The PBClone Windows Library

The PBClone Windows Library

Version 1.1

This is PBCwin, a general-purpose library of 79 routines for use with Visual Basic for Windows

Type Conversion

Swap Values

Maximum and Minimum Values

Bit Twiddling

Character Tests

Integer Tests

Odd
Odd Long

Compressed Time/Date Manipulation

Array Initialization

Peek and Poke Memory

Input/Output

Miscellaneous

Binary Integer

Function: BinI

This function converts a binary value, passed to it as a string, to an integer. It stops the conversion on reaching the end of the string or at the first character that is not a valid binary digit ("0" or "1").

See also BinL, which returns a long integer value.

Form:

Result%= BinI(St$)

Arguments:

St$:
   binary number, in string form

Result:

Result%: integer equivalent of binary number

Binary Long

Function: BinL

This function converts a binary value, passed to it as a string, to a long integer. It stops the conversion on reaching the end of the string or at the first character that is not a valid binary digit ("0" or "1").

See also BinI, which returns an integer value.

Form:

Result& = BinL(St$)

Arguments:

St$:
binary number, in string form

Result:

Result&: long integer equivalent of binary number

Bytes to Integer

Function: Bytes2Int

This function combines two bytes, contained in separate integers, into a single integer value.

See also HiByte and LoByte, which may be used to reverse the process, splitting an integer into two bytes.

Form:

Result% = Bytes2Int(Lo%:, Hi%:)

Arguments:

Lo%:
   low, or least significant, byte
Hi%:
   high, or most significant, byte

Result:

Result%: result of combining bytes into an integer

Checksum

Function: Checksum

This function calculates an 8-bit checksum for a string.   The result is compatible with Xmodem and Ymodem file transfer protocols, and can be used as a fast and simple check of data validity.   For more rigorous testing, see CRC16.

Form:

Result% = Checksum(St$, Bytes%)

Arguments:

St$:
   string for which to calculate checksum
Bytes%:
   number of characters for which to calculate checksum

Result:

Result%: checksum of specified part of string

Comm Ports

Function: ComPorts

This function returns the number of communications (serial) ports installed.

Form:
Result% = ComPorts()

Result:

Result%: comm ports (0-3)

CRC 16-bit

Function: CRC

This function calculates a 16-bit "cyclical redundancy check" checksum, or CRC, for a string. The result is compatible with Xmodem and Ymodem file transfer protocols, and can be used as a check of data validity.

Note that the Xmodem and Ymodem file transfer protocols use a different byte ordering method than typical of Intel machines. If you intend to use this function in writing file transfer protocols, you will need to reverse the byte order to MSB first, LSB second. This can be accomplished with either LRotateI or RRotateI with a shift count of 8 (eight), or by splitting the integer into bytes with LoByte and HiByte, and swapping the results.

Form:

Result% = CRC16(St$, Bytes%)

Arguments:

St$:
   string for which to calculate CRC
Bytes%:
   number of characters for which to calculate CRC

Result:

Result%: CRC of specified part of string

Date Squeeze

Function: DateSq

This function compresses a date into a single integer.   This provides a very efficient storage format for dates ranging from January 1, 1900 to December 31, 2028.
Uncompression is done with DayUnsq, MonthUnsq, and YearUnsq. See also TimeSq, which allows you to compress a time value similarly.

Note that compressed dates are not in a format that may be readily used for comparison or date math purposes.   If you need such capabilities, convert the date to a BASIC time/date serial number first-- see your BASIC manual for details.

If you pass a year of 0-99, it will be translated to 1900-1999 before the compression is done.   Depending on your application, you may wish to assume 0-28 is the same as 2000-2028 instead. If so, make sure you do an explicit conversion before this function is called.

Form:

Result% = DateSq(MonthNr%, DayNr%, YearNr%)

Arguments:

MonthNr%:
   month number (1-12)
DayNr%:
   day number (1-31)
YearNr%:
   year number (1900-2028)

Result:

Result%: compressed date

Day Unsqueeze

Function: DayUnsq

This function returns the day from a compressed date. It works in conjunction with the DateSq date compression function.

