PICkit Serial DLL (PICkitS.dll) Function Prototypes

Document #:	
Title:	PICkit Serial DLL (PICkitS.dll) Function Prototypes
Subtitle:	
Original Date:	July 12, 2007
Description:	This document describes the functions available in PICkitS.dll. These functions can be called from any .NET application. While it may be possible to use PICkitS.dll in a non .NET environment, non .NET applications are not supported at this time.

Software License Agreement

Table of Contents

Introduction

History

3-11-09 Added class mTouchLV for use with LabVIEW interface
Fixed Error in USART class; Set_Baud_Rate was only changing the status block, not control block – now changes control block.
This release corresponds to PICkitS.dll version 2.4.0.0 and PICkit Serial Analyzer firmware version 0304.

01-30-09 Fixed description of I2CM.Tell_PKSA_To_Use_External_Voltage_Source
Added descriptions for I2CM Read and Receive wait functions
Added mTouchCap command ReadDevToolData
Added check in USART.Send_Data to verify p_byte_count < 251
Added check in SPIM.Send_Data to verify p_byte_count < 247
Added check in SPIM.Send_And_Receive_Data to verify p_byte_count < 246
Added new class Firmware which supplies a hook in the PKSA to update its firmware
Added Device functions `Get_PKSA_FW_Version and Get_PickitS_DLL_Version`
This release corresponds to PICkitS.dll version 2.2.0.0 and PICkit Serial Analyzer firmware version 0304.

08-18-08 Added SPI function `Send_And_Receive_Data` and overloaded I2CM functions Read and Write to handle two byte addressing
Added LIN slave function `Add_LIN_Slave_Profile_To_PKS`
This release corresponds to PICkitS.dll version 2.1.0.0 and PICkit Serial Analyzer firmware version 0303.

07-02-08 Added mTouch2 class.
Added LIN slave function prototypes.
This release corresponds to PICkitS.dll version 2.0.0.0 and PICkit Serial Analyzer firmware version 0303.

12-05-07 Updated to include Device functions: `Get_Buffer_Flush_Parameters, Set_Buffer_Flush_Parameters,`
`Get_Buffer_Flush_Time, Set_Buffer_Flush_Time,`
I2CM function `Write_Using_PEC,`
and new class mTouchCap.
This release corresponds to PICkitS.dll version 1.9.0.0 and PICkit Serial Analyzer firmware version 0202

05-03-07 Original release – for PICkitS.dll version 1.5.0.0 and PICkit Serial Analyzer firmware version 0201.

This document describes the functions available in PICkitS.dll. These functions can be called from any .NET application. While it may be possible to use PICkitS.dll in a non .NET environment, non .NET applications are not supported at this time.

