DECLARE

Syntax
[DECLARE] *code modifying directive = modifying value*

Overview
Adjust certain aspects of the produced code, i.e. Crystal frequency, LCD port and pins, serial baud rate etc.

Operators
code modifying directive is a set of pre-defined words. See list below.
modifying value is the value that corresponds to the command. See list below.

The **DECLARE** directive is an indispensable part of the compiler. It moulds the library subroutines, and passes essential user information to them. However, the **DECLARE** part of a declare directive is optional.

For example, instead of using: -

DECLARE XTAL 4

The text: -

XTAL = 4

May be used.

Notice that there is an optional equals character separating the declare command and the value to pass. The structure will still be referred to as a **DECLARE** in the manual, help file, and any future projects.

MISC Declares.

DECLARE WATCHDOG = ON or OFF, or TRUE or FALSE, or 1, 0

The **WATCHDOG DECLARE** directive enables or disables the watchdog timer. It also sets the PICmicro's CONFIG fuses for no watchdog. In addition, it removes any **CLRWDT** mnemonics from the assembled code, thus producing slightly smaller programs. The default for the compiler is **WATCHDOG OFF**, therefore, if the watchdog timer is required, then this **DECLARE** will need to be invoked.

The **WATCHDOG DECLARE** can be issued multiple times within the BASIC code, enabling and disabling the watchdog timer as and when required.

DECLARE BOOTLOADER = ON or OFF, or TRUE or FALSE, or 1, 0

The **BOOTLOADER DECLARE** directive enables or disables the special settings that a serial bootloader requires at the start of code space. This directive is ignored if a PICmicro without bootloading capabilities is targeted.

Disabling the bootloader will free a few bytes from the code produced. This doesn't seem a great deal, however, these few bytes may be the difference between a working or non-working program. The default for the compiler is **BOOTLOADER ON**

DECLARE ICD = ON or OFF, or TRUE or FALSE, or 1, 0

When using an ICD (In-Circuit-Developer), the very first word in code space. i.e. location 0, needs to be a **NOP** instruction. The **ICD DECLARE** enables or disables the creation of this **NOP** within the program.

Note that if the **BOOTLOADER DECLARE** is used, then the **NOP** is automatically inserted into the code, thus circumventing the need to use of the **ICD DECLARE**. However, no harm will be caused if both the **BOOTLOADER**, and **ICD** declares are used in the same program.

DECLARE SHOW_SYSTEM_VARIABLES = ON or OFF, or TRUE or FALSE, or 1, 0

When using the PROTEUS VSM to simulate BASIC code, it is sometimes beneficial to observe the behaviour of the compiler's SYSTEM variables that are used for its library routines. The **SHOW_SYSTEM_VARIABLES DECLARE** enables or disables this option.

DECLARE FSR_CONTEXT_SAVE = ON or OFF, or TRUE or FALSE, or 1, 0

When using HARDWARE interrupts, it is not always necessary to save the FSR register. So in order to save code space and time spent within the interrupt handler, the **FSR_CONTEXT_SAVE DECLARE** can enable or disable the auto CONTEXT saving and restoring of the FSR register. For 16-bit core devices, this will enable/disable FSR0 context handling.

DECLARE PLL_REQ = ON or OFF, or TRUE or FALSE, or 1, 0

Most 16-bit core devices have a built in PLL (Phase Locked Loop) that can multiply the oscillator by a factor of 4. This is set by the fuses at programming time, and the **PLL_REQ DECLARE** enables or disables the PLL fuse. Using the PLL fuse allows a 1:1 ratio of instructions to clock cycles instead of the normal 4:1 ratio. It can be used with XTAL settings from 4 to 10MHz. Note that the compiler will automatically set it's frequency to a multiple of 4 if the **PLL_REQ DECLARE** is used to enable the PLL fuse. For example, if a 4MHz XTAL setting is declared, and the **PLL_REQ DECLARE** is used in the BASIC program, the compiler will automatically set itself up as using a 16MHz XTAL. i.e. 4 * 4. Thus keeping the timings for library functions correct.

DECLARE WARNINGS = ON or OFF, or TRUE or FALSE, or 1, 0

The **WARNINGS DECLARE** directive enables or disables the compiler's warning messages. This can have disastrous results if a warning is missed or ignored, so use this directive sparingly, and at your own peril.

