System Library Reference

Section 1 - Routines by Category

The ZBasic System Library provides a rich collection comprising hundreds of subroutines and functions that you can use to add functionality to your application. The routines may be divided into several conceptual categories as shown below.

Type Conversion Functions

`CBit()`	convert a value to type `Bit`
`CBool()`	convert a value to type `Boolean`
`CByte()`	convert a value to type `Byte`
`CByteArray()`	convert an integral value to a reference to a `Byte` array
`CInt()`	convert a value to type `Integer`
`CLng()`	convert a value to type `Long`
`CNibble()`	convert a value to type `Nibble`
`CSng()`	convert a value to type `Single`
`CStr()`	convert a value to type `String`
`CStrHex()`	convert a value to a `String` containing hexadecimal characters
`CType()`	convert a value to an enumeration member
`CUInt()`	convert a value to type `UnsignedInteger`
`CULng()`	convert a value to type `Long`
`FixB()`	convert a `Single` value to type `Byte`
`FixI()`	convert a `Single` value to type `Integer`
`FixL()`	convert a `Single` value to type `Long`
`FixUI()`	convert a `Single` value to type `UnsignedInteger`
`FixUL()`	convert a `Single` value to type `UnsignedLong`
`To<enum>()`	convert a value to an enumeration member

Mathematical Functions

`Abs()`	absolute value
`Acos()`	arc cosine
`Asin()`	arc sine
`Atn()`	arc tangent
`Atn2()`	arc tangent (quadrant-correct)
`Ceiling()`	largest integer not greater than a `Single` value
`Cos()`	cosine
`DegToRad()`	convert degrees to radians
`Exp()`	e^x
`Exp10()`	10^x
`Fix()`	integer portion of a `Single` value
`Floor()`	smallest integer not less than a `Single value`
`Fraction()`	fractional portion of a `Single` value
`Log()`	natural logarithm
`Log10()`	common logarithm
`Max()`	determine the largest of two values
`Min()`	determine the smallest of two values
`Pow()`	raise a value to a power
`RadToDeg()`	convert radians to degrees
`Signum()`	determine if a value is negative, zero or positive
`Sin()`	sine
`SngClass()`	return the class information for a `Single` value
`Sqr()`	square root
`Tan()`	tangent

Memory-related Routines

`BitCopy()`	copy a sequence of bits from one part of RAM to another
`BlockMove()`	copy data from one part of RAM to another
`GetBit()`	extract a bit from a value in RAM
`GetEEPROM()`	copy data from Program Memory to RAM
`GetPersistent()`	copy data from Persistent Memory to RAM
`GetProgMem()`	copy data from Program Memory to RAM
`MemAddress()`	determine the RAM address of a variable
`MemAddressU()`	determine the RAM address of a variable
`MemCmp()`	compare two blocks of data in RAM
`MemCopy()`	copy data from one part of RAM to another
`MemSet()`	initialize a block of memory with a byte value
`PersistentPeek()`	read a byte from Persistent Memory
`PersistentPoke()`	write a byte to Persistent Memory
`PutBit()`	set or clear a bit in a value in RAM
`PutEEPROM()`	copy data from RAM to Program Memory
`PutPersistent()`	copy data from RAM to Persistent Memory
`PutProgMem()`	copy data from RAM to Program Memory
`RamPeek()`	read a byte from RAM
`RamPeekDword()`	read a 32-bit value from RAM
`RamPeekWord()`	read a 16-bit value from RAM
`RamPoke()`	write a byte to RAM
`RamPokeDword()`	write a 32-bit value to RAM
`RamPokeWord()`	write a 16-bit value to RAM
`System.Alloc()`	allocate a block of memory
`System.Free()`	deallocate a block of memory
`System.HeapHeadRoom()`	determine the amount of unused space in the heap
`System.HeapSize()`	determine the amount of space reserved for the heap
`VarPtr()`	determine the RAM address of a variable

String-related Routines

`Asc()`	extract a character value from a string
`Chr()`	convert a character value to a string
`Fmt()`	convert a `Single` value to a string
`LCase()`	convert upper case letters to lower case in a string
`Left()`	return the leftmost characters from a string
`Len()`	determine the number of characters in a string
`Mid()`	extract or set a substring in a string
`Right()`	return the rightmost characters from a string
`StrAddress()`	determine the address where string characters are stored
`StrCompare()`	compare two strings, optionally ignoring alphabetic case
`StrFind()`	search for the first occurrence of a string within a string
`StrReplace()`	replace character sequences in a string
`StrType()`	determine the characteristics of a string
`Trim()`	remove leading and trailing spaces from a string
`UCase()`	convert lower case letters to upper case in a string
`ValueI()`	convert string characters to the equivalent `Integer` value
`ValueL()`	convert string characters to the equivalent `Long` value
`ValueS()`	convert string characters to the equivalent `Single` value