I2C Master Functions

The following functions are all found in class PICkitS.I2CM.

```
        public static bool Configure_PICkitSerial_For_I2CMaster()
            ////////////////////////////////////////////////////////////////////////////
            //
            // Returns:      True if successful, False if not
            // Inputs:       None
            // Description:  Configures PICkit Serial control block for I2C_M
            //               (I2C Master) communication and tells class library
            //               to interpret incoming data as same.
            //
            ////////////////////////////////////////////////////////////////////////////

        public static bool Configure_PICkitSerial_For_I2CMaster(bool p_aux1_def,
                                                                bool p_aux2_def,
                                                                bool p_aux1_dir,
                                                                bool p_aux2_dir,
                                                                bool p_enable_pu,
                                                                double p_voltage)
            ////////////////////////////////////////////////////////////////////////////
            //
            // Returns:      True if successful, False if not
            // Inputs:       p_aux1_def  - default state for aux1 (true = on)
            //               p_aux2_def  - default state for aux2 (true = on)
            //               p_aux1_dir  - direction for aux1 (true = input)
            //               p_aux2_dir  - direction for aux2 (true = input)
            //               p_enable_pu - enable pullups (true or false)
            //               p_voltage   - target source voltage, must be in range
            //                             0.0 <= p_voltage <= 5.0
            // Description:  Configures PICkit Serial control block for I2C Master
            //               communication and tells class library to interpret
            //               incoming data as same.
            //
            ////////////////////////////////////////////////////////////////////////////
```

```
public static bool Write(byte p_slave_addr,
                         byte p_start_data_addr,
                         byte p_num_bytes_to_write,
                         ref byte[] p_data_array,
                         ref string p_script_view)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       byte p_slave_addr - I2C slave address of UUT
    //               byte p_start_data_addr - I2C command code or address
    //                                        of memory to begin writing to
    //               byte p_num_bytes_to_write - number of bytes to be written
    //               byte[] p_data_array - reference to byte array that holds
    //                                     data to be written
    //               string p_script_view - reference to a string to which
    //                                      will be copied a formatted
    //                                      view of the command
    // Description:  Attempts to perform I2C write command using above
    //               parameters.  If successful, p_num_bytes_to_write bytes are
    //               written to the I2C device beginning at p_start_data_addr.
    //               Regardless of  success or failure, the PICkit status
    //               packet is updated after the write attempt and stored
    //               for retrieval by the function There_Is_A_Status_Error.
    //
    // NOTE:         This is the first overload of this function.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Write(byte p_slave_addr,
                         byte p_command1,
                         byte p_command2,
                         byte p_num_bytes_to_write,
                         ref byte[] p_data_array,
                         ref string p_script_view)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       byte p_slave_addr    - I2C slave address of UUT
    //               byte p_command1      -
    //               byte p_command2      - p_command1 and p_command2 are
    //                                      command bytes that follow the slave
    //                                      address
    //               byte p_num_bytes_to_write - number of bytes to be written
    //               byte[] p_data_array - reference to byte array that holds
    //                                      data to be written
    //               string p_script_view - reference to a string to which
    //                                       will be copied a formatted
    //                                       view of the command
    // Description:  Attempts to perform I2C write command using above
    //               parameters.  If successful, p_num_bytes_to_write bytes are
    //               written to the I2C device using the two command bytes.
    //               In the case of word addressing for EEPROM, for example,
    //               the two command bytes can be interpreted as the starting
    //               word address for a memory location.  Regardless of
    //               success or failure, the PICkit status packet is updated
    //               after the write attempt and stored for retrieval by
    //               the function There_Is_A_Status_Error.
    //
    // NOTE:         This is the second overload of this function.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Read(byte p_slave_addr,