The **WARNINGS DECLARE** can be issued multiple times within the BASIC code,

enabling and disabling the warning messages at key points in the code as and when required.

DECLARE REMINDERS = **ON** or **OFF**, or **TRUE** or **FALSE**, or 1, 0
The **REMINDERS DECLARE** directive enables or disables the compiler's reminder messages. The compiler issues a reminder for a reason, so use this directive sparingly, and at your own peril.

The **REMINDERS DECLARE** can be issued multiple times within the BASIC code, enabling and disabling the warning messages at key points in the code as and when required.

DECLARE LABEL_BANK_RESETS = **ON** or **OFF**, or **TRUE** or **FALSE**, or 1, 0
The compiler has very intuitive RAM bank handling, however, if you think that an anomaly is occuring due to misplaced or mishandled RAM bank settings, you can issue this **DECLARE** and it will reset the RAM bank on every BASIC label, which will force the compiler to re-calculate its bank settings. If nothing else, it will reasure you that bank handling is not the cause of the problem, and you can get on with finding the cause of the programming problem. However, if it does cure a problem then please let me know and I will make sure the anomaly is fixed as quickly as possible.

Using this **DECLARE** will increase the size of the code produced, as it will place **BCF** mnemonics in the case of a 12 or 14-bit core device, and a **MOVLB** mnemonic in the case of a 16-bit core device.

The **LABEL_BANK_RESETS DECLARE** can be issued multiple times within the BASIC code, enabling and disabling the bank resets at key points in the code as and when required. See LINE LABELS for more information.

TRIGONOMETRY Declares.

When using a 16-bit core device, the compiler defaults to using floating point trigonometry functions SIN and COS, as well as SQR. However, if only the BASIC Stamp compatible integer functions are required, they can be enabled by the following three declares. Note that by enabling the integer type function, the floating point function will be disabled permanently within the BASIC code. As with most of the declares, only one of any type is recognised per program.

DECLARE STAMP_COMPATIBLE_COS = **ON** or **OFF**, or **TRUE** or **FALSE**, or 1, 0
Enable/Disable floating point COS function in favour of the BASIC Stamp compatible integer COS function.

DECLARE STAMP_COMPATIBLE_SIN = **ON** or **OFF**, or **TRUE** or **FALSE**, or 1, 0
Enable/Disable floating point SIN function in favour of the BASIC Stamp compatible integer SIN function.

DECLARE STAMP_COMPATIBLE_SQR = **ON** or **OFF**, or **TRUE** or **FALSE**, or 1, 0

Enable/Disable floating point SQR (square root) function in favour of the BASIC Stamp compatible integer SQR function.

ADIN Declares.

DECLARE ADIN_RES 8 , 10 , or 12.
Sets the number of bits in the result.

If this **DECLARE** is not used, then the default is the resolution of the PICmicro type used. For example, the new 16F87X range will result in a resolution of 10-bits, while the standard PICmicro types will produce an 8-bit result. Using the above **DECLARE** allows an 8-bit result to be obtained from the 10-bit PICmicro types, but NOT 10-bits from the 8-bit types.

DECLARE ADIN_TAD 2_FOSC , 8_FOSC , 32_FOSC , or FRC.
Sets the ADC's clock source.

All compatible PICmicros have four options for the clock source used by the ADC; 2_FOSC, 8_FOSC, and 32_FOSC, are ratios of the external oscillator, while FRC is the PICmicro's internal RC oscillator. Instead of using the predefined names for the clock source, values from 0 to 3 may be used. These reflect the settings of bits 0-1 in register **ADCON0**.

Care must be used when issuing this **DECLARE**, as the wrong type of clock source may result in poor resolution, or no conversion at all. If in doubt use FRC which will produce a slight reduction in resolution and conversion speed, but is guaranteed to work first time, every time. FRC is the default setting if the **DECLARE** is not issued in the BASIC listing.

DECLARE ADIN_STIME 0 to 65535 microseconds (us).
Allows the internal capacitors to fully charge before a sample is taken. This may be a value from 0 to 65535 microseconds (us).

A value too small may result in a reduction of resolution. While too large a value will result in poor conversion speeds without any extra resolution being attained.