Form:

DayNr% = DayUnsq(Number%)

Arguments:

Number%:
compressed date

Result:

DayNr%: day number

Floppies

Function: Floppies

This function returns the number of floppy disk drives installed, up to two. Although it is possible to have up to four floppy drives, the PC was designed to expect a maximum of two, and this routine can't tell if there are more than that.

Form:

Result% = Floppies()

Arguments:

none

Result:

Result%:
  floppy drives (0-2)

Get Comm Address

Function: GetComAddr

This function returns the I/O base port address for a specified communications (serial) port. If there is no such serial port, or if the port is in use, a zero will be returned.

Form:

Address% = GetComAddr(PortNr%)

Arguments:

PortNr%:
   communications port number (0-3)

Result:

Address%:
   I/O base port address for comm port

Get Port Address

Function: GetPortAddr

This function returns the I/O base port address for a specified printer (parallel) port. If there is no such parallel port, a zero will be returned.

Form:

```
Address% = GetPrtAddr(PortNr%)
```

Arguments:

PortNr%:
  printer port number (0-3)

Result:

Address%:
  I/O base port address for printer port

Get Clock Tick

Function: GetTick

This function returns the current system time count.   The count is the amount of time after midnight, in (approximately) 1/18th seconds.   This can be used as a fairly high- resolution timer.

DO NOT use this function to write a delay routine!   That would eat precious system time that could be more profitably used by other programs while your program is idle-- remember, Windows is a multitasking environment.   If you need a delay, use the SLEEP statement provided by BASIC.

Form:

Result& = GetTick()

Arguments:

none

Result:

Result&:
   system timer tick

High Byte

Function: HiByte

This function returns the high, or most significant, byte of an integer.
See also Bytes2Int, which can be used to reverse the process.

Form:

```
Byte% = HiByte(Number%)
```

Arguments:

Number%:
   number from which to pick high byte

Result:

Byte%:
   high byte of number

High Nybble

Function: HiNybble

This function returns the high, or most significant, nybble of a byte.
See also Nybs2Byte, which can be used to reverse the process.

Form:

Nybble% = HiNybble(Number%)

Arguments:

Number%:
byte from which to pick high nybble

Result:

Nybble%:
high nybble of byte

High Word

Function: HiWord

This function returns the high, or most significant, word of a long integer.
See also Ints2Long, which can be used to reverse the process.

Form:

```
Word% = HiWord(Number&)
```

Arguments:

Number&:
   long integer from which to pick high word

Result:

Word%:
   high word of long integer

Hour Unsqueeze

Function: HourUnsq

This function returns the hour from a compressed time. It works in conjunction with the TimeSq time compression function.

Form:

HourNr% = HourUnsq(Number%)

Arguments:

Number%:
compressed time

Result:

HourNr%:
hour number

Integers to Long

Function: Ints2Long

This function combines two integers into a single long integer value.

See also HiWord and LoWord, which may be used to reverse the process, splitting a long integer into two integers.

Form:

```
Result& = Ints2Long(Lo%, Hi%)
```

Arguments:

Lo%:
   low, or least significant, word
Hi%:
   high, or most significant, word

Result:

Result& Result of combining integers into a long

Is AlphaNumeric

Function: IsAlNum

This function tells you whether a character is alphanumeric, that is, either a letter of the alphabet or a digit.   It operates on the first character of a string you pass it.

Form:

Result% = IsAlNum(St$)

Arguments:

St$:
   character to test

Result:

Result%:
   whether char is alphanumeric (-1 yes, 0 no)

Is ASCII

Function: IsASCII

This function tells you whether a character is a member of the ASCII character set.   It operates on the first character of a string you pass it.

Form:

Result% = IsASCII(St$)

Arguments:

St$:
   character to test

Result:

Result%:
   whether char is ASCII (-1 yes, 0 no)

Is Alphabetic

Function: IsAlpha

This function tells you whether a character is alphabetic.   It operates on the first character of a string you pass it.

Form:

Result% = IsAlpha(St$)

Arguments:

St$:
   character to test

Result:

Result%:
   whether char is alphabetic (-1 yes, 0 no)

Is Control

Function: IsControl

Arguments:

St$:
   character to test

Result:

Result%:
   whether char is a control code (-1 yes, 0 no)

Is Digit

Function: IsDigit

This function tells you whether a character is a digit. It operates on the first character of a string you pass it.