                        byte p_start_data_addr,
                        byte p_num_bytes_to_read,
                        ref byte[] p_data_array,
                        ref string p_script_view)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       byte p_slave_addr - I2C slave address of UUT
    //               byte p_start_data_addr - I2C command code or address
    //                                        of memory to begin reading
    //               byte p_num_bytes_to_read - number of bytes to be retured
    //               byte[] p_data_array - reference to byte array that will
    //                                     store retrieved data - must be at
    //                                     least as large as
    //                                     p_num_bytes_to_read
    //               string p_script_view - reference to a string to which
    //                                      will be copied a formatted
    //                                      view of the command
    // Description:  Attempts to perform I2C read command using above
    //               parameters.  If successful, p_num_bytes_to_read is
    //               copied into p_data_array.  It is the users responsibility
    //               to ensure p_data_array has enough room.  Regardless of
    //               success or failure, the PICkit status packet is updated
    //               after the read attempt and stored for retrieval by the
    //               function There_Is_A_Status_Error.
    //
    //               Note: this function issues the I2C Read Bytes Nack Last
    //               Byte command, not the I2C Read Bytes command.
    //
    //               Note2: this command is actually a combination read/write
    //               since we also include the word address (p_start_data_addr)
    //
    //               Note3: this is the first overload of this function
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Read(byte p_slave_addr,
                        byte p_command1,
                        byte p_command2,
                        byte p_num_bytes_to_read,
                        ref byte[] p_data_array,
                        ref string p_script_view)
    {
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       byte p_slave_addr - I2C slave address of UUT
    //               byte p_command1     -
    //               byte p_command2     - p_command1 and p_command2 are
    //                                     command bytes that follow the slave
    //                                     address
    //               byte p_num_bytes_to_read - number of bytes to be retured
    //               byte[] p_data_array - reference to byte array that will
    //                                     store retrieved data - must be at
    //                                     least as large as
    //                                     p_num_bytes_to_read
    //               string p_script_view - reference to a string to which
    //                                      will be copied a formatted
    //                                      view of the command
    // Description:  Attempts to perform I2C read command using above
    //               parameters.  If successful, p_num_bytes_to_read is
    //               copied into p_data_array.  It is the users responsibility
    //               to ensure p_data_array has enough room.  Regardless of
    //               success or failure, the PICkit status packet is updated
    //               after the read attempt and stored for retrieval by the
    //               function There_Is_A_Status_Error.  In the case of word
    //               addressing for EEPROM, for example, the two command bytes
    //               can be interpreted as the starting word address for a
    //               memory location.
    //
    //               Note: this function issues the I2C Read Bytes Nack Last
    //               Byte command, not the I2C Read Bytes command.
    //
    //               Note2: this command is actually a combination read/write
//                  since we also include the two command bytes
//
//                  Note3: this is the second overload of this function
//
////////////////////////////////////////////////////////////////////////////
```

```
public static bool Receive(byte p_slave_addr,
                           byte p_num_bytes_to_read,
                           ref byte[] p_data_array,
                           ref string p_script_view)
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       byte p_slave_addr - I2C slave address of UUT
    //               byte p_num_bytes_to_read - number of bytes to be returned
    //               byte[] p_data_array - reference to byte array that will
    //                                      store retrieved data - must be at
    //                                      least as large as
    //                                      p_num_bytes_to_read
    //               string p_script_view - reference to a string to which
    //                                       will be copied a formatted
    //                                       view of the command
    // Description:  Attempts to perform I2C simple read command using above
    //               parameters.  If successful, p_num_bytes_to_read is
    //               copied into p_data_array.  It is the users responsibility
