar, Start, Length*)

Mid(*StringVar, Start, Length*) = *F*

Arguments (function return)

Item	Type	Direction	Description
StringVar	String	Input	Source string
Start	Integer	Input	Start of substring in *StringVar*
Length	Integer	Input	Length of substring in *StringVar*
F	String	Output	Destination string

Arguments (left side of assignment)

Item	Type	Direction	Description
StringVar	String	Output	Destination string
Start	Integer	Input	Start of substring in *StringVar*
Length	Integer	Input	Length of substring in *StringVar*

Description

Mid copies a substring from one string to another. Mid is a unique function that can be used on either side of an assignment statement (see examples).

Warning

If the source and destination strings don't have the same length, the destination string is either truncated or blank-filled as needed.

Example

```
Sub Main()

    Dim Istr As String
    Dim Ostr As String

    Istr = "Time heals all wounds"
    Ostr = Istr

    Mid(Ostr, 6, 6) = Mid(Istr, 16, 6)
    Mid(Ostr, 12, 5) = Mid(Istr, 11, 5)
    Mid(Ostr, 17, 5) = Mid(Istr, 6, 5)
    Mid(Ostr, 19, 1) = "e"

    ' At this point, Ostr = "Time wounds all heels"

End Sub
```

OpenCom procedure

Syntax

Call OpenCom(*PortNumber, BaudRate, InputQueue, OutputQueue*)

Arguments

Item	Type	Direction	Description
PortNumber	Byte	Input	Serial port number. Range is 1 to 3.
BaudRate	Long	Input	Baud rate. See below for allowable values.
InputQueue	Byte array	Input/Output	Data buffer for incoming data.
OutputQueue	Byte array	Input/Output	Data buffer for outgoing data.

Allowable values for *BaudRate:*

For port 1 (Com1) -- range is 2400 to 460 800.
For port 2 (Com2) -- range is 300 to 19 200 (BX-01 only)
For port 3 (Com3) -- range is 300 to 19 200 (BX-24, BX-35 only)

Description

OpenCom is used to set up and initialize a BasicX serial port. The procedure attaches two queues to the port -- one for input and one for output. You must call OpenQueue for both queues before calling OpenCom. All ports use 1 start bit and 1 stop bit. Ports 1 and 2 use no parity and 8 data bits. Port 3 has more flexibility regarding parity, data bits and inverted signals (see DefineCom3).

Once a port is opened, bytes placed in the output queue are sent out the port, and any bytes that arrive are placed in the input queue. The two queues are used for data buffering, and interrupt-driven I/O occurs in the background. Pin numbers:

BX-01: Com1 uses pins 10 and 11. Com2 uses pins 1 and 12; also Timer1.
BX-24: Com1 uses pins 1 and 2. Com3 uses any I/O pins (other than 1 or 2); also Timer2.
BX-35: Com1 uses pins 14 and 15. Com3 uses any I/O pins (other than 14 or 15); also Timer2.

Warning

OpenQueue must be called for both input and output queues before calling OpenCom. If an input queue fills with bytes faster than the program can remove them, the bytes will be lost.

On the BX-01, Com1 is also the network and cannot be used as a serial port at the same time. **If you use Com1 as a serial port on the BX-01 Developer Board, you must disable the network** by setting pin 14 to output-high (see procedure PutPin).

On the BX-24 and BX-35, DefineCom3 must be called before OpenCom for port 3 (see DefineCom3).

Example

See OpenComEx.bas example file.

OpenNetwork procedure

Syntax

BX-01 Only

Call OpenNetwork(*BoardAddress, GroupAddress*)

Arguments

Item	Type	Direction	Description
BoardAddress	UnsignedInteger	Input	Node address. Range is 0 to 65 279 (&H0000 to &HFEFF).
GroupAddress	Byte	Input	Group address. Range is 0 to 254 (&H00 to &HFE).

Description

Defines the network and group address of the local BasicX Chip and enables access from remote BasicX chips.

If you select the network via the BasicX downloading system, the network is started automatically. OpenNetwork is not necessary in this case.

Data sent and received via the network needs to be addressed so that it goes to the correct BasicX chip or correct group of BasicX chips.

Some node addresses have special meanings:

`&HFFFF` -- Broadcast this message to **all** BasicX chips
`&HFFxx` -- Broadcast this message to all BasicX chips that are members of group &Hxx

Warning

Every BasicX chip on a network must have a unique address. Opening a networked BasicX chip with the same address as another BasicX chip can cause problems.

Example

```
' This call allows us to receive all packets addressed to
' BoardAddress 1234h. We will also receive groupcasts to
' GroupAddress 32h
Call OpenNetwork (&H1234, &H32)
```

OpenQueue procedure

Syntax

Call OpenQueue(*Queue, Size*)

Arguments

Item	Type	Direction	Description
Queue	Byte array	Input/Output	Array used to create the queue.
Size	Integer	Input	Size (in bytes) of *Queue*. Minimum is 10 bytes.

Description

Creates a queue from an array of bytes.

Queues are data structures that have special properties. Queues act as data storage elements that can be filled and emptied by tasks. Special code within the BasicX chip is used for automatically transferring queue data between tasks.

Internally, a queue is implemented as a circular buffer, and pointers for the queue are maintained within the queue itself. Opening the queue initializes the pointers. The internal pointer overhead requires 9 bytes, so if you define a 20 byte queue array (for example), you really only have 11 bytes available for data.

Warning

Queues need to be large enough to accept the largest data items placed in them, in addition to 9 bytes required for internal overhead. The smallest allowable queue is 10 bytes.

Example

```
Dim ICom2(1 to 30) As Byte
Dim OCom2(1 to 30) As Byte

Sub Main()

    Dim Ch As Byte

    ' Open input and output queues.
    Call OpenQueue(ICom2, 30)
    Call OpenQueue(OCom2, 30)

    ' Open serial port and attach both queues to the port.
    Call OpenCom(2, 19200, ICom2, OCom2)

    Call PutQueueStr(OCom2, "Hello World!")

End Sub
```

OpenSPI procedure

Syntax

Call OpenSPI(*Channel, SetupByte, PinNumber*)

Arguments

Item	Type	Direction	Description
Channel	Byte	Input	Channel number. Range is 1 to 4.
SetupByte	Byte	Input	A byte used to initialize the SPI port prior to accessing the device using the SPICmd command. See below for format.
PinNumber	Byte	Input	Pin number of the pin used to chip select the SPI device. The chip select is an active low signal.

Format for *SetupByte:*

Name **Action**

```
SPI_LSB        LSB is transmitted first
SPI_CPOL       SCK is high when idle
SPI_CPHA       See Table


SPI_   SPI_
CPHA   CPOL    Result

 0      0      Rising edge in middle of bit cell
 0      1      Falling edge in middle of bit cell
 1      0      Falling edge in middle of bit cell
 1      1      Rising edge in middle of bit cell

SPI_SCK04      SCK = CLK / 4
SPI_SCK16      SCK = CLK / 16
SPI_SCK64      SCK = CLK / 64
SPI_SCK128     SCK = CLK / 128
```

Description

BasicX has a Serial Peripheral Interface (SPI) bus built into the hardware of the chip. Using this bus, peripherals from other manufacturers such as Motorola and National Semiconductor can be utilized for special functions not capable of being performed by the BasicX chip directly.