Data Manipulation Routines

`FlipBits()`	reverse the order of bits in a byte
`HiByte()`	extract the high byte of a value
`HiWord()`	extract the high word of a value
`LoByte()`	extract the low byte of a value
`LoWord()`	extract the low word of a value
`MakeDword()`	construct a 32-bit value from two 16-bit values
`MakeWord()`	construct a 16-bit value from two 8-bit values
`MakeString()`	construct a string from a sequence of bytes
`MidWord()`	extract the middle two bytes of a 4-byte value
`SetBits()`	set the state of specified bits in a byte
`Shl()`	shift a value to the left
`Shr()`	shift a value to the right
`ToggleBits()`	change the state of specified bits in a byte

Serial Communication Routines

`Debug.Print`	send strings to the debug console
`CloseCom()`	terminate the use of a serial channel
`ComChannels()`	prepare for using multiple serial channels
`Console.Read()`	retrieve a character from the console input queue
`Console.ReadLine()`	retrieve a line from the console input queue
`Console.Write()`	send a string to the console output queue
`Console.WriteLine()`	send a string to the console output queue
`ControlCom()`	specify flow control pins for a serial channel
`DefineCom()`	set the characteristics of a serial channel
`DefineCom3()`	set the characteristics of serial Com3
`OpenCom()`	prepare a serial channel for use
`SerialIn()`	read a character from an input pin
`SerialOut()`	send a character or the characters of a string out on a pin
`StatusCom()`	determine the status of a serial channel

Queue Management Routines

`ClearQueue()`	delete data from a queue
`GetQueue()`	retrieve data from a queue
`GetQueueBufferSize()`	determine the size of the data area of a queue
`GetQueueCount()`	determine the number of bytes of data in a queue
`GetQueueSpace()`	determine the amount of space available in a queue
`GetQueueStr()`	populate a string with characters from a queue
`OpenQueue()`	prepare a queue for use
`PeekQueue()`	copy data from a queue without removing it
`PutQueue()`	put data in a queue
`PutQueueByte()`	put a byte into a queue
`PutQueueStr()`	put the characters of a string in a queue
`SearchQueue()`	search a queue for a data byte or sequence
`StatusQueue()`	determine if a queue has data available

Date/Time Routines

`GetDate()`	get the month, day, year corresponding to a day number
`GetDateVdlue()`	get the month, day, year corresponding to a day number (packed)
`GetDayNumber()`	compute the day number corresponding to a day of a year
`GetDayOfWeek()`	get the day of the week corresponding to a date value
`GetDayOfYear()`	get the ordinal day of the year corresponding to a date value
`GetElapsedMicroTime()`	compute an elapsed time relative to previous timing data
`GetMicroTime()`	populate a buffer with high resolution timing data
`GetTime()`	get the current hour, minute and second
`GetTimestamp()`	get the current date and time information
`GetTimeValue()`	get the current hour, minute and second (packed)
`PutDate()`	set the current month, day, year
`PutTime()`	set the current hour, minute and second

Input/Output Routines

`ADCtoCom1()`	stream analog conversion data to Com1
`BusRead()`	read data from a bus-oriented device
`BusWrite()`	write data to a bus-oriented device
`CloseI2C()`	deinitialize an I2C communication channel
`ClosePWM()`	deinitialize a 16-bit PWM channel
`ClosePWM8()`	deinitialize an 8-bit PWM channel
`CloseSPI()`	deinitialize an SPI communication channel
`CloseX10()`	deinitialize an X-10 communication channel
`Com1toDAC()`	receive stream of analog conversion data
`CountTransitions()`	count transitions on an input pin
`DACPin()`	produce an analog voltage on an output pin
`DefineBus()`	specify the parameters for accessing a bus-oriented device
`DefineSPI()`	specify the parameters for software-based SPI communication
`DefineX10()`	specify the communication parameters for an X-10 channel
`FreqOut()`	produce a dual-frequency sine wave on an output pin
`Get1Wire()`	receive a bit using the 1-Wire protocol
`Get1WireByte()`	receive a byte using the 1-Wire protocol
`Get1WireData()`	receive one or more bytes using the 1-Wire protocol
`GetADC()`	perform an analog to digital conversion on an input
`GetPin()`	read the state of an input pin
`I2CCmd()`	send/receive data over an I2C channel
`I2CGetByte()`	receive a byte on an I2C channel
`I2CPutByte()`	send a byte on an I2C channel
`I2CStart()`	create a Start condition on an I2C channel
`I2CStop()`	create a Stop condition on an I2C channel
`InputCapture()`	record the high/low times of a pulse train on an input pin
`InputCaptureEx()`	record the high/low times of a pulse train on an input pin
`OpenI2C()`	prepare for I2C communication with an external device
`OpenI2CSlave()`	activate I2C slave mode
`OpenSPI()`	prepare for SPI communication with an external device
`OpenSPISlave()`	activate SPI slave mode
`OpenPWM()`	prepare for 16-bit PWM generation
`OpenPWM8()`	prepare for 8-bit PWM generation
`OpenX10()`	prepare an X-10 communication channel for use
`OutputCapture()`	produce a pulse train
`OutputCaptureEx()`	produce a pulse train on any output pin
`PlaySound()`	reproduce sampled audio on an output pin
`PortBit()`	compose a designator for a specific bit in an I/O port
`PortMask()`	compute the bitmask for the port with which a pin is associated
`PulseIn()`	measure a pulse width on an input pin