    //               to ensure p_data_array has enough room.  Regardless of
    //               success or failure, the PICkit status packet is updated
    //               after the read attempt and stored for retrieval by the
    //               function There_Is_A_Status_Error.
    //
    //               Note: this function issues the I2C Read Bytes Nack Last
    //               Byte command, not the I2C Read Bytes command.
    //
    //               Note2: this command differs from the Read command in that
    //               it does not include the first portion of the Read command
    //               that specifies the word address (p_start_data_addr)
    //               and relies on the slave to know the start_data_addr
    //
    ////////////////////////////////////////////////////////////////////////////
```

```
public static bool Set_I2C_Bit_Rate(double p_Bit_Rate)
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      true if baud rate was successfully changed, false if not
    // Inputs:       double p_Bit_Rate - desired actual bit rate in kHz - must be
    //                                    in range of 39.1 to 5000.0
    // Description:  Reconfigures PKSA control block for new baud rate -
    //               specifically, changes byte 23 to coded value using
    //               l_calc_baud = (int)System.Math.Round((double)(CONST.FOSC *
    //                               1000) / (double)(Baud) / 4.0) - 1;
    //               does not compare resultant bit rate against p_Bit_Rate
    //               do to quantization of bit rate - it is up to user to
    //               confirm resultant bit rate is close enough to p_Bit_Rate
    //
    ////////////////////////////////////////////////////////////////////////////


public static double Get_I2C_Bit_Rate()
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      Bit rate of PKSA in kHz, returns zero if an error occured
    // Inputs:       None
    // Description:  Read Status Packet from PKSA and calculates baud rate
    //               from byte 19 of the status block
    //
    ////////////////////////////////////////////////////////////////////////////
```

```
public static void Set_Read_Wait_Time(int p_time)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      void
    // Inputs:       p_time  - time in ms the Read command should wait for all
    //                         of the user requested data to return
    // Description:  Sets time Read command to wait for all requested data to
    //               arrive in PKSA buffer.  If all requested data has not
    //               arrived in this time, the Read command will fail.
    //
    ///////////////////////////////////////////////////////////////////////////


public static int Get_Read_Wait_Time()
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      Read wait time in ms.
    // Inputs:       none
    // Description:  Returns time the Read command will wait for all requested
    //               data to return.  If all requested data has not arrived
    //               in this time, the Read command will fail.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static void Set_Receive_Wait_Time(int p_time)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:        void
    // Inputs:         p_time   - time in ms the Receive command should wait for all
    //                            of the user requested data to return
    // Description:    Sets time Receive command to wait for all requested data to
    //                 arrive in PKSA buffer.  If all requested data has not
    //                 arrived in this time, the Receive command will fail.
    //
    ///////////////////////////////////////////////////////////////////////////


public static int Get_Receive_Wait_Time()
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:        Receive wait time in ms.
    // Inputs:         none
    // Description:    Returns time the Receive command will wait for all requested
    //                 data to return.  If all requested data has not arrived
    //                 in this time, the Receive command will fail.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Get_Source_Voltage(ref double p_voltage, ref bool p_PKSA_power)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      true if voltage was successfully read, false if not
    // Inputs:       p_voltage    - reference to source voltage
    //               p_PKSA_power - reference to bool that reflects whether
    //                              PKSA is powering the I2C device
    // Description:  Reads Status Packet from PKSA and sets p_voltage
    //               to the source voltage, then sets p_PKSA_power to true
    //               if PKSA is powering the I2C device, false if it is not
    //
    ///////////////////////////////////////////////////////////////////////////

public static bool Set_Source_Voltage(double p_voltage)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      true if voltage was successfully set, false if not
    // Inputs:       p_voltage - target source voltage, must be in range
    //                           0.0 <= p_voltage <= 5.0
    // Description:  Sets source voltage to p_voltage, toggles control block
    //               byte 9, bit 5 (Vsrc)to true, toggles control block
    //               byte 9, bit 6 (Variable Vsrc) to true
    //
    ///////////////////////////////////////////////////////////////////////////