The OpenSPI command provides the programmer the ability to have 4 SPI devices attached to a BasicX chip.

The SPI bus can be configured in many different ways: polarity, clock phase, speed. The OpenSPI command allows you to setup each channel independently of other devices.

Example

```
Dim SetupByte As Byte

SetupByte = SPI_CPHA or SPI_SCK64
Call OpenSPI( 3, SetupByte, 16 )
```

OpenWatchdog procedure

Syntax

Call OpenWatchdog(*TimeoutValue*)

Arguments

Item	Type	Direction	Description
TimeoutValue	Byte	Input	The TimeoutValue N is such that ($16 * 2^N$) is the timeout delay in ms, where N is range 0 to 7 (see below).

Allowable values for *TimeoutValue:*

0	= 16 milliseconds
1	= 32 ms
2	= 64 ms
3	= 128 ms
4	= 256 ms
5	= 512 ms (approximately 1/2 second)
6	= 1024 ms (approximately 1 second)
7	= 2048 ms (approximately 2 seconds)

Description

OpenWatchdog starts the watchdog timer, which will restart the processor unless the timer is periodically refreshed.

What's a watchdog timer? Sometimes an application is so critical that you want to keep it running in almost any condition. If a program locks up or crashes for some reason -- perhaps it executes an unforseen path, or an electrical spike causes garbled data -- a safety feature called a *watchdog timer* can restart the processor. The timer counts down to a preset value, and if the timer is not refreshed before *TimeoutValue* elapses, the processor is reset.

OpenWatchdog starts the watchdog timer. Afterwards, the program is supposed to kick the timer periodically by calling procedure Watchdog. This call is typically inserted in a critical section of code that is executed periodically. If the program malfunctions and fails to execute the critical code, the timer (ideally) never gets refreshed. The watchdog will then reset and reboot the processor after the timeout period elapses, and the program starts over at the beginning.

Warning

For safety reasons, the BasicX operating system includes no provisions for turning off a watchdog timer once it’s turned on. Note also that the watchdog may interfere with downloading new programs -- if the watchdog is active, you may need to do a hard reset whenever you download a new program.

Example

See WatchDogEx.bas example file.

OutputCapture procedure

Syntax

Call OutputCapture(*CaptureArray, NumberOfPulses, StartingEdge*)

Arguments

Item	Type	Direction	Description
CaptureArray	Array of UnsignedInteger	Input	Array of pulse widths. Units are in 1 / 7 372 800 seconds (about 135.6 ns).
NumberOfPulses	Integer	Input	Number of pulses to generate.
StartingEdge	Byte	Input	Edge type of starting pulse. Falling edge is 0, rising edge is 1.

Description

OutputCapture generates a pulsetrain on the output capture pin (see pin definitions). By utilizing special hardware within the BasicX chip, the procedure generates pulse widths to very precise tolerances -- values are in units of 1 / 7 372 800 seconds (about 135.6 nanoseconds).

OutputCapture suspends the calling task until *CaptureArray* is exhausted. OutputCapture is a convenient way to reproduce the pulsetrain detected by the InputCapture procedure.

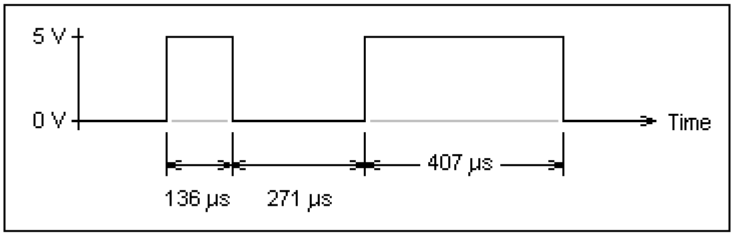
Warning

OutputCapture takes over Timer1. If any other task or device is using Timer1, there will be a conflict. The Com2 serial port is an example of a device that use Timer1.

Example

```
Sub Main()

    Dim PulseTrain(1 To 3) As New UnsignedInteger

    PulseTrain(1) = 1000   ' (1000 / 7 372 800) = 136 microseconds
    PulseTrain(2) = 2000   ' (2000 / 7 372 800) = 271 microseconds
    PulseTrain(3) = 3000   ' (3000 / 7 372 800) = 407 microseconds

    ' Generate the 3 pulses, starting with a rising edge.
    Call OutputCapture(PulseTrain, 3, 1)

End Sub
```

This example produces the following pulse train at the output capture pin:



BX-01 OutputCapture pin: 29 (PDIP)

BX-24 OutputCapture pin: 27

BX-35 OutputCapture pin: 18 (PDIP)

PeekQueue procedure

Syntax

Call PeekQueue(*Queue, Variable, Count*)

Arguments

Item	Type	Direction	Description
Queue	Byte array	Input	Queue from which data is copied.
Variable	Any type	Output	Destination of copied data.
Count	Integer	Input	Number of bytes copied.

Description

PeekQueue copies data from a queue into RAM variables, but without actually removing the data from the queue. PeekQueue can cross boundries between variables to copy multiple pieces of data in a single operation. Variables do not have to be the same type going in as going out.

If there is nothing in the queue, PeekQueue will suspend the current task until the correct amount of data is placed into the Queue.

Warning

If there is nothing in the queue, and no task ever places anything in the queue, the procedure will not return and the task will halt indefinitely.

Example

See PeekQueueEx.bas example file.

PersistentPeek function

Syntax

F = PersistentPeek(*Address*)

Arguments

Item	Type	Direction	Description
Address	UnsignedInteger	Input	Address of data source, range 32 to 511
F	Byte	Output	Destination of copied data.

Description

PersistentPeek reads one byte of data located in persistent memory.

There are 480 bytes of persistent memory, located at addresses 32 to 511.

Example

```
Dim Data As Byte

' Read EEPROM data at address 1234.
Data = PersistentPeek(1234)
```

PersistentPoke procedure

Syntax

Call PersistentPoke(*Value, Address*)

Arguments

Item	Type	Direction	Description
Value	Byte	Input	Address of destination, range 32 to 511.
Address	UnsignedInteger	Input	Source of copied data.

Description

PersistentPoke writes one byte of data to a location in persistent memory.

There are 480 bytes of persistent memory, located at addresses 32 to 511.

Warning

Writing to addresses outside the legal range may crash the system.

Note -- persistent memory is implemented in EEPROM, which has limits on how many times you can write to it before it becomes unusable. Typical write limits are 100 000 to 1 000 000. Make sure your program is not stuck in a fast loop writing to persistent memory or it will be destroyed quickly.

Example

```
Dim Data As Byte

' Write value 65 to EEPROM address 1234.
Data = 65
Call PersistentPoke(Data, 1234)
```

PlaySound procedure

Syntax

BX-24, BX-35 Only

Call PlaySound(*Pin, StartAddress, Length, SampleRate, RepeatCount*)

Arguments

Item	Type	Direction	Description
Pin	Byte	Input	Output pin number.
StartAddress	UnsignedInteger	Input	Starting address of data in EEPROM.
Length	UnsignedInteger	Input	Length of data. Units are in bytes.
SampleRate	UnsignedInteger	Input	Sample rate. Units are Hz.
RepeatCount	UnsignedInteger	Input	Number of times to repeat the sound.