A typical value for **ADIN_STIME** is 50 to 100. This allows adequate charge time without loosing too much conversion speed.

But experimentation will produce the right value for your particular requirement. The default value if the **DECLARE** is not used in the BASIC listing is 50.

BUSIN - BUSOUT Declares.

DECLARE SDA_PIN PORT **.** PIN
Declares the port and pin used for the data line (SDA). This may be any valid port on the PICmicro. If this declare is not issued in the BASIC program, then the default Port and Pin is PORTA.0

DECLARE SCL_PIN PORT **.** PIN
Declares the port and pin used for the clock line (SCL). This may be any valid port on the PICmicro. If this declare is not issued in the BASIC program, then the default Port and Pin is PORTA.1

DECLARE SLOW_BUS ON - **OFF** or 1 - 0
Slows the bus speed when using an oscillator higher than 4MHz.

The standard speed for the I2C bus is 100KHz. Some devices use a higher bus speed of 400KHz. If you use an 8MHz or higher oscillator, the bus speed may exceed the devices specs, which will result in intermittent writes or reads, or in some cases, none at all. Therefore, use this **DECLARE** if you are not sure of the device's spec. The datasheet for the device used will inform you of its bus speed.

DECLARE BUS_SCL ON - **OFF**, 1 - 0 or **TRUE** - **FALSE**
Eliminates the necessity for a pullup resistor on the SCL line.

The I2C protocol dictates that a pullup resistor is required on both the SCL and SDA lines, however, this is not always possible due to circuit restrictions etc, so once the **BUS_SCL ON DECLARE** is issued at the top of the program, the resistor on the SCL line can be omitted from the circuit. The default for the compiler if the **BUS_SCL DECLARE** is not issued, is that a pullup resistor is required.

HBUSIN - HBUSOUT Declare.

DECLARE HBUS_BITRATE Constant 100, 400, 1000 etc.
The standard speed for the I2C bus is 100KHz. Some devices use a higher bus speed of 400KHz. The above **DECLARE** allows the I2C bus speed to be increased or decreased. Use this **DECLARE** with caution, as too high a bit rate may exceed the device's specs, which will result in intermittent transactions, or in some cases, no transactions at all. The datasheet for the device used will inform you of its bus speed. The default bit rate is the standard 100KHz.

HSERIN, HSEROUT, HRSIN and HRSOUT Declares.

DECLARE HSERIAL_BAUD Constant value
Sets the BAUD rate that will be used to receive a value serially. The baud rate is calculated using the **XTAL** frequency declared in the program. The default baud rate if the **DECLARE** is not included in the program listing is 2400 baud.

DECLARE HSERIAL_RCSTA Constant value (0 to 255)
HSERIAL_RCSTA, sets the respective PICmicro hardware register RCSTA, to the value in the **DECLARE**. See the Microchip data sheet for the device used for more information regarding this register.

DECLARE HSERIAL_TXSTA Constant value (0 to 255)
HSERIAL_TXSTA, sets the respective PICmicro hardware register, TXSTA, to the

value in the **DECLARE**. See the Microchip data sheet for the device used for more information regarding this register. The TXSTA register BRGH bit (bit 2) controls the high speed mode for the baud rate generator. Certain baud rates at certain oscillator speeds require this bit to be set to operate properly. To do this, set **HSERIAL_TXSTA** to a value of $24 instead of the default $20. Refer to the Microchip data sheet for the hardware serial port baud rate tables and additional information.

DECLARE HSERIAL_PARITY ODD or **EVEN**
Enables/Disables parity on the serial port. For HRSIN , HRSOUT , HSERIN and HSEROUT. The default serial data format is 8N1, 8 data bits, no parity bit and 1 stop bit. 7E1 (7 data bits, even parity, 1 stop bit) or 7O1 (7data bits, odd parity, 1 stop bit) may be enabled using the **HSERIAL_PARITY** declare.

```
DECLARE HSERIAL_PARITY = EVEN          ' Use if even parity desired
DECLARE HSERIAL_PARITY = ODD           ' Use if odd parity desired
```

DECLARE HSERIAL_CLEAR ON or **OFF**
Clear the overflow error bit before commencing a read.