public static bool Tell_PKSA_To_Use_External_Voltage_Source()
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      true if successfully configures PKSA for external voltage
    //               source, false if not
    // Inputs:       none
    // Description:  toggles control block byte 9, bit 5 (Vsrc)to false, thereby
    //               turning off voltage to attached device
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Get_Aux_Status(ref bool p_aux1_state,
                                  ref bool p_aux2_state,
                                  ref bool p_aux1_dir,
                                  ref bool p_aux2_dir)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       p_aux1_def  - reference to default state for aux1 (true = on)
    //               p_aux2_def  - reference to default state for aux2 (true = on)
    //               p_aux1_dir  - reference to direction for aux1 (true = input)
    //               p_aux2_dir  - reference to direction for aux2 (true = input)
    // Description:  Reads Status Packet from PKSA and sets bool parameters
    //               according to the control block
    //
    ///////////////////////////////////////////////////////////////////////////

public static bool Set_Aux1_Direction(bool p_dir)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       p_dir  - true to set aux1 direction to input
    //                        false to set to output
    // Description:  Fast command to set direction of Aux1 without having to
    //               alter control block
    //
    ///////////////////////////////////////////////////////////////////////////

public static bool Set_Aux2_Direction(bool p_dir)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       p_dir  - true to set aux2 direction to input
    //                        false to set to output
    // Description:  Fast command to set direction of Aux1 without having to
    //               alter control block
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Set_Aux1_State(bool p_state)
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       p_state  - true to set aux1 state to a high voltage
    //                          false to reset it to a zero voltage
    // Description:  Fast command to set state of Aux1 without having to
    //               alter control block.  This command only has an effect
    //               when the direction of the aux line is set to output
    //
    ////////////////////////////////////////////////////////////////////////////


public static bool Set_Aux2_State(bool p_state)
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       p_state  - true to set aux2 state to a high voltage
    //                          false to reset it to a zero voltage
    // Description:  Fast command to set state of Aux2 without having to
    //               alter control block.  This command only has an effect
    //               when the direction of the aux line is set to output
    //
    ////////////////////////////////////////////////////////////////////////////


public static bool Set_Pullup_State(bool p_enable)
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      true if successfully configures PKSA for pullups
    // Inputs:       bool p_enable - true to enable, false to disable
    // Description:  toggles control block byte 9, bit 4, thereby
    //               enabling or disabling pullups
    //
    ////////////////////////////////////////////////////////////////////////////
```

```
public static bool Write_Using_PEC(byte p_slave_addr, byte p_start_data_addr,
                                   byte p_num_bytes_to_write, ref byte[] p_data_array,
                                   ref byte p_PEC, ref string p_script_view)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       byte p_slave_addr - I2C slave address of UUT
    //               byte p_start_data_addr - I2C command code or address
    //                                        of memory to begin writing to
    //               byte p_num_bytes_to_write - number of bytes to be written
    //                                           not including PEC
    //               byte[] p_data_array - reference to byte array that holds
    //                                     data to be written not including PEC
    //               string p_script_view - reference to a string to which
    //                                      will be copied a formatted
    //                                      view of the command
    //               byte p_PEC - reference to a byte that will contain
    //                            the PEC byte
    // Description:  Attempts to perform I2C write command using above
    //               parameters.  PEC is calculated using CRC-8 (SMBus
    //               protocol) on p_slave_addr, p_start_data_addr, and
    //               all the bytes in p_data_array.  The resulting PEC value
    //               is appended immediatly after the last data byte and is
    //               returned in p_PEC.  If successful, p_num_bytes_to_write + 1
    //               bytes are written to the UUT beginning at p_start_data_addr.
    //               Regardless of  success or failure, the PICkit status
    //               packet is updated after the write attempt and stored
    //               for retrieval by the function There_Is_A_Status_Error.
    //
    ////////////////////////////////////////////////////////////////////////////
```