Description

PlaySound generates sound from sampled data stored in EEPROM.

Warning

This procedure halts all multitasking for the duration of the call. The real time clock (RTC), task switching and network traffic are suspended during this time. If the combination of *Length*, *SampleRate* and *RepeatCount* is such that the sound duration exceeds about 1.95 ms, the RTC will lose time.

Pow function

Syntax

F = Pow(*Operand, Exponent*)

Arguments

Item	Type	Direction	Description
Operand	Single	Input	Operand
Exponent	Single	Input	Exponent
F	Single	Output	Function return

Description

Raises the operand to the power specified by the exponent.

Example

```
Dim F As Single

F = Pow(10.0, 3.0) ' F = 10^3 = 1000.0
```

Known Bugs

The function return is incorrect if the operand is negative and the exponent has an integer value.

PulseIn procedure (float version)

Syntax

Call PulseIn(*Pin, State, PulseWidth*)

Arguments

Item	Type	Direction	Description
Pin	Byte	Input	Pin number
State	Byte	Input	Specifies either high (1) or low (0) pulse
PulseWidth	Single	Output	Time interval. Units are in seconds. Valid range is about 1.085 µs to 71.1 ms. Timeout returns 0.0.

Description

Measure's the width of a pulse on the specified I/O pin.

PulseIn waits for a transition to the state you define, then measures the pulse's duration until it either changes state again or times out. PulseIn times out in approximately 71 milliseconds and returns 0.0 for *PulseWidth*.

PulseWidth resolution is about 1.085 µs.

Warning

PulseIn dedicates the processor to looking for pulses. The real time clock (RTC), task switching and network traffic are suspended during this time. Input pulses longer than 1.95 milliseconds will result in a loss of time in the RTC.

Example

```
Dim PulseWidth As Single

' Wait for a high pulse on pin 16.
Call PulseIn(16, 1, PulseWidth)
```

PulseIn function (integer version)

Syntax

PulseWidth = PulseIn(*Pin, State*)

Arguments

Item	Type	Direction	Description
Pin	Byte	Input	Pin number
State	Byte	Input	Specifies either high (1) or low (0) pulse
PulseWidth	Integer	Output	Time interval, in units of 8 / 7 372 800 seconds (about 1.085 µs). Valid range is 1 to 32 767 units. Timeout returns zero or negative value.

Description

Measure's the width of a pulse on the specified I/O pin.

PulseIn waits for a transition to the state you define, then measures the pulse's duration until it either changes state again or times out. PulseIn times out in approximately 35.5 ms and returns a 0 or negative value for *PulseWidth*.

Warning

PulseIn dedicates the processor to looking for pulses. The real time clock (RTC), task switching and network traffic are suspended during this time. Input pulses greater than a count of 1800 will result in the loss of time in the RTC.

Example

```
Dim PulseWidth As Integer

' Wait for a high pulse on pin 16.
PulseWidth = PulseIn(16, 1)
```

PulseOut procedure (float version)

Syntax

Call PulseOut(*Pin, PulseWidth, State*)

Arguments

Item	Type	Direction	Description
Pin	Byte	Input	Pin number
PulseWidth	Single	Input	Time interval. Units are in seconds, range is about 1.085 µs to 71.1 ms.
State	Byte	Input	Specifies either high (1) or low (0) pulse

Description

PulseOut sends a logic high or logic low pulse from any available I/O pin. The procedure waits until the pulse has been sent before returning.

The resolution of PulseOut is 8 / 7 372 800 seconds (about 1.085 µs).

Note -- PulseOut can be used solely as a means of generating a delay -- that is, without affecting physical I/O pins. This is done by using pin 0 as the pin parameter. Pin 0 is treated as a dummy pin.

Warning

This procedure halts all multitasking for the duration of the call. The real time clock (RTC), task switching and network traffic are suspended during this time. Output pulses greater than 1.95 milliseconds will result in the loss of time in the RTC.

Also, the behavior of PulseOut is undefined if *PulseWidth* violates range constraints.

Example

```
' Send a high pulse to pin 17. Pulse width is 1.5 ms.
Call PulseOut(17, 1.5E-3, 1)
```

PulseOut procedure (integer version)

Syntax

Call PulseOut(*Pin, PulseWidth, State*)

Arguments

Item	Type	Direction	Description
Pin	Byte	Input	Pin number
PulseWidth	UnsignedInteger	Input	Time interval, in units of 8 / 7 372 800 seconds (about 1.085 µs). Range is 1 to 65 535 units.
State	Byte	Input	Specifies either high (1) or low (0) pulse

Description

PulseOut sends a logic high or logic low pulse from any available I/O pin. The procedure waits until the pulse has been sent before returning.

The resolution of PulseOut is 8 / 7 372 800 seconds (about 1.085 µs).

Note -- PulseOut can be used solely as a means of generating a delay -- that is, without affecting physical I/O pins. This is done by using pin 0 as the pin parameter. Pin 0 is treated as a dummy pin.

Warning

This procedure halts all multitasking for the duration of the call. The real time clock (RTC), task switching and network traffic are suspended during this time. Output pulses greater than 1.95 ms will result in the loss of time in the RTC.

Also, the behavior of PulseOut is undefined if *PulseWidth* violates range constraints.

Example

```
Dim PulseWidth As Integer

' Pulse width is 1.5 ms.
PulseWidth = 1382 ' Unit conversion: 1.5E-3/1.085E-6 = 1382

' Send a high pulse to pin 17.
Call PulseOut(17, PulseWidth, 1)
```

Put1Wire procedure

Syntax

BX-24, BX-35 Only

Call Put1Wire(*PinNumber, BitValue*)

Arguments

Item	Type	Direction	Description
PinNumber	Byte	Input	Pin number.
BitValue	Byte	Input	Bit value. Range is 0 to 1.

Description

Transmits a single bit using the Dallas 1-Wire protocol. The bit is output on the specified pin number.

PutBit procedure

Syntax

BX-24, BX-35 Only

Call PutBit(*Operand, BitNumber, Value*)

Arguments

Item	Type	Direction	Description
Operand	Any variable or array	Input	Destination of bit.
BitNumber	Byte	Input	Bit number (numbering starts at 0). Range is 0 to 255.
Value	Byte	Input/Output	Value of bit. Range is 0 to 1.

Description

PutBit sets the specified bit to the state defined by *Value*. Bit numbering starts at 0. If the operand is an array, PutBit can write to any of the first 256 bits of the array.