Because the hardware serial port only has a 2-byte input buffer, it can easily overflow is characters are not read from it often enough. When this occurs, the USART stops accepting any new characters, and requires resetting. This overflow error can be reset by strobing the CREN bit within the RCSTA register. Example: -

```
RCSTA.4 = 0
RCSTA.4 = 1
```
or
```
CLEAR RCSTA.4
SET RCSTA.4
```

Alternatively, the **HSERIAL_CLEAR** declare can be used to automatically clear this error, even if no error occurred. However, the program will not know if an error occurred while reading, therefore some characters may be lost.

```
DECLARE HSERIAL_CLEAR = ON
```

Second USART Declares for use with HRSIN2, HSERIN2, HRSOUT2 and HSEROUT2.

DECLARE HSERIAL2_BAUD Constant value
Sets the BAUD rate that will be used to transmit a value serially. The baud rate is calculated using the **XTAL** frequency declared in the program. The default baud rate if the **DECLARE** is not included in the program listing is 2400 baud.

DECLARE HSERIAL2_RCSTA Constant value (0 to 255)
HSERIAL2_RCSTA, sets the respective PICmicro hardware register RCSTA2, to

the value in the **DECLARE**. See the Microchip data sheet for the device used for more information regarding this register. Refer to the upgrade manual pages for a description of the RCSTA2 register.

DECLARE HSERIAL2_TXSTA Constant value (0 to 255)
HSERIAL2_TXSTA, sets the respective PICmicro hardware register, TXSTA2, to the value in the **DECLARE**. See the Microchip data sheet for the device used for more information regarding this register. The TXSTA register BRGH2 bit (bit 2) controls the high speed mode for the baud rate generator. Certain baud rates at certain oscillator speeds require this bit to be set to operate properly. To do this, set **HSERIAL2_TXSTA** to a value of $24 instead of the default $20. Refer to the Microchip data sheet for the hardware serial port baud rate tables and additional information. Refer to the upgrade manual pages for a description of the TXSTA2 register.

DECLARE HSERIAL2_PARITY ODD or **EVEN**
Enables/Disables parity on the serial port. For HRSOUT2, HRSIN2 , HSEROUT2 and HSERIN2. The default serial data format is 8N1, 8 data bits, no parity bit and 1 stop bit. 7E1 (7 data bits, even parity, 1 stop bit) or 7O1 (7data bits, odd parity, 1 stop bit) may be enabled using the **HSERIAL2_PARITY** declare.

```
DECLARE HSERIAL2_PARITY = EVEN     ' Use if even parity desired
DECLARE HSERIAL2_PARITY = ODD            ' Use if odd parity desired
```

DECLARE HSERIAL2_CLEAR ON or **OFF**
Clear the overflow error bit before commencing a read.

Because the hardware serial port only has a 2-byte input buffer, it can easily overflow is characters are not read from it often enough. When this occurs, the USART stops accepting any new characters, and requires resetting. This overflow error can be reset by strobing the CREN bit within the RCSTA2 register. Example: -

```
RCSTA2.4 = 0
RCSTA2.4 = 1
```
or
```
CLEAR RCSTA2.4
SET RCSTA2.4
```

Alternatively, the **HSERIAL2_CLEAR** declare can be used to automatically clear this error, even if no error occurred. However, the program will not know if an error occurred while reading, therefore some characters may be lost.

```
DECLARE HSERIAL2_CLEAR = ON
```

HPWM Declares.

Some devices, such as the PIC16F62x, and PIC18F4xx, have alternate pins that may be used for **HPWM**. The following **DECLARES** allow the use of different pins: -

DECLARE CCP1_PIN PORT **.** PIN ' Select HPWM port and bit for CCP1 module. i.e. channel 1
DECLARE CCP2_PIN PORT **.** PIN ' Select HPWM port and bit for CCP2 module. i.e. channel 2

LCD PRINT Declares.

DECLARE LCD_DTPIN PORT **.** PIN
Assigns the Port and Pins that the LCD's DT lines will attach to.