I2C Slave Functions

The following functions are all found in class PICkitS.I2CS.

```
    public delegate void GUINotifierReceive(byte slave_addr);
        ///////////////////////////////////////////////////////////////////////////
        //
        // Triggered:      When the DLL receives a Receive command byte sequence.
        //
        // Parameters:     byte   slave_addr - Address received by DLL.  The
        //                                     application which handles the
        //                                     Receive trigger must know how
        //                                     to respond to the slave_addr.
        //
        // Description:  A delegate that is triggered whenever the dll interprets
        //               incoming data as a Receive request.  The user should register
        //               GUINotifierReceive during initialization and associate the
        //               trigger with their own routine to send appropriate data
        //               using the Send_Bytes command.  See the I2C Slave Example
        //               GUI.doc for an example.
        //
        ///////////////////////////////////////////////////////////////////////////
```

```
public delegate void GUINotifierRead(byte slave_addr, ushort byte_count, ref byte[] data);
    ///////////////////////////////////////////////////////////////////////////
    //
    // Triggered:      When the DLL receives a Read command byte sequence.
    //
    // Parameters:     byte   slave_addr - Address received by DLL.  The
    //                                     application which handles the
    //                                     Receive trigger must know how
    //                                     to respond to the slave_addr.
    //                 ushort byte_count - Number of bytes in between the
    //                                     slave Read address and the slave
    //                                     Write address.
    //                 byte[] data       - Data in-between the slave Read
    //                                     address and the slave Write address.
    //
    // Description:  A delegate that is triggered whenever the dll interprets
    //               incoming data as a Read request.  The user should register
    //               GUINotifierRead during initialization and associate the
    //               trigger with their own routine to send appropriate data
    //               using the Send_Bytes command.  See the I2C Slave Example
    //               GUI.doc for an example.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public delegate void GUINotifierWrite(byte slave_addr, ushort byte_count, ref byte[] data);
    ///////////////////////////////////////////////////////////////////////////
    //
    // Triggered:      When the DLL receives a Write command byte sequence.
    //
    // Parameters:     byte   slave_addr - Address received by DLL.  The
    //                                     application which handles the
    //                                     Receive trigger must know how
    //                                     to respond to the slave_addr.
    //                 ushort byte_count - Number of bytes following the
    //                                     slave Write address.
    //                 byte[] data       - Data following the slave Write
    //                                     address.
    //
    // Description:  A delegate that is triggered whenever the dll interprets
    //               incoming data as a Write request.  The user should register
    //               GUINotifierWrite during initialization and associate the
    //               trigger with their own routine to handle the incoming
    //               data.  See the I2C Slave Example GUI.doc for an example.
    //
    ///////////////////////////////////////////////////////////////////////////


public delegate void GUINotifierError();
    ///////////////////////////////////////////////////////////////////////////
    //
    // Triggered:      When an I2C slave status error event is received.
    //
    // Parameters:     None
    //
    // Description:    When a status error is received, the errror event is
    //                 triggered and the PKSA is reset.  Any data stored in
    //                 the PKSA buffers is lost.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Configure_PICkitSerial_For_I2CSlave_Default_Mode(byte p_slave_addr,
                                                                    byte p_slave_mask,
                                                                    byte p_read_byte_0_data,
                                                                    byte p_read_bytes_1_N_data)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       p_slave_addr           - address to which PKSA should respond
    //               p_slave_mask           - configuration mask for slave address
    //               p_read_byte_0_data     - first byte that is returned in
    //                                        response to a read or receive request
    //               p_read_bytes_1_N_data  - remaining bytes that are returned in
    //                                        response to a read or receive request
    // Description:  Configures PICkit Serial control block for I2C Slave
    //               communication with mode set to 0 (Default) and tells class
    //               library to interpret incoming data as same.
    //
    ///////////////////////////////////////////////////////////////////////////


public static bool Configure_PICkitSerial_For_I2CSlave_Interactive_Mode(byte p_slave_addr,
                                                                        byte p_slave_mask)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       p_slave_addr       - address to which PKSA should respond
    //               p_slave_mask       - configuration mask for slave address
    // Description:  Configures PICkit Serial control block for I2C Slave
    //               communication with mode set to 1 (Interactive) and tells
    //               class library to interpret incoming data as same.  Mode 0
    //               read data bytes are retained.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Configure_PICkitSerial_For_I2CSlave_Auto_Mode(byte p_slave_addr,
                                                                 byte p_slave_mask,
                                                                 byte p_array_byte_count,
                                                                 ref byte[] p_profile_array,
                                                                 ref string p_result_str)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       p_slave_addr       - address to which PKSA should respond
    //               p_slave_mask       - configuration mask for slave address
    //               p_array_byte_count - number of bytes in p_profile_array
    //               p_profile_array    - reference to array of bytes that
    //                                    contain the slave profile
    //               p_result_str       - reference to string that will contain
    //                                    error description if result is false
    // Description:  Configures PICkit Serial control block for I2C Slave
    //               communication with mode set to 2 (Auto) and tells
    //               class library to interpret incoming data as same.
    //
    ///////////////////////////////////////////////////////////////////////////


public static bool Get_I2CSlave_Address_and_Mask(ref byte p_slave_addr,
                                                 ref byte p_slave_mask)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       p_slave_addr       - reference to PKSA slave address
    //               p_slave_mask       - reference to slave configuration mask
    // Description:  Retrieves PKSA I2C Slave address and slave address mask
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Set_I2CSlave_Address_and_Mask(byte p_slave_addr,
                                                 byte p_slave_mask)
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       p_slave_addr       - address to which PKSA should respond
    //               p_slave_mask       - configuration mask for slave address
    // Description:  Sets I2C slave address and slave address mask.  Does not
    //               change I2C slave mode.
    //
    ////////////////////////////////////////////////////////////////////////////


public static bool Send_Bytes(byte p_num_bytes_to_write,
                              ref byte[] p_data_array,
                              ref string p_script_view)
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       byte p_num_bytes_to_write - number of bytes to be written
    //               byte[] p_data_array - reference to byte array that holds
    //                                     data to be written
    //               string p_script_view - reference to a string to which
    //                                      will be copied a formatted
    //                                      view of the command
    // Description:  Attempts to perform I2C SendBytes command using above
    //               parameters.  If successful, p_num_bytes_to_write bytes are
    //               returned to the UUT.  Regardless of  success or failure,
    //               the PICkit status packet is updated after the write
    //               attempt and stored for retrieval by the function
    //               There_Is_A_Status_Error.
    //
    ////////////////////////////////////////////////////////////////////////////
```