Example

```
' This illustrates PutBit for a single byte.

    Dim A As Byte, B As Byte, C As Byte

    A = bx00100000

    Call PutBit(A, 2, 1) ' Here A = bx00100100
    Call PutBit(A, 5, 0) ' Here A = bx00000100

' This illustrates PutBit for a 32-bit Long array.

    Dim L(1 To 2) as Long

    L(2) = 0

    ' Set the first bit of the second element.
    Call PutBit(L, 32, 1)  ' Here, L(2) = 1.
```

PutDAC procedure

Syntax

Call PutDAC(*Pin, NondimVolt, DACcounter*)

Arguments

Item	Type	Direction	Description
Pin	Byte	Input	Pin number.
NondimVolt	Single	Input	Nondimensional voltage. Range is 0.0 to 1.0. Resolution is about 0.392 %.
DACcounter	Byte	Input/Output	DACcounter is a value that must be returned each time the routine is called so that the DAC remains in sync. If you have multiple DACs running concurrently, then you must have a different DACcounter for every pin.

Description

PutDAC generates an 8-bit pseudo-analog voltage on an I/O pin. On 5 volt systems, the voltage range is 0.0 V to 5.0 V, with a resolution of about 19.6 mV.

A rapid set of pulses is precisely timed to produce the desired voltage. A simple low pass filter circuit is needed externally to filter the output. PutDAC produces this "blast" of pulses for a short time, then places the pin in a high impedance state before returning.

The external filter circuit is relied upon to maintain the voltage between calls. PutDAC should be called periodically to refresh the pin and keep the voltage within tolerances. The optimum refresh rate depends on the characteristics of the circuit to which the pin is connected. You might consider calling PutDAC in a separate task if you need to refresh the pin continuously.

See **DACPin** for the integer equivalent of PutDAC.

Warning

PutDAC turns the selected pin into an output pin independent of any other setting. Also, if the output pin is not refreshed periodically, the analog output voltage will not be maintained

Example

```
Dim DACcounter As Byte
Const Pin As Byte = 16

' Set pin 16 to 75 percent of full scale.
Call PutDAC(Pin, 0.75, DACcounter)
```

PutDate procedure

Syntax

Call PutDate(*Year, Month, Day*)

Arguments

Item	Type	Direction	Description
Year	Integer	Input	Year. Range is 1999 to 2177.
Month	Byte	Input	Month.
Day	Byte	Input	Day of month.

Description

Sets the date. The day of week is also defined automatically when PutDate is called (see GetDayOfWeek function).

PutEEPROM procedure

Syntax

Call PutEEPROM(*Address, Value, Length*)

Arguments

Item	Type	Direction	Description
Address	Long	Input	Starting location of the destination in EEPROM.
Value	Any type	Input	Starting location of the source in RAM.
Length	Integer	Input	Number of bytes to transfer from RAM to EEPROM.

Description

PutEEPROM transfers data from RAM to EEPROM. The EEPROM memory is where the BasicX program is stored. Since a particular program may not use all available memory, PutEEPROM allows you to use leftover space for nonvolatile data storage.

PutEEPROM can transfer an arbitrarily large block of memory in a single operation, and the block is allowed to span multiple variables in RAM.

Warning

Writing to code space in EEPROM can corrupt an executing program. Any writes should be to addresses beyond the end of the program. In order to determine the last address occupied by code, refer to the code memory section in the MPP map file (the MPP file is generated whenever you compile a program).

Note that EEPROMs have limits on how many times you can write to them before they become unusable. Typical write limits are 100 000 to 1 000 000. Make sure your program is not stuck in a fast loop writing to EEPROM or it will be destroyed quickly.

Example

```
Dim Name As String * 20
Dim Address As String * 20

Sub Main()

    Name = "W.C.Fields"
    Address = "Chattanooga"

    ' Copy 2 strings at 22 bytes per string (20
    ' characters plus 2 bytes overhead per string).
    Call PutEEPROM(1000, Name, 22)
    Call PutEEPROM(1022, Address, 22)

End Sub
```

PutNetwork procedure

Syntax

BX-01 Only

Call PutNetwork(*NodeAddress, MemoryAddress, Value, Result*)

Arguments

Item	Type	Direction	Description
NodeAddress	UnsignedInteger	Input	Node address of the remote system.
MemoryAddress	UnsignedInteger	Input	RAM address of the data to be written. See the discussion of MPX map files for more information about variable locations.
Value	Any scalar type	Input	Source of the copy.
Result	Byte	Output	Result of the network operation. See below for allowable values.

Allowable values for *Result:*

bxNetOk	= 0	No errors
bxNetNoResponse	= 1	No response from remote system
bxNetBusy	= 255	Network command in progress

Description

PutNetwork copies a scalar variable to a RAM location you specify in a remote BasicX system.

The task that executes the PutNetwork procedure will suspend until the data transfer is either acknowledged by the remote system, or a number of retries has been attempted. The task is then awakened and a result value is returned.

Warning

Care must be taken when sending data to a remote system. If you do not send data to the correct location, data in the remote system could become corrupted and make the system unreliable.

Known Bugs

If another node on the network attempts to transmit a network packet simultaneously, the processor may hang.

PutNetworkP procedure

Syntax

BX-01 Only

Call PutNetworkP(*NodeAddress, MemoryAddress, Value, Result*)

Arguments

Item	Type	Direction	Description
NodeAddress	UnsignedInteger	Input	Node address of the remote system.
MemoryAddress	UnsignedInteger	Input	EEPROM address of the data to be written. See the discussion of MPX map files for more information about variable locations.
Value	Any scalar type	Input	Source of the copy.
Result	Byte	Output	Result of the network operation. See below for allowable values.

Allowable values for *Result:*

bxNetOk	= 0	No errors
bxNetNoResponse	= 1	No response from remote system
bxNetBusy	= 255	Network command in progress

Description

PutNetworkP copies a scalar variable to an EEPROM (persistent) location you specify in a remote BasicX system.

The task that executes the PutNetworkP procedure will suspend until the data transfer is either acknowledged by the remote system, or a number of retries has been attempted. The task is then awakened and a result value is returned.

Warning

Care must be taken when sending data to a remote system. If you do not send data to the correct location, data in the remote system could become corrupted and make the system unreliable.

Known Bugs

See procedure PutNetwork.

PutNetworkPacket procedure

Syntax **BX-01 Only**

Call PutNetworkPacket(*Packet, Result*)

Arguments

Item	Type	Direction	Description
Packet	Byte array	Input	Packet.
Result	Byte	Output	Result of the network operation. See below for allowable values.

Allowable values for *Result:*

bxNetOk	= 0	No errors
bxNetNoResponse	= 1	No response from remote system
bxNetBusy	= 255	Network command in progress

Description

PutNetworkPacket is a command typically used by operating system functions that need to send a network packet in a particular format.

PutNetworkPacket assumes that you know the packet format and have already built a formatted packet in memory, which is sent out directly without any operating system supervision.

Warning

This procedure used by operating system functions and is typically not used by user programs.

Known Bugs

See procedure PutNetwork.

PutNetworkQueue procedure

Syntax

BX-01 Only

Call PutNetworkQueue(*NodeAddress, QueueAddress, Value, Result*)

Arguments

Item	Type	Direction	Description
NodeAddress	UnsignedInteger	Input	Address of remote system.
QueueAddress	UnsignedInteger	Input	RAM address of queue on remote system. See the discussion of MPX map files for more information about variable locations.
Value	Any type	Input	Data to be placed in the remote queue.
Result	Byte	Output	Result of network operation. See below for allowable values.