Form:

Result% = IsDigit(St$)

Arguments:

St$:
   character to test

Result:

Result%:
   whether char is a digit (-1 yes, 0 no)

Is Lowercase

Function: IsLower

This function tells you whether a character is a lowercase letter ("a" through "z").   It operates on the first character of a string you pass it.

Form:

Result% = IsLower(St$)

Arguments:

St$:
   character to test

Result:

Result%:
   whether char is lowercase (-1 yes, 0 no)

Is Punctuation

Function: IsPunct

This function tells you whether a character may be construed as punctuation.   This includes the space and most symbols. This function operates on the first character of a string you pass it.

Form:

Result% = IsPunct(St$)

Arguments:

St$:
   character to test

Result:

Result%:
   whether char is punctuation (-1 yes, 0 no)

Is Space

Function: IsSpace

This function tells you whether a character is "white space" (ASCII 9-13 and 32, including tab, linefeed, formfeed, carriage return, and space).   It operates on the first character of a string you pass it.

Form:

Result% = IsSpace(St$)

Arguments:

St$:
   character to test

Result:

Result%:
   whether char is white space (-1 yes, 0 no)

Is Uppercase

Function: IsUpper

This function tells you whether a character is an uppercase letter ("A" through "Z").   It operates on the first character of a string you pass it.

Form:

```
Result% = IsUpper(St$)
```

Arguments:

St$:
   character to test

Result:

Result%:
   whether char is uppercase (-1 yes, 0 no)

Is heX Digit

Function: IsXDigit

This function tells you whether a character is a hexadecimal digit. This includes 0-9, a-z, and A-Z. This function operates on the first character of a string you pass it.

Form:

Result% = IsXDigit(St$)

Arguments:

St$:
   character to test

Result:

Result%:
   whether char is a hex digit (-1 yes, 0 no)

Low Byte

Function: LoByte

This function returns the low, or least significant, byte of an integer.
See also Bytes2Int, which can be used to reverse the process.

Form:

```
Byte% = LoByte(Number%)
```

Arguments:

Number%:
   number from which to pick low byte

Result:

Byte%:
   low byte of number

Low Nybble

Function: LoNybble

This function returns the low, or least significant, nybble of a byte.
See also Nybs2Byte, which can be used to reverse the process.

Form:

```
Nybble% = LoNybble(Number%)
```

Arguments:

Number%:
   byte from which to pick low nybble

Result:

Nybble%:
   low nybble of byte

Low Word

Function: LoWord

This function returns the low, or least significant, word of a long integer.
See also Ints2Long, which can be used to reverse the process.

Form:

Word% = LoWord(Number&)

Arguments:

Number&:
  long integer from which to pick low word

Result:

Word%:
  low word of long integer

Left Rotate Integer

Function: LRotateI

This function returns the result of rotating an integer left by a specified number of bits.

Form:

Result% = LRotateI(Number%, Count%)

Arguments:

Number%:
number to rotate
Count%:
number of bits by which to rotate

Result:

Result%:
rotated number

Left Rotate Long

Function: LRotateL

This function returns the result of rotating a long integer left by a specified number of bits.

Form:

Result& = LRotateL(Number&, Count%)

Arguments:

Number&:
   number to rotate
Count%:
   number of bits by which to rotate

Result:

Result&:
   rotated number

Left Shift Integer

Function: LShiftI

This function returns the result of shifting an integer left by a specified number of bits.

Form:

Result% = LShiftI(Number%, Count%)

Arguments:

Number%:
  number to shift
Count%:
  number of bits by which to shift

Result:

Result%:
  shifted number

Left Shift Long

Function: LShiftL

This function returns the result of shifting a long integer left by a specified number of bits.

Form:

```
Result& = LShiftL(Number&, Count%)
```

Arguments:

Number&:
   number to shift
Count%:
   number of bits by which to shift

Result:

Result&:
   shifted number

Maximum Double-precision

Function: MaxD

This function returns the larger of two double-precision numbers.

Form:

Result# = MaxD(Nr1#, Nr2#)

Arguments:

Nr1#:
   first number
Nr2#:
   second number

Result:

Result#:
   larger of the two numbers

Maximum Integer

Function: MaxI

This function returns the larger of two integers.