SPI Master Functions

The following functions are all found in class PICkitS.SPIM.

```
        public static bool Configure_PICkitSerial_For_SPIMaster()
            ///////////////////////////////////////////////////////////////////////////
            //
            // Returns:      True if successful, False if not
            // Inputs:       None
            // Description:  Configures PICkit Serial control block for SPI
            //               Master communication and tells class library
            //               to interpret incoming data as same.  Calling this function
            //               is the equivalent of calling the overloaded function below
            //               with the following defaults:
            //               p_sample_phase          default = True
            //               p_clock_edge_select     default = True
            //               p_clock_polarity        default = False
            //               p_auto_output_disable   default = False
            //               p_chip_sel_polarity     default = False
            //               p_supply_5V             default = True
            //
            ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Configure_PICkitSerial_For_SPIMaster(bool p_sample_phase,
                                                       bool p_clock_edge_select,
                                                       bool p_clock_polarity,
                                                       bool p_auto_output_disable,
                                                       bool p_chip_sel_polarity,
                                                       bool p_supply_5V)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:  True if successful, False if not
    // Inputs:   p_sample_phase         - sample phase, true means
    //                                    input data sampled at end of data
    //                                    output time
    //           p_clock_edge_select    - clock edge select,
    //                                    true = transmit occurs on transition
    //                                    from idle to active clock state
    //           p_clock_polarity       - clock polarity, true =
    //                                    idle state for clock is high
    //           p_auto_output_disable  - auto output disable
    //                                    true = disable output during input,
    //                                    allows SDI and SDO to be shorted
    //                                    for 3-wire communication
    //           p_chip_sel_polarity    - chip select polarity,
    //                                    true = Hi-true, false = Lo-true
    //           p_supply_5V            - turn on 5V source voltage, false
    //                                    means the device is powered externally
    // Description:  Configures PICkit Serial control block for SPI
    //               Master communication and tells class library
    //               to interpret incoming data as same.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Send_Data(byte p_byte_count,
                             ref byte[] p_data_array,
                             bool p_assert_cs,
                             bool p_de_assert_cs,
                             ref string p_script_view)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    //
    // Inputs:       p_byte_count   - number of bytes in p_data_array to send,
    //                                must be less than 247
    //               p_data_array   - reference to byte array that holds
    //                                data to be written
    //               p_assert_cs    - boolean that says whether or not to
    //                                prepend the command with a chip select
    //                                assert
    //               p_de_assert_cs - boolean that says whether or not to
    //                                append the command with a chip select
    //                                de-assert
    //               p_script_view  - reference to a string to which will be
    //                                copied a formatted view of the command
    //
    // Description:  Attempts to perform generic SPI command using above
    //               parameters.  Regardless of success or failure, the
    //               PICkit Serial status packet is updated after the
    //               write attempt and stored for retrieval by the function
    //               There_Is_A_Status_Error.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Receive_Data(byte p_byte_count,
                                ref byte[] p_data_array,
                                bool p_assert_cs,
                                bool p_de_assert_cs,
                                ref string p_script_view)
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    //
    // Inputs:       p_byte_count   - number of bytes in p_data_array to read
    //               p_data_array   - reference to byte array that holds
    //                                data to be read
    //               p_assert_cs    - boolean that says whether or not to
    //                                prepend the command with a chip select
    //                                assert
    //               p_de_assert_cs - boolean that says whether or not to
    //                                append the command with a chip select
    //                                de-assert
    //               p_script_view  - reference to a string to which will be
    //                                copied a formatted view of the command
    //
    // Description:  Attempts to perform generic SPI data read using above
    //               parameters.  Regardless of success or failure, the
    //               PICkit Serial status packet is updated after the
    //               write attempt and stored for retrieval by the function
    //               There_Is_A_Status_Error.
    //
    ////////////////////////////////////////////////////////////////////////////
```

```
public static bool Send_And_Receive_Data(byte p_byte_count,
                                         ref byte[] p_send_data_array,
                                         ref byte[] p_receive_data_array,
                                         bool p_assert_cs,
                                         bool p_de_assert_cs,
                                         ref string p_script_view)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    //
    // Inputs:       p_byte_count          - number of bytes in data array to
    //                                       send and receive,
    //                                       must be less than 246
    //               p_send_data_array     - reference to byte array that holds
    //                                       data to be written
    //               p_receive_data_array - reference to byte array that holds
    //                                       data to be read
    //               p_assert_cs           - boolean that says whether or not to