The LCD may be connected to the PICmicro using either a 4-bit bus or an 8-bit bus. If an 8-bit bus is used, all 8 bits must be on one port. If a 4-bit bus is used, it must be connected to either the bottom 4 or top 4 bits of one port. For example: -

```
DECLARE LCD_DTPIN PORTB.4    ' Used for 4-line interface.

DECLARE LCD_DTPIN PORTB.0    ' Used for 8-line interface.
```

In the above examples, PORTB is only a personal preference. The LCD's DT lines can be attached to any valid port on the PICmicro. If the DECLARE is not used in the program, then the default Port and Pin is PORTB.4, which assumes a 4-line interface.

DECLARE LCD_ENPIN PORT **.** PIN
Assigns the Port and Pin that the LCD's EN line will attach to. This also assigns the graphic LCD's EN pin, however, the default value remains the same as for the alphanumeric type, so this will require changing.

If the DECLARE is not used in the program, then the default Port and Pin is PORTB .2.

DECLARE LCD_RSPIN PORT **.** PIN
Assigns the Port and Pins that the LCD's RS line will attach to. This also assigns the graphic LCD's RS pin, however, the default value remains the same as for the alphanumeric type, so this will require changing.

If the DECLARE is not used in the program, then the default Port and Pin is PORTB .3.

DECLARE LCD_INTERFACE 4 or 8
Inform the compiler as to whether a 4-line or 8-line interface is required by the LCD.

If the DECLARE is not used in the program, then the default interface is a 4-line type.

DECLARE LCD_LINES 1 , 2 , or 4
Inform the compiler as to how many lines the LCD has.

LCD's come in a range of sizes, the most popular being the 2 line by 16 character types. However, there are 4-line types as well. Simply place the number of lines that the particular LCD has into the declare.

If the **DECLARE** is not used in the program, then the default number of lines is 2.

GRAPHIC LCD Declares.

DECLARE LCD_TYPE 1 or 0 , **GRAPHIC** or **ALPHA**
Inform the compiler as to the type of LCD that the PRINT command will output to. If GRAPHIC or 1 is chosen then any output by the PRINT command will be directed to a graphic LCD based on the Samsung S6B0108 chipset. A value of 0 or ALPHA, or if the **DECLARE** is not issued will target the standard alphanumeric LCD type

Targeting the graphic LCD will also enable commands such as PLOT, UNPLOT, LCDREAD, and LCDWRITE.

DECLARE LCD_DTPORT PORT
Assign the port that will output the 8-bit data to the graphic LCD.

If the **DECLARE** is not used, then the default port is PORTB.

DECLARE LCD_RWPIN PORT **.** PIN
Assigns the Port and Pin that the graphic LCD's RW line will attach to.

If the **DECLARE** is not used in the program, then the default Port and Pin is PORTC.0.

DECLARE LCD_CS1PIN PORT **.** PIN
Assigns the Port and Pin that the graphic LCD's CS1 line will attach to.

If the **DECLARE** is not used in the program, then the default Port and Pin is PORTC.0.

DECLARE LCD_CS2PIN PORT **.** PIN
Assigns the Port and Pin that the graphic LCD's CS2 line will attach to.

If the DECLARE is not used in the program, then the default Port and Pin is PORTC.0.

DECLARE INTERNAL_FONT ON - **OFF**, 1 or 0
The graphic LCD's that are compatible with PROTON+ are non-intelligent types, therefore, a separate character set is required. This may be in one of two places, either externally, in an I2C eeprom, or internally in a CDATA table.

If an external font is chosen, the I2C eeprom must be connected to the specified SDA and SCL pins (as dictated by **DECLARE SDA** and **DECLARE SCL**).

If an internal font is chosen, it must be on a PICmicro device that has self modifying code features, such as the 16F87X, or 18XXXX range.

The **CDATA** table that contains the font must have a label, named FONT: preceding it. For example: -

```
FONT: CDATA $7E , $11 , $11 , $11 , $7E , $0        ' Chr "A"
    CDATA $7F , $49 , $49 , $49 , $36 , $0          ' Chr "B"
    { rest of font table }
```

The font table may be anywhere in memory, however, it is best placed after the main program code.

If the **DECLARE** is omitted from the program, then an external font is the default setting.

DECLARE FONT_ADDR 0 to 7
Set the slave address for the I2C eeprom that contains the font.

When an external source for the font is chosen, it may be on any one of 8 eeproms attached to the I2C bus. So as not to interfere with any other eeproms attached, the slave address of the eeprom carrying the font code may be chosen.