Form:

Result% = MaxI(Nr1%, Nr2%)

Arguments:

Nr1%:
  first number
Nr2%:
  second number

Result:

Result%:
  larger of the two numbers

Maximum Long

Function: MaxL

This function returns the larger of two long integers.

Form:

Result& = MaxL(Nr1&, Nr2&)

Arguments:

Nr1&:
  first number
Nr2&:
  second number

Result:

Result&:
  larger of the two numbers

Maximum Single-precision

Function: MaxS

This function returns the larger of two single-precision numbers.

Form:

Result! = MaxS(Nr1!, Nr2!)
Arguments:

Nr1#:
   first number
Nr2#:
   second number

Result:

Result#:
   smaller of the two numbers

Minimum Double-precision

Function: MinD

This function returns the smaller of two double-precision numbers.

Form:

Result# = MinD(Nr1#, Nr2#)

Arguments:

Nr1#:
   first number
Nr2#:
   second number

Result:

Result#:
   larger of the two numbers

Minimum Integer

MinI   (Minimum Integer)

This function returns the smaller of two integers.

Form:

Result% = MinI(Nr1%, Nr2%)

Arguments:

Nr1%:
   first number
Nr2%:
   second number

Result:

Result%:
   smaller of the two numbers

Minimum Long

Function: MinL

This function returns the smaller of two long integers.

Form:

Result& = MinL(Nr1&, Nr2&)

Arguments:

Nr1&:
  first number
Nr2&:
  second number

Result:

Result&:
  smaller of the two numbers

Minimum Single-precision

Function: MinS

This function returns the larger of two single-precision numbers.

Form:

Result! = MinS(Nr1!, Nr2!)
Arguments:

Nr1#:
   first number
Nr2#:
   second number

Result:

Result#:
   smaller of the two numbers

Minute Unsqueeze

Function: MinuteUnsq

This function returns the minute from a compressed time. It works in conjunction with the TimeSq time compression function.

Form:

```
MinuteNr% = MinuteUnsq(Number%)
```

Arguments:

Number%:
  compressed time

Result:

MinuteNr%:
  minute number

Month Unsqueeze

Function: MonthUnsq

This function returns the month from a compressed date. It works in conjunction with the DateSq date compression function.

Form:

```
MonthNr% = MonthUnsq(Number%)
```

Arguments:

Number%:
   compressed date

Result:

MonthNr%:
   month number

Nybbles to Byte

Function: Nybs2Byte

This function combines two nybbles into a single byte value. Since BASIC supports neither byte nor nybble data types, the values are all kept in integers.
See also HiNybble and LoNybble, which may be used to reverse the process, splitting a byte into two nybbles.

Form:

Result% = Nybs2Byte(Lo%, Hi%)

Arguments:

Lo%:
   low, or least significant, nybble
Hi%:
   high, or most significant, nybble

Result:

Result%:
   Result of combining nybbles into a byte

Odd

Function: Odd

This function tells you whether an integer is an odd number.

Form:

Result% = Odd(Number%)

Arguments:

Number%:
   number to test

Result:

Result%:
   whether number is odd (-1 yes, 0 no)

Odd Long

Function: OddL

This function tells you whether a long integer is an odd number.

Form:

Result% = OddL(Number&)

Arguments:

Number&:
   number to test

Result:

Result%:
   whether number is odd (-1 yes, 0 no)

PBCwin Version

Function: PBCwinVer

This function returns the version of PBCwin available. You can use this to make sure the PBCWIN.DLL being used is sufficiently current to handle the routines you need. Don't check the exact version number, since it should be ok for the user to have a newer version of PBCwin than your program expects. Instead, make sure that the returned version number is greater than or equal to the version you expect.

The version number is multiplied by 100 so it can be returned as an integer. For example, PBCwin v1.0 returns a result of 100 here. PBCwin v1.1 will return 110, and so on.

Form:

```
Version% = PBCwinVer()
```

Arguments:

none

Result:

Version%:
  PBCWIN.DLL version times 100

Peek Byte

Function: PeekB

This routine gets a byte from a specified memory location. The memory location is specified as a pointer, which is a combined segment and offset value.   The VarPtr function can be used to get a pointer to a variable.   If you want to create a pointer to an address for which you know the segment and offset, you can use the Ints2Long function to do so, by loading the segment into the high word and offset into the low word.