Allowable values for *Result:*

bxNetOk	= 0	No errors
bxNetNoResponse	= 1	No response from remote system
bxNetBusy	= 255	Network command in progress

Description

PutNetworkQueue places data in a queue in a remote system across the network.

This procedure is useful for cases where multiple BasicX systems send data to a common node. An example is a security system where a central station will monitor events. Remote systems can send data to a queue in the central station when events occur.

PutNetworkQueue places the node address of the sending node in the queue in front of the data. In this way the remote computer knows the sender of the data. If the remote system does not need this data, the system must extract and discard it.

Warning

Care must be taken when sending data to a remote system. If you do not send data to the correct location, data in the remote system could become corrupted and make the system unreliable.

Known Bugs

See procedure PutNetwork.

PutPin procedure

Syntax

Call PutPin(*Pin, State*)

Arguments

Item	Type	Direction	Description
Pin	Byte	Input	Pin number.
State	Byte	Output	Pin state. See below for allowable values.

Allowable values for *State:*

bxOutputLow	= 0	Output low (typically 0 volts)
bxOutputHigh	= 1	Output high (typically 5 volts)
bxInputTristate	= 2	Tristate (Z, or high impedance)
bxInputPullup	= 3	Pull-up (P, on-chip 120 k-Ohm pull-up)

Description

PutPin configures an I/O pin to be output low (0), output high (1), input tristate (Z), or input pull-up (P).

PutPin gives you total control over the state of a pin. You can output a high or low value as you might expect. You can also set the pin to tristate, which is also called a high impedance. This is valuable when you are communicating with a bi-directional bus. The fourth state is pull-up, which connects an on-chip pull-up resistor of approximately 120 kΩ. This state is useful when you are reading data from a passive device like a switch.

PutPin is typically used in conjunction with the GetPin function, where PutPin is used to define the state of the pin before reading it.

Example

```
' Set I/O pin 17 high, then wait 1/2 second before pulling it low.
Call PutPin(17, bxOutputHigh)
Call Sleep(0.5)
Call PutPin(17, bxOutputLow)
```

PutQueue procedure

Syntax

Call PutQueue(*Queue, Variable, Count*)

Arguments

Item	Type	Direction	Description
Queue	Byte array	Input/Output	Queue into which data is inserted.
Variable	Any type	Input	Data to insert into queue.
Count	Integer	Input	Number of bytes to insert.

Description

PutQueue copies data from RAM variables into a queue. PutQueue can cross boundries between variables to transfer multiple pieces of data in a single operation. Variables do not have to be the same type going in as going out (see example code below). Note that if an entire array is copied to the queue in a single operation, the data is transferred starting with the lowest element.

If the queue is full, PutQueue will suspend the task until there is enough room to insert the data.

Queues are a convenient way to pass data between tasks or to store data for future processing.

Warning

If there is not enough space left in the queue, and no task ever removes anything from the queue, the procedure will not return and the task will halt indefinitely.

Example

```
Dim Oven(1 To 50) As Byte
Dim Pi As Single
Dim Fridge(1 To 4) As Byte

Sub Main()

    Call OpenQueue(Oven, 50)
    Pi = 3.14159

    ' Put some Pi in the oven.
    Call PutQueue(Oven, Pi, 4)

    ' Put four byte-size pieces of Pi in the Fridge.
    Call GetQueue(Oven, Fridge, 4)

End Sub
```

PutQueueStr procedure

Syntax

Call PutQueueStr(*Queue, StringSource*)

Arguments

Item	Type	Direction	Description
Queue	Byte array	Input/Output	Queue into which string is copied.
StringSource	String	Input	String source to be copied into *Queue*.

Description

PutQueueStr places a string in a queue.

A typical use for this function is to make the equivalent of a "Print" statement, by placing text in a serial port queue for transmission to another device.

Warning

If there is not enough space in the queue, and no task ever removes anything from the queue, PutQueueStr will not return and the task will halt indefinitely. One way to avoid this problem is to break a string into smaller pieces that are fed to the queue incrementally.

Example

```
Dim OCom(1 To 30) As Byte
Dim ICom(1 To 30) As Byte

Sub Main()

    Dim Howdy As String * 20

    Call OpenQueue(OCom, 30)
    Call OpenQueue(ICom, 30)
    Call OpenCom(2, 19200, ICom, OCom)

    Howdy = "Hello World!"
    Call PutQueueStr(OCom, Howdy)

    ' Append carriage return and line feed
    Call PutQueueStr(OCom, Chr(13) & Chr(10))

End Sub
```

PutTime procedure

Syntax

Call PutTime(*Hour, Minute, Second*)

Arguments

Item	Type	Direction	Description
Hour	Byte	Input	Hours. Range is 0 to 23.
Minute	Byte	Input	Minutes after the hour.
Second	Single	Input	Seconds. Resolution is about 1.95 ms.

Description

Sets the time of day in 24-hour format.

PutTimestamp procedure

Syntax

Call PutTimestamp(*Year, Month, Day, Hour, Minute, Second*)

Arguments

Item	**Type**	**Direction**	**Description**
Year	Integer	Input	Year. Range is 1999 to 2177.
Month	Byte	Input	Month.
Day	Byte	Input	Day.
Hour	Byte	Input	Hours. Range is 0 to 23.
Minute	Byte	Input	Minutes.
Second	Single	Input	Seconds. Resolution is about 1.95 ms.

Description

Sets the date and time of day. Time is in 24-hour format. PutTimestamp also automatically defines the day of week (see GetDayOfWeek function).

PutXIO procedure

Syntax **BX-01 only**

Call PutXIO(*Address, Value*)

Arguments

Item	Type	Direction	Description
Address	UnsignedInteger	Input	I/O address, range 607 to 65 535.
Value	Byte	Input	Value to be sent to the port.

Description

PutXIO sends data to an eXtended I/O port. BasicX supports up to 65 536 of these I/O ports, making a total of 512 Kbits of I/O.

Using the same pins as RAM for addressing (the RD line, the WR line and the IO Request line), BasicX addresses the 65 536 ports.

Warning

For the *Address* argument, do not use values below 607 (&H25F).

This command enables the RAM/XIO pins. If you have any other functions or data on these pins, they will be overridden.

Example

```
Dim Address As New UnsignedInteger
Dim Value As Byte

Address = &H3213
Value = &H47

' Output the data.
Call PutXIO(Address, Value)
```

PutXRAM procedure

Syntax

BX-01 only

Call PutXRAM(*Address, Buffer, Count*)

Arguments

Item	Type	Direction	Description
Address	UnsignedInteger	Input	Starting address in extended RAM. Range is 608 to 65 535.
Buffer	Any type	Input	Variable or array in RAM from which data is copied
Count	UnsignedInteger	Input	Number of bytes to transfer. Range is 1 to 64 928.

Description

PutXRAM copies data from local RAM variables into extended RAM. The lengths of both local and extended RAM are 64 KB.

PutXRAM can transfer an arbitrarily large block of memory in a single operation, and the block is allowed to span multiple variables in RAM.