`PulseOut()`	generate a pulse on an output pin
`Put1Wire()`	send a bit using the 1-Wire protocol
`Put1WireByte()`	send a byte using the 1-Wire protocol
`Put1WireData()`	send one or more bytes using the 1-Wire protocol
`PutDAC()`	produce an analog voltage on an output pin
`PutPin()`	configure an I/O pin
`PWM()`	initiate 16-bit PWM generation or change the duty cycle
`PWM8()`	initiate 8-bit PWM generation or change the duty cycle
`RCTime()`	measure an RC charge/discharge time
`Reset1Wire()`	send a reset signal using the 1-Wire protocol
`ShiftIn()`	perform synchronous serial input
`ShiftInEx()`	perform synchronous serial input with more configurability
`ShiftOut()`	perform synchronous serial output
`ShiftOutEx()`	perform synchronous serial output with more configurability
`SPICmd()`	perform SPI communication with an external device
`SPIGetByte()`	retrieve a byte from an SPI slave
`SPIGetData()`	retrieve a series of bytes from an SPI slave
`SPIPutByte()`	send a byte to an SPI slave
`SPIPutData()`	send a series of bytes to an SPI slave
`SPIStart()`	initialize an SPI channel
`SPIStop()`	deinitialize an SPI channel
`StatusX10()`	determine the status of an X-10 communication channel
`X10Cmd()`	send commands using the X-10 protocol
`PutTimeStamp()`	set the current date and time information
`Timer()`	get the current clock tick value

Task-related Routines

`CallTask`	prepare a task to begin execution
`DisableInt()`	disable interrupts
`Delay()`	pause a task
`DelayUntilClockTick()`	pause a task
`EnableInt()`	conditionally re-enable interrupts
`ExitTask()`	cause a task to terminate
`LockTask()`	suspend normal task switching
`Pause()`	pause a task without relinquishing control
`ResumeTask()`	cause a waiting task to resume execution
`RunTask()`	cause a specific task to run
`Semaphore()`	coordinate the use of a resource
`SetInterval()`	set the interval timer period
`Sleep()`	pause a task
`StackCheck()`	enable or disable stack checking
`StatusTask()`	determine the status of a task
`System.TaskHeadRoom()`	determine the unused space in a task's stack
`TaskIsLocked()`	determine if a task is locked
`TaskIsValid()`	determine if a task stack is in the task list
`UnlockTask()`	resume normal task switching
`UpdateRTC()`	update RTC registers to account for missed ticks
`WaitForInterrupt()`	pause a task until an external event occurs
`WaitForInterval()`	pause a task until an interval timer expires
`Yield()`	allow another task to run

Miscellaneous Routines

`CloseWatchDog()`	deactivate the watchdog timer
`CPUSleep()`	cause the CPU to go into sleep mode
`CRC16()`	compute a 16-bit CRC value
`CRC32()`	compute a 32-bit CRC value
`FirstTime()`	determine if this is the first the program has been run since downloading
`GetMicroTime()`	populate a buffer with higher precision timing information
`GetElapsedMicroTime()`	determine the elapsed time relative to previous time information
`IIf()`	select the value of one of two expressions
`LBound()`	determine the lower bound of an array
`LongJmp()`	perform a non-local goto (e.g. for exception handling)
`NoOp()`	execute a “nop” instruction
`OpenWatchDog()`	activate the watchdog timer
`ParityCheck()`	check the parity of a data byte
`Randomize()`	initialize the random number generator
`ResetProcessor()`	reset the CPU
`Rnd()`	retrieve the next random number
`SerialNumber()`	retrieve the system software serial number
`SetJmp()`	prepare for a non-local Goto (e.g. exception handling)
`SizeOf()`	determine the size of a data item
`SizeOfU()`	determine the size of a data item
`Span()`	determine the number of elements in an array dimension
`System.DeviceID()`	retrieve the identification characters for the device
`UBound()`	determine the upper bound of an array
`WatchDog()`	reset the watchdog timer
`ZXCmdMode()`	activate the “command mode” (for downloading)