Form:

Nr% = PeekB(Ptr&)

Arguments:

Ptr&:
   far pointer

Result:

Nr%:
   byte retrieved from memory

Peek Integer

Function: PeekI

This routine gets an integer from a specified memory location. The memory location is specified as a pointer, which is a combined segment and offset value.   The VarPtr function can be used to get a pointer to a variable.   If you want to create a pointer to an address for which you know the segment and offset, you can use the Ints2Long function to do so, by loading the segment into the high word and offset into the low word.

Form:

Nr% = PeekI(Ptr&)

Arguments:

Ptr&:
   far pointer

Result:

Nr%:
   integer retrieved from memory

Peek Long

Function: PeekL

This routine gets a long integer from a specified memory location. The memory location is specified as a pointer, which is a combined segment and offset value. The VarPtr function can be used to get a pointer to a variable. If you want to create a pointer to an address for which you know the segment and offset, you can use the Ints2Long function to do so, by loading the segment into the high word and offset into the low word.

Form:

```
Nr& = PeekL(Ptr&)
```

Arguments:

Ptr&:
far pointer

Result:

Nr&:
long integer retrieved from memory

Poke Byte

Function: PokeB

This routine pokes a byte into a specified memory location. The memory location is specified as a pointer, which is a combined segment and offset value.   The VarPtr function can be used to get a pointer to a variable.   If you want to create a pointer to an address for which you know the segment and offset, you can use the Ints2Long function to do so, by loading the segment into the high word and offset into the low word.

Form:

PokeB Ptr&, Nr%

Arguments:

Ptr&:
   far pointer
Nr%:
   byte to place in memory at the pointer location

Result:

see above

Poke Integer

Function: PokeI

This routine pokes an integer into a specified memory location. The memory location is specified as a pointer, which is a combined segment and offset value.   The VarPtr function can be used to get a pointer to a variable.   If you want to create a pointer to an address for which you know the segment and offset, you can use the Ints2Long function to do so, by loading the segment into the high word and offset into the low word.

Form:

PokeI Ptr&, Nr%

Arguments:

Ptr&:
   far pointer
Nr%:
   integer to place in memory at the pointer location

Result:

See above.

Poke Long

Function: PokeL

This routine pokes a long integer into a specified memory location. The memory location is specified as a pointer, which is a combined segment and offset value. The VarPtr function can be used to get a pointer to a variable. If you want to create a pointer to an address for which you know the segment and offset, you can use the Ints2Long function to do so, by loading the segment into the high word and offset into the low word.

Form:

PokeL Ptr&, Nr&

Arguments:

Ptr&:
   far pointer
Nr%:
   long integer to place in memory at the pointer posn

Result:

See above.

Power of 2 Integer

Function: Power2I

This function returns the result of raising 2 (two) to the specified power. It's much faster than using the floating-point raise-to-power operator in BASIC, and is especially handy for bit twiddling.

Form:

```
Result% = Power2I(Power%)
```

Arguments:

Power%:
   power to which to raise two

Result:

Result%:
   two to the specified power

Power of 2 Long

Function: Power2L

This function returns the result of raising 2 (two) to the specified power.   It's much faster than using the floating-point raise-to-power operator in BASIC, and is especially handy for bit twiddling.

Form:

Result& = Power2L(Power%)

Arguments:

Power%:
   power to which to raise two

Result:

Result&:
   two to the specified power

Printer Ports

Function: PrtPorts

This function returns the number of printer (parallel) ports installed.

Form:

```
Result% = PrtPorts()
```

Arguments:

none

Result:

Result%:
  printer ports (0-3)

Pointer Array Double-precision

Function: PtrMatD

This routine initializes each element of a double-precision array to an increasingly large value.   The value starts at a specified beginning and is incremented by one for each subsequent element.

Form:

PtrMatD VarPtr(Array#(FirstElem)), Elements%, InitValue#

Arguments:

Array#(FirstElem):
   first element of array to initialize
Elements%:
   number of elements to initialize
InitValue#:
   value to which to init first array element

Result:

Array#(FirstElem thru FirstElem + Elements% - 1) are initialized

Pointer Array Integer

Function: PtrMatI

This routine initializes each element of an integer array to an increasingly large value.   The value starts at a specified beginning and is incremented by one for each subsequent element.