Example

```
Sub Main()

    Dim LocalData(1 To 20) As Single

    ' Write the array to XRAM, starting at location
    ' 4096 (&H1000). Use four bytes per element
    ' for floating point type.
    Call PutXRAM( &H1000, LocalData, 20*4 )

    ' Retrieve the array from XRAM. Syntax is similar.
    Call GetXRAM( &H1000, LocalData, 20*4 )

End Sub
```

Randomize procedure

Syntax

Call Randomize

Arguments

None

Description

Randomize uses the system clock to set the value of the seed for the random number generator. See Rnd function for details.

RAMPeek function

Syntax

F = RAMPeek(*Address*)

Arguments

Item	Type	Direction	Description
Address	UnsignedInteger	Input	RAM address
F	Byte	Output	Value of the byte at the above address

Description

RAMPeek allows you to read any byte in RAM memory, while bypassing the rules normally associated with variable types. For example, you can look at the third byte of a 4-byte floating point variable, or look at the bytes of a string directly.

Example

```
Dim Gbyte As Byte
Dim TestString As String * 32

Sub Main()

    ' Read byte at memory location 8756 (&h2234).
    Gbyte = RAMPeek(&h2234)

    TestString = "Hello World!"

    ' Read character 7 of the test string, which
    ' is actually offset 8 bytes after the
    ' beginning of the string in memory.
    Gbyte = RAMPeek( MemAddress(TestString)+ 8 )

    ' At this point, Gbyte is 87 (ASCII "W").

End Sub
```

RAMPoke procedure

Syntax

Call RAMPoke(*Value, Address*)

Arguments

Item	Type	Direction	Description
Value	Byte	Input	Value of the byte to copy to RAM
Address	UnsignedInteger	Input	Address of destination

Description

RAMPoke allows you to write a byte anywhere in RAM memory, while bypassing the rules normally associated with variable types. For example you can modify the top byte of an integer, or modify the bytes of a string directly.

Warning

Internal RAM in the BasicX chip occupies addresses in range 0 to 607 (&H25F). **Any transfers into RAM used by the BasicX operating system may crash the system.** Please see BasicX RAM for more information about this subject.

Example

```
Sub Main()

    Dim TestString As String * 32
    Dim Gbyte As Byte

    TestString = "Hel o World!"

    ' Read character 3 of the test string, which
    ' is actually offset 4 bytes after the
    ' beginning of the string in memory.
    Gbyte = RAMPeek( MemAddress(TestString) + 4 )

    ' At this point, Gbyte is 108 (ASCII "l"). Copy
    ' the byte to the next character.
    Call RAMPoke( Gbyte, MemAddress(TestString) + 5 )

    ' The string now reads "Hello, World!"

End Sub
```

RCTime procedure (float version)

Syntax

Call RCTime(*Pin, State, Interval*)

Arguments

Item	**Type**	**Direction**	**Description**
Pin	Byte	Input	Pin number
State	Byte	Input	Pin state – 0 (logic low) or 1 (logic high)
Interval	Single	Output	Time interval, in units of seconds. The valid range is about 1.085 µs to 71.1 ms. Timeout returns 0.0.

Description

RCTime measures how long an I/O pin stays at a specified state. The pin is configured to input-tristate (high impedance) for the measurement. Timeout returns 0.0. Resolution is about 1.085 µs.

Warning

RCTime dedicates the processor to looking for a transition. The real time clock, task switching and network traffic are suspended during this time.

The procedure overrides any previous pin configuration and leaves the pin as input-tristate.

If the pin is not at the specified state when you call RCTime, the procedure immediately returns with the smallest valid nonzero value for *Interval* (about 1.085E-6).

Example

This example illustrates the use of RCTime to measure the time it takes for a capacitor to discharge.

```
Dim TimeDelay As Single

Call PutPin(17, bxOutputLow) ' Pull I/O pin 17 low.

' Wait about 8.7 microseconds for the capacitor to discharge.
Call Delay(8.7E-6)

' Measure the time it takes for the capacitor to charge to a
' set point. Set pin 17 to input-tristate and then measure how
' long the pin stays at logic low.
Call RCTime(17, 0, TimeDelay)
```

RCTime function (integer version)

Syntax

Interval = RCTime(*Pin, State*)

Arguments

Item	Type	Direction	Description
Pin	Byte	Input	Pin number
State	Byte	Input	Pin state – 0 (logic low) or 1 (logic high)
Interval	Integer	Output	Time interval, in units of 8 / 7 372 800 seconds (about 1.085 μs). The valid range is 1 to 32 767 units. Timeout returns 0 or negative value.

Description

RCTime measures how long an I/O pin stays at a specified state. The pin is configured to input-tristate (high impedance) for the measurement. Timeout returns a 0 or negative value.

Warning

RCTime dedicates the processor to looking for a transition. The real time clock, task switching and network traffic are suspended during this time.

The procedure overrides any previous pin configuration and leaves the pin as input-tristate.

If the pin is not at the specified state when you call RCTime, the procedure immediately returns with 1 as *Interval*

Example

This example illustrates the use of RCTime to measure the time it takes for a capacitor to discharge.

```
Dim TimeDelay As New UnsignedInteger

Call PutPin(17, bxOutputLow) ' Pull I/O pin 3 low.

' Wait about 8.7 microseconds for the capacitor to discharge.
Call Sleep(8)

' Measure the time it takes for the capacitor to charge to a
' set point. Set pin 3 to input-tristate and then measure how
' long the pin stays at logic low.
TimeDelay = RCTime(17, 0)
```

ResetProcessor procedure

Syntax

Call ResetProcessor( )

Arguments

None.

Description

ResetProcessor causes the BasicX processor to reset and reboot within 17 milliseconds. Internally, this procedure actually uses the watchdog timer to reset the processor.

Rnd function

Syntax

F = Rnd

Arguments

Item	Type	Direction	Description
F	Single	Output	Function return

Description

Rnd returns a random number greater than or equal to 0.0 and less than 1.0.

Rnd is a multiplicative congruential random number generator that uses a 32-bit integer seed in static memory. Procedure Randomize can be used to set the seed based on the value of the system clock.

Alternatively, you also have direct access to the seed, which is a system-supplied global variable called SeedPRNG. The seed is a 32-bit Long type.

Semaphore function

Syntax

F = Semaphore(*Variable*)

Arguments

Item	Type	Direction	Description
Variable	Boolean	Input	Boolean variable being used as a semaphore.
F	Boolean	Output	Function returns *true* if the semaphore is owned by this task, *false* if semaphore is already in use by another task.

Description

Semaphore is a function that allows tasks to share variables in a cooperative fashion.

Semaphores protect shared data. A semaphore is a signalling mechanism that allows a task to signal to other tasks whether or not it "owns" a particular block of data. When a task owner is done with the data, the task clears the semaphore, giving up ownership and allowing others to use the data.

Due to the complex nature of the function please refer to the entire section covering the semaphore.

Warning

If a task fails to set a semaphore to *false* when it's done with shared data, other tasks will never be able to use the data, and your system could grind to a halt.

Example

See SemaphoreEx.bas example file.