If the **DECLARE** is omitted from the program, then address 0 is the default slave address of the font eeprom.

DECLARE GLCD_CS_INVERT ON - **OFF**, 1 or 0
Some graphic LCD types have inverters on their CS lines. Which means that the LCD displays left hand data on the right side, and vice-versa. The **GLCD_CS_INVERT DECLARE**, adjusts the library LCD handling library subroutines to take this into account.

DECLARE GLCD_STROBE_DELAY 0 to 65535 us (microseconds).
Create a delay of n microseconds between strobing the EN line of the graphic LCD. This can help noisy, or badly decoupled circuits overcome random bits appearing on the LCD. The default if the **DECLARE** is not used in the BASIC program is a delay of 0.

KEYPAD Declare.

DECLARE KEYPAD_PORT PORT
Assigns the Port that the keypad is attached to.

The keypad routine requires pull-up resistors, therefore, the best Port for this device is PORTB which comes equipped with internal pull-ups. If the **DECLARE** is not used in the program, then PORTB is the default Port.

RSIN - RSOUT Declares.

DECLARE RSOUT_PIN PORT **.** PIN
Assigns the Port and Pin that will be used to output serial data from the RSOUT command. This may be any valid port on the PICmicro.

If the **DECLARE** is not used in the program, then the default Port and Pin is PORTB .0.

DECLARE RSIN_PIN PORT **.** PIN
Assigns the Port and Pin that will be used to input serial data by the RSIN command. This may be any valid port on the PICmicro.

If the **DECLARE** is not used in the program, then the default Port and Pin is PORTB .1.

DECLARE RSOUT_MODE INVERTED , **TRUE** or 1 , 0
Sets the serial mode for the data transmitted by RSOUT. This may be inverted or true. Alternatively, a value of 1 may be substituted to represent inverted, and 0 for true.

If the **DECLARE** is not used in the program, then the default mode is INVERTED.

DECLARE RSIN_MODE INVERTED , **TRUE** or 1 , 0
Sets the serial mode for the data received by RSIN. This may be inverted or true. Alternatively, a value of 1 may be substituted to represent inverted, and 0 for true.

If the **DECLARE** is not used in the program, then the default mode is INVERTED.

DECLARE SERIAL_BAUD 0 to 65535 bps (baud)
Informs the RSIN and RSOUT routines as to what baud rate to receive and transmit data.

Virtually any baud rate may be transmitted and received, but there are standard bauds, namely: -

300, 600, 1200, 2400, 4800, 9600, and 19200.

When using a 4MHz crystal, the highest baud rate that is reliably achievable is 9600. However, an increase in the oscillator speed allows higher baud rates to be achieved, including 38400 baud.

If the DECLARE is not used in the program, then the default baud is 9600.

DECLARE RSOUT_PACE 0 to 65535 microseconds (us)
Implements a delay between characters transmitted by the RSOUT command.

On occasion, the characters transmitted serially are in a stream that is too fast for

the receiver to catch, this results in missed characters. To alleviate this, a delay may be implemented between each individual character transmitted by RSOUT.

If the **DECLARE** is not used in the program, then the default is no delay between characters.

DECLARE RSIN_TIMEOUT 0 to 65535 milliseconds (ms)
Sets the time, in ms, that **RSIN** will wait for a start bit to occur.

RSIN waits in a tight loop for the presence of a start bit. If no timeout parameter is issued, then it will wait forever.

The RSIN command has the **option** of jumping out of the loop if no start bit is detected within the time allocated by timeout.

If the **DECLARE** is not used in the program, then the default timeout value is 10000ms which is 10 seconds.

SERIN - SEROUT Declares.

If communications are with existing software or hardware, its speed and mode will determine the choice of baud rate and mode. In general, 7-bit/even-parity (7E) mode is used for text, and 8-bit/no-parity (8N) for byte-oriented data. Note: the most common mode is 8-bit/no-parity, even when the data transmitted is just text. Most devices that use a 7-bit data mode do so in order to take advantage of the parity feature. Parity can detect some communication errors, but to use it you lose one data bit. This means that incoming data bytes transferred in 7E (even-parity) mode can only represent values from 0 to 127, rather than the 0 to 255 of 8N (no-parity) mode.