Form:

```
PtrMatI VarPtr(Array%(FirstElem)), Elements%, InitValue%
```

Arguments:

Array%(FirstElem):
   first element of array to initialize
Elements%:
   number of elements to initialize
InitValue%:
   value to which to init first array element

Result:

Array%(FirstElem thru FirstElem + Elements% - 1) are initialized

Pointer Array Long

Function: PtrMatL

This routine initializes each element of a long integer array to an increasingly large value.   The value starts at a specified beginning and is incremented by one for each subsequent element.

Form:

PtrMatL VarPtr(Array&(FirstElem)), Elements%, InitValue&

Arguments:

Array&(FirstElem):
   first element of array to initialize
Elements%:
   number of elements to initialize
InitValue&:
   value to which to init first array element

Result:

Array&(FirstElem thru FirstElem + Elements% - 1) are initialized

Pointer Array Single-precision

Function: PtrMatS

This routine initializes each element of a single-precision array to an increasingly large value. The value starts at a specified beginning and is incremented by one for each subsequent element.

Form:

```
PtrMatS VarPtr(Array!(FirstElem)), Elements%, InitValue!
```

Arguments:

Array!(FirstElem):
   first element of array to initialize
Elements%:
   number of elements to initialize
InitValue!:
   value to which to init first array element

Result:

Array!(FirstElem thru FirstElem + Elements% - 1) are initialized

Right Rotate Integer

Function: RRotateI

This function returns the result of rotating an integer right by a specified number of bits.

Form:

Result% = RRotateI(Number%, Count%)

Arguments:

Number%:
  number to rotate
Count%:
  number of bits by which to rotate

Result:

Result%:
  rotated number

Right Rotate Long

Function: RRotateL

This function returns the result of rotating a long integer right by a specified number of bits.

Form:

Result& = RRotateL(Number&, Count%)

Arguments:

Number&:
  number to rotate
Count%:
  number of bits by which to rotate

Result:

Result&:
  rotated number

Right Shift Integer

Function: RShiftI

This function returns the result of shifting an integer right by a specified number of bits.

Form:

Result% = RShiftI(Number%, Count%)

Arguments:

Number%:
number to shift
Count%:
number of bits by which to shift

Result:

Result%:
shifted number

Right Shift Long

Function: RShiftL

This function returns the result of shifting a long integer right by a specified number of bits.

Form:

Result& = RShiftL(Number&, Count%)

Arguments:

Number&:
  number to shift
Count%:
  number of bits by which to shift

Result:

Result&:
  shifted number

Second Unsqueeze

Function: SecondUnsq

This function returns the second from a compressed time. It works in conjunction with the TimeSq time compression function.

Note that the second value will always be even, due to the limited amount of information that can be squeezed into an integer.

Form:

SecondNr% = SecondUnsq(Number%)

Arguments:

Number%:
  compressed time

Result:

SecondNr%:
  second number

Set Array Currency

Function: SetMatC

This routine initializes each element of a currency array to a specified value.

Form:

SetMatC VarPtr(Array@(FirstElem)), Elements%, Value@

Arguments:

Array@(FirstElem):
first element of array to initialize
Elements%:
number of elements to initialize
Value@: value to which to initialize array

Result:

Array@(FirstElem thru FirstElem + Elements% - 1) are initialized

Set Array Double-precision

Function: SetMatD

This routine initializes each element of a double-precision array to a specified value.

Form:

```
SetMatD VarPtr(Array#(FirstElem)), Elements%, Value#
```

Arguments:

Array#(FirstElem):
   first element of array to initialize
Elements%:
   number of elements to initialize
Value#:
   value to which to initialize array

Result:

Array#(FirstElem thru FirstElem + Elements% - 1) are initialized

Set Array Integer

Function: SetMatI

This routine initializes each element of an integer array to a specified value.