SerialNumber procedure

Syntax

Call SerialNumber(*Value*)

Arguments

Item	Type	Direction	Description
Value	Array of Byte(1 to 6)	Output	Array containing version and serial numbers. Internal format: Byte 1 -- Major version number Byte 2 -- Minor version number Bytes 3 to 6 -- Four-byte serial number (BX-01 only)

Description

This procedure returns major and minor version numbers of the BasicX chip. On BX-01 systems, the procedure also returns unique serial number data in bytes 3 to 6. On BX-24 systems, bytes 3 to 6 are undefined.

Example

```
Dim SNC(1 to 6) As Byte
Dim MajorVersion As Byte
Dim MinorVersion As Byte
Dim SNumber(1 to 4) As Byte

' Read composite data.
Call SerialNumber(SNC)

' Extract the version numbers.
MajorVersion = SNC(1)
MinorVersion = SNC(2)

' Extract the 4-byte serial number (BX-01 only).
SNumber(1) = SNC(3)
SNumber(2) = SNC(4)
SNumber(3) = SNC(5)
SNumber(4) = SNC(6)
```

Sin function

Syntax

F = Sin(*Operand)*

Arguments

Item	**Type**	**Direction**	**Description**
Operand	Single	Input	Operand
F	Single	Output	Function return

Description

Sine function. Operand is in units of radians.

Example

```
Dim F As Single
Const Pi As Single = 3.14159265

' 30 degrees, converted to radians.
F = Sin(Pi/6.0) ' Here F is 0.5
```

Sleep procedure (float version)

Syntax

Call Sleep(*SleepInterval*)

Arguments

Item	**Type**	**Direction**	**Description**
SleepInterval	Single	Input	The sleep interval has a range of about 0.0 s to 128.0 s. Resolution is about 1.95 ms.

Description

Suspends the current task for approximately the specified time interval. At the end of *SleepInterval*, the task will become ready again. How soon the task actually resumes execution depends on how busy the system is with other tasks.

A sleep of 0.0 is a useful way to allow other tasks to execute, while allowing immediate resumption if no other tasks are eligible to run.

If the task is locked, Sleep will unlock the task (see procedure LockTask).

Warning

Sleep actually waits for a certain number of clock ticks, which means *SleepInterval* represents neither a guaranteed minimum nor maximum delay, since the exact timing depends on where the program is relative to a tick cycle (if you want a guaranteed minimum delay, see procedure Delay).

Example

```
'Set pin 1 high
Call PutPin(1, 1)

'Pause this task for approximately 1/2 s, then wake up
Call Sleep(0.5)

'Set pin 1 low
Call PutPin(1, 0)
```

Sleep procedure (integer version)

Syntax

Call Sleep(*SleepInterval*)

Arguments

Item	Type	Direction	Description
SleepInterval	UnsignedInteger	Input	The sleep interval has a range of 0 to 65 535. Units are 1/512 seconds (about 1.95 ms).

Description

Suspends the current task for approximately the specified time interval. At the end of *SleepInterval*, the task will become ready again. How soon the task actually resumes execution depends on how busy the system is with other tasks.

A sleep of 0 is a useful way to allow other tasks to execute, while allowing immediate resumption if no other tasks are eligible to run.

If the task is locked, Sleep will unlock the task (see procedure LockTask).

Warning

Sleep actually waits for the *SleepInterval* number of clock ticks, which means *SleepInterval* represents neither a guaranteed minimum nor maximum delay, since the exact timing depends on where the program is relative to a tick cycle (if you want a guaranteed minimum delay, see procedure Delay).

Example

```
' Set pin 1 high
Call PutPin(1, 1)

' Pause this task for approximately 1/2 s, then wake up.
Call Sleep(256)

' Set pin 1 low
Call PutPin(1, 0)
```

ShiftIn function

Syntax

BX-24, BX-35 Only

F = ShiftIn(*DataPin, ClockPin, NumberOfBits*)

Arguments

Item	Type	Direction	Description
DataPin	Byte	Input	Data source pin number.
ClockPin	Byte	Input	Clock pin number.
NumberOfBits	Byte	Input	Number of bits. Range is 1 to 8.
F	Byte	Output	Function return.

Description

This function shifts in up to 8 bits of data through the *DataPin* input. The operating system automatically clocks in each bit by using the specified *ClockPin*. In order to be compatible with I2C devices, the bit rate is less than 400 kHz.

Bit ordering is MS bit first, LS bit last.

Before calling ShiftIn, the clock pin must first be set to the proper level (either high or low).

Example

```
Dim A As Byte

' Set the clock pin low.
Call PutPin(17, bxOutputLow)

' Shift 4 bits into A. Pin 16 is used for the data input.
A = ShiftIn(16, 17, 4)
```

ShiftOut procedure

Syntax

BX-24, BX-35 Only

Call ShiftOut(*DataPin, ClockPin, NumberOfBits, Operand*)

Arguments

Item	Type	Direction	Description
DataPin	Byte	Input	Data source pin number.
ClockPin	Byte	Input	Clock pin number.
NumberOfBits	Byte	Input	Number of bits. Range is 1 to 8.
Operand	Byte	Input	Source of data.

Description

This function shifts out up to 8 bits of data from *Operand* through the *DataPin* output. The operating system automatically clocks out each bit by using the specified *ClockPin*. In order to be compatible with I2C devices, the bit rate is less than 400 kHz.

Bit ordering is MS bit first, LS bit last.

Before calling ShiftIn, the clock pin must first be set to the proper level (either high or low).

Example

```
Dim A As Byte

' Set the clock pin high.
Call PutPin(17, bxOutputHigh)

' Shift 4 bits out of A. Pin 16 is used for the data output.
Call ShiftOut(16, 17, 4, A)
```

SPICmd procedure

Syntax

Call SPICmd(*Channel, PutCount, PutData, GetCount, GetData*)

Arguments

Item	Type	Direction	Description
Channel	Byte	Input	SPI channel number. Range is 1 to 4.
PutCount	Byte	Input	Number of bytes to be sent to the device. Zero means no data.
PutData	Any type	Input	Data to be sent (if PutCount = 0, you still need a dummy argument here).
GetCount	Byte	Input	Number of bytes to receive from the device.
GetData	Any type	Input/Output	Data to be received.

Description

BasicX has a Serial Peripheral Interface (SPI) bus built into the chip. Using this bus, peripherals from other manufacturers such as Motorola and National Semiconductor can be utilized for special functions not capable of being performed by the BasicX chip directly.

The SPI bus is an interesting bus in that data is exchanged by the sender and receiver at the same time. In other words, data is going in both directions simultaneously. Data flow can also be unidirectional if desired -- SPICmd allows either case.

Before calling SPICmd, you must call OpenSPI to initialize an SPI channel.

Warning

This command is for users who understand the SPI bus well. BasicX code is typically fetched from an SPI EEPROM, which means that if the SPI bus is not handled correctly, instruction fetching could be affected.

Sqr function

Syntax

F = Sqr(*Operand)*

Arguments

Item	Type	Direction	Description
Operand	Single	Input	Operand
F	Single	Output	Function return

Description

Square root function.