The compiler's serial commands SERIN and SEROUT, have the option of still using a parity bit with 4 to 8 data bits. This is through the use of a **DECLARE**: -

With parity disabled (the default setting): -

```
DECLARE SERIAL_DATA 4 ' Set SERIN and SEROUT data bits to 4
DECLARE SERIAL_DATA 5 ' Set SERIN and SEROUT data bits to 5
DECLARE SERIAL_DATA 6 ' Set SERIN and SEROUT data bits to 6
DECLARE SERIAL_DATA 7 ' Set SERIN and SEROUT data bits to 7
DECLARE SERIAL_DATA 8 ' Set SERIN and SEROUT data bits to 8 (default)
```

With parity enabled: -

```
DECLARE SERIAL_DATA 5 ' Set SERIN and SEROUT data bits to 4
DECLARE SERIAL_DATA 6 ' Set SERIN and SEROUT data bits to 5

DECLARE SERIAL_DATA 7 ' Set SERIN and SEROUT data bits to 6
DECLARE SERIAL_DATA 8 ' Set SERIN and SEROUT data bits to 7 (default)
```

```
DECLARE SERIAL_DATA 9 ' Set SERIN and SEROUT data bits to 8
```

SERIAL_DATA data bits may range from 4 bits to 8 (the default if no **DECLARE** is issued). Enabling parity uses one of the number of bits specified.

Declaring **SERIAL_DATA** as 9 allows 8 bits to be read and written along with a 9th parity bit.

Parity is a simple error-checking feature. When a serial sender is set for even parity it counts the number of 1s in an outgoing byte and uses the parity bit to make that number even. For example, if it is sending the 7-bit value: %0011010, it sets the parity bit to 1 in order to make an even number of 1s (four).

The receiver also counts the data bits to calculate what the parity bit should be. If it matches the parity bit received, the serial receiver assumes that the data was received correctly. Of course, this is not necessarily true, since two incorrectly received bits could make parity seem correct when the data was wrong, or the parity bit itself could be bad when the rest of the data was correct.

Many systems that work exclusively with text use 7-bit/ even-parity mode. For example, to receive one data byte through bit-0 of PORTA at 9600 baud, 7E, inverted.

DECLARE SERIAL_PARITY ODD or **EVEN**
Sets the parity type for SERIN, and SEROUT. When parity is enabled, this declare chooses whether it is ODD or EVEN. If parity is enabled but the **SERIAL_PARITY DECLARE** is not issued in the program, the default setting is EVEN parity.

```
DECLARE SERIAL_PARITY = EVEN          ' Use if even parity desired
DECLARE SERIAL_PARITY = ODD           ' Use if odd parity desired
```

SHIN - SHOUT Declare.

DECLARE SHIFT_DELAYUS 0 - 65535 microseconds (us)
Extend the active state of the shift clock.

The clock used by SHIN and SHOUT runs at approximately 45KHz dependent on the oscillator. The active state is held for a minimum of 2 microseconds. By placing this declare in the program, the active state of the clock is extended by an additional number of microseconds up to 65535 (65.535 milliseconds) to slow down the clock rate.

If the **DECLARE** is not used in the program, then the default is no clock delay.

CRYSTAL Frequency Declare.

DECLARE XTAL 4 , 8 , 10 , 12 , 16 , 20, or 24, For 12, and 14-bit core devices.
DECLARE XTAL 4 , 8 , 10 , 12 , 16 , 20 , 24 , 25 ,32 , 33 , or 40

Inform the compiler as to what frequency crystal is being used.

Some commands are very dependant on the oscillator frequency, RSIN, RSOUT, DELAYMS, and DELAYUS being just a few. In order for the compiler to adjust the correct timing for these commands, it must know what frequency crystal is being used.

If the **DECLARE** is not used in the program, then the default frequency is 4MHz.

Notes
The **DECLARE** directive alters the corresponding library subroutine at runtime. This means that once the **DECLARE** is added to the BASIC program, it cannot be UNDECLARED later, or changed in any way.

The **DECLARE** directive is also capable of passing information to an assembly routine. For example: -

DECLARE USE_THIS_PIN PORTA , 1

Notice the use of a comma, instead of a point for separating the register and bit number. This is because it is being passed directly to the assembler as a #DEFINE directive.