Form:

```
SetMatI VarPtr(Array%(FirstElem)), Elements%, Value%
```

Arguments:

Array%(FirstElem):
   first element of array to initialize
Elements%:
   number of elements to initialize
Value%:
   value to which to initialize array

Result:

Array%(FirstElem thru FirstElem + Elements% - 1) are initialized

Set Array Long

Function: SetMatL

This routine initializes each element of a long integer array to a specified value.

Form:

SetMatL VarPtr(Array&(FirstElem)), Elements%, Value&

Arguments:

Array&(FirstElem):
   first element of array to initialize
Elements%:
   number of elements to initialize
Value&:
   value to which to initialize array

Result:

Array&(FirstElem thru FirstElem + Elements% - 1) are initialized

Set Array Single-precision

Function: SetMatS

This routine initializes each element of a single-precision array to a specified value.

Form:

SetMatS VarPtr(Array!(FirstElem)), Elements%, Value!

Arguments:

Array!(FirstElem):
first element of array to initialize
Elements%:
number of elements to initialize
Value!:
value to which to initialize array

Result:

Array!(FirstElem thru FirstElem + Elements% - 1) are initialized

Swap Currency

Function: SwapC

This routine swaps two currency values.

Form:

SwapC Number1@, Number2@

Arguments:

Number1@:	first number
Number2@:	second number

Result:

Number1@:	former second number
Number2@:	former first number

Swap Double-precision

Function: SwapD

This routine swaps two double-precision numbers.

Form:

```
SwapD Number1#, Number2#
```

Arguments:

Number1#:
  first number
Number2#:
  second number

Result:

Number1#:
  former second number
Number2#:
  former first number

Swap Integer

Function: SwapI

This routine swaps two integers.

Form:

SwapI Number1%, Number2%

Arguments:

Number1%:
  first number
Number2%:
  second number

Result:

Number1%:
  former second number
Number2%:
  former first number

Swap Long

Function: SwapL

This routine swaps two long integers.

Form:

SwapL Number1&, Number2&

Arguments:

Number1&:
  first number
Number2&:
  second number

Result:

Number1&:
  former second number
Number2&:
  former first number

Swap Single-precision

Function: SwapS

This routine swaps two single-precision numbers.

Form:

SwapS Number1!, Number2!

Arguments:

Number1!:
  first number
Number2!:
  second number

Result:

Number1!:
  former second number
Number2!:
  former first number

Time Squeeze

Function: TimeSq

This function compresses a time into a single integer.   This provides a very efficient storage format. Note, however, that an integer is not quite large enough to store an exact time-- the seconds value, if odd, will be rounded down to the next closest even number.

Uncompression is done with HourUnsq, MinuteUnsq, and SecondUnsq. See also DateSq, which allows you to compress a date value similarly.

Note that compressed times are not in a format that may be readily used for comparison or time math purposes.   If you need such capabilities, convert the time to a BASIC time/date serial number first-- see your BASIC manual for details.

Form:

Result% = TimeSq(HourNr%, MinuteNr%, SecondNr%)

Arguments:

HourNr%:
   hour number (0-23)

MinuteNr%:
   minute number (0-59)

SecondNr%:
   second number (0-59; see note on truncation)

Result:

Result%:
   compressed time

Variable Pointer

Function: VarPtr

This function returns a far pointer to a variable.   It works with any variable type except BASIC strings, which are stored in an unusual format.   This function is required for passing arrays to the PBCwin routines which take arrays as parameters. It has not been tested with VB/Win 2.0 arrays, and is likely not to work with such arrays if they're over 64k bytes.   Since BASIC arrays may move in memory, it is advised that you get the pointer to an array just before using it, to minimize the risk of accessing the wrong area of memory.

This function can also be used to provide a pointer for use with the various Peek_ and Poke_ routines in PBCwin.

Note that the PBCwin function, VarPtr, is not identical to the DOS BASIC function, VARPTR.   It returns an absolute address consisting of both segment and offset, rather than merely an offset value.

Form:

Ptr& = VarPtr(Vbl)

Arguments:

Vbl:
   variable for which to get a pointer

Result:

Ptr&:
   far pointer to variable

Year Unsqueeze

Function: Function: YearUnsq

This function returns the year from a compressed date.   It works in conjunction with the DateSq date compression function.

Form:

YearNr% = YearUnsq(Number%)

Arguments:

none

Number%:
   compressed date

Result:

YearNr%:
   year number