Example

```
Dim F As Single

F = Sqr(9.0) ' F is 3.0
```

StatusQueue function

Syntax

F = StatusQueue(*Queue*)

Arguments

Item	Type	Direction	Description
Queue	Byte array	Input	Name of the queue to check.
F	Boolean	Output	Returns *true* if there is data in *Queue*. Otherwise returns *false*.

Description

StatusQueue allows the programmer to see if there is any data within a queue before the task tries to obtain data. If a task does not check the queue using StatusQueue and then tries to read data from an empty queue, the task will block until data is available.

Warning

When a queue is being used as a serial port output buffer, StatusQueue is not suitable for determining if the buffer has been flushed. In particular, it is possible for StatusQueue to indicate the queue is empty before a transmission is completed.

Example

```
Dim Queue(1 To 30) As Byte

Sub Main()

    Dim Data As Byte

    Call OpenQueue(Queue, 30)
    Do
        ' If data is in the queue, extract one byte.
        If StatusQueue(Queue) Then
            Call GetQueue(Queue, Data, 1)
        End If
    Loop
End Sub
```

Tan function

Syntax

F = Tan(*Operand)*

Arguments

Item	**Type**	**Direction**	**Description**
Operand	Single	Input	Operand
F	Single	Output	Function return

Description

Tangent function. The operand is in units of radians.

Example

```
Dim F As Single

' Tan(Pi/4)
F = Tan(0.785398) ' F is 1.0.
```

TaskIsLocked function

Syntax

F = TaskIsLocked( )

Arguments

Item	Type	Direction	Description
F	Boolean	Output	Whether task is locked.

Description

TaskIsLocked allows you to find out if the current task is locked. The function is useful if you have a subprogram that needs to lock the task, then restore the lock status upon return.

Timer function

Syntax

F = Timer( )

Arguments

Item	Type	Direction	Description
F	Single	Output	Floating point seconds since midnight. Range is 0.0 to 86 400.0 s.

Description

Returns the time elapsed since midnight. Resolution depends on time of day -- best case is about 1.95 ms (1/512 seconds) for small time values.

Example

```
Dim T1 As Single, T2 As Single, DT As Single

' Find starting time.
T1 = Timer

Call TimedProcedure

' Find ending time.
T2 = Timer

' Calculate elapsed time.
DT = T2 - T1
```

Trim function

Syntax

F = Trim(*StringVar*)

Arguments

Item	Type	Direction	Description
StringVar	String	Input	Input string
F	String	Output	Output string

Description

Removes leading and trailing blanks from a string.

Example

```
Dim Tx1 As String
Dim Tx2 As String

Tx1 = "   Hello, world   "

Tx2 = Trim(Tx1) ' Tx2 is "Hello, world"
```

UCase function

Syntax

F = UCase(*StringVar*)

Arguments

Item	Type	Direction	Description
StringVar	String	Input	Input string
F	String	Output	Output string

Description

Converts a string to upper case.

Example

```
Dim Tx1 As String
Dim Tx2 As String

Tx1 = "abc"
Tx2 = UCase(Tx1)   ' Tx2 is "ABC"
```

UnlockTask procedure

Syntax

Call UnlockTask( )

Arguments

None.

Description

UnlockTask releases a task from being locked. The procedure reverses the effect of the LockTask procedure (locking a task inhibits the operating system from switching to another task). Unlocking a task causes normal task switching to resume.

It is permissible to call UnlockTask if a task is already unlocked -- multiple calls to UnlockTask have the same effect as a single call if a task is already unlocked. For example, you don't need 2 calls to LockTask in order to undo 2 calls to UnlockTask, generally speaking.

ValueS procedure

Syntax

Call ValueS(*StringVar, Value, Success*)

Arguments

Item	Type	Direction	Description
StringVar	String	Input	Input string
Value	Single	Output	Return value
Success	Boolean	Output	Success flag

Description

Converts a string to a Single type. If no errors occur, the number is returned in Value and the Success flag is set to True. Otherwise Value is set to 0.0 and Success is set to False.

The number in the string must consist of numeric digits with optional signs for the number and exponent. A decimal point is also optional. Leading and trailing control characters (such as spaces or tabs) are ignored.

Example

```
Dim Tx As String
Dim Value As Single
Dim Success As Boolean

Tx = "  123 "
Call ValueS(Tx, Value, Success) ' Value is 123.0, Success is True

Tx = "-4.5E+03"
Call ValueS(Tx, Value, Success) ' Value is -4500.0, Success is True

' Illegal characters.
Tx = "&HFF"
Call ValueS(Tx, Value, Success) ' Value is 0.0, Success is False
```

WaitForInterrupt procedure

Syntax

Call WaitForInterrupt(*InterruptType*)

Arguments

Item	Type	Direction	Description
InterruptType	Byte	Input	Interrupt type. See below for allowable values.

Allowable values for *InterruptType:*

bxComparatorToggle	= 0	Comparator toggle state (BX-01 only)
bxComparatorFallingEdge	= 2	Falling edge of comparator (BX-01 only)
bxComparatorRisingEdge	= 3	Rising edge of comparator (BX-01 only)
bxPinLow	= 16	Low level on interrupt pin
bxPinFallingEdge	= 24	Falling edge on interrupt pin
bxPinRisingEdge	= 28	Rising edge on interrupt pin

BX-01 interrupt pin number: 13 (PDIP)
BX-24 interrupt pin number: 11 (shared with I/O pin)
BX-35 interrupt pin number: 17 (PDIP)

Description

WaitForInterrupt allows a task to respond immediately to a critical event from the outside world. This procedure gives you access to hardware interrupts built into the BasicX chip.

WaitForInterrupt blocks the calling task until the triggering event happens. When the event occurs, the task is scheduled to be run immediately. The trigger has priority, even if another task is running and locked (see procedure LockTask), in which case the other task becomes temporarily unlocked.

Warning

If no external event is generated, the calling task could wait indefinitely.

On the BX-24, the interrupt line is shared with I/O pin 11, which means pin 11 should be set to input-tristate or input-pullup if you want to use the interrupt line.

Example

```
' Wait for rising edge on comparator.
Call WaitForInterrupt( bxComparatorRisingEdge )
```

Watchdog procedure

Syntax

Call Watchdog( )

Arguments

None.

Description

Watchdog resets the watchdog timer before it times out.

Before calling Watchdog, you need to call OpenWatchdog to start the watchdog timer. See OpenWatchdog for more information.

X10Cmd procedure

Syntax

BX-24, BX-35 Only

Call X10Cmd(*PinOut, Pin60Hz, HouseCode, KeyCode, RepeatCycles*)

Arguments

Item	Type	Direction	Description
PinOut	Byte	Input	Output pin.
Pin60Hz	Byte	Input	60 Hz pin.
HouseCode	Byte	Input	House code.
KeyCode	Byte	Input	Key code.
RepeatCycles	Byte	Input	Number of repeat cycles.

Description

Transmits an X-10 command at a repetition rate determined by *RepeatCycles*.

Example

```
Const X10_P As Byte = &HC
Const X10_Dim As Byte = &H9
Const X10_Bright As Byte = &HB

Call X10Cmd(16, 17, X10_P, X10_Dim, 8)
Call Delay(1.0)
Call X10Cmd(16, 17, X10_P, X10_Bright, 8)
```