    //                                       prepend the command with a chip select
    //                                       assert
    //               p_de_assert_cs        - boolean that says whether or not to
    //                                       append the command with a chip select
    //                                       de-assert
    //               p_script_view         - reference to a string to which will be
    //                                       copied a formatted view of the command
    //
    // Description:  Attempts to perform generic SPI data write/read using above
    //               parameters.  Regardless of success or failure, the
    //               PICkit Serial status packet is updated after the
    //               write attempt and stored for retrieval by the function
    //               There_Is_A_Status_Error.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Set_SPI_BitRate(double p_Bit_Rate)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:       true if baud rate was successfully changed, false if not
    // Inputs:        double p_Bit_Rate - desired actual bit rate in KHz
    //                                    p_Bit_Rate must be in the range:
    //                  0.61 kHz <= p_Bit_Rate <= 1250 kHz
    // Description:   Sends script to change bit rate in status block - control
    //                block is unchanged.  Bit rate is determined by the equation
    //                bite rate = FOSC / prescale / scale * 1000.0, where
    //                FOSC - 20.0 MHz, prescale = 8, 32, or 128, scale = 1-256.
    //                Since there are multiple ways a given bit rate can be
    //                derived, this function determines the prescale and scale
    //                values that best match the target bit rate.
    //
    ///////////////////////////////////////////////////////////////////////////


public static double Get_SPI_Bit_Rate()
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:       Bit rate of PKSA in kHz, returns zero if an error occured
    //                or if status block is not configured properly
    // Inputs:        None
    // Description:   Read Status Packet from PKSA and calculates bit rate
    //                from bytes 18 and 19 of the status block
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Get_SPI_Status(ref bool p_sample_phase,
                                  ref bool p_clock_edge_select,
                                  ref bool p_clock_polarity,
                                  ref bool p_auto_output_disable,
                                  ref bool p_SDI_state,
                                  ref bool p_SDO_state,
                                  ref bool p_SCK_state,
                                  ref bool p_chip_select_state)
    {
    //////////////////////////////////////////////////////////////////////////
    //
    // Returns:  True if successful, False if not
    // Inputs:   p_sample_phase         - reference to sample phase, true means
    //                                    input data sampled at end of data
    //                                    output time
    //           p_clock_edge_select    - reference to clock edge select,
    //                                    true = transmit occurs on transition
    //                                    from idle to active clock state
    //           p_clock_polarity       - reference to clock polarity, true =
    //                                    idle state for clock is high
    //           p_auto_output_disable  - reference to auto output disable
    //                                    true = disable output during input,
    //                                    allows SDI and SDO to be shorted
    //                                    for 3-wire communication
    //           p_SDI_state            - reference for state of data in,
    //                                    true = SDI Line is High
    //           p_SDO_state            - reference for state of data out,
    //                                    true = SDO Line is High
    //           p_SCK_state            - reference for state of clock,
    //                                    true = Clock Pin is High
    //           p_chip_select_state    - reference for state of ship select,
    //                                    true = Chip Select Line is High
    // Description:  Reads Status Packet from PKSA and returns bool parameters
    //               according to the status block
    //
    //////////////////////////////////////////////////////////////////////////
```

```
public static bool Tell_PKSA_To_Use_External_Voltage_Source()
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      true if successfully configures PKSA for external voltage
    //               source, false if not
    // Inputs:       none
    // Description:  toggles control block byte 9, bit 5 (Vsrc)to false, thereby
    //               turning off voltage to attached device
    //
    ////////////////////////////////////////////////////////////////////////////


public static bool Tell_PKSA_To_Power_My_Device()
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      true if successfully configures PKSA for internal (5.0)
    //               voltage source, false if not
    // Inputs:       none
    // Description:  toggles control block byte 9, bit 5 (Vsrc)to true, thereby
    //               turning on voltage to attached device
    //
    ////////////////////////////////////////////////////////////////////////////
```

Microwire Master Functions

The following functions are all found in class PICkitS.MicrowireM.

```
        public static bool Configure_PICkitSerial_For_MicrowireMaster()
            ///////////////////////////////////////////////////////////////////////////
            //
            // Returns:      True if successful, False if not
            // Inputs:       None
            // Description:  Configures PICkit Serial control block for Microwire
            //               Master communication and tells class library
            //               to interpret incoming data as same.
            //
            ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Configure_PICkitSerial_For_MicrowireMaster(bool p_sample_phase,
                                                              bool p_clock_edge_select,
                                                              bool p_clock_polarity,
                                                              bool p_auto_output_disable,
                                                              bool p_chip_sel_polarity,
                                                              bool p_supply_5V)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:  True if successful, False if not
    // Inputs:   p_sample_phase         - sample phase, true means
    //                                    input data sampled at end of data
    //                                    output time
    //           p_clock_edge_select    - clock edge select,
    //                                    true = transmit occurs on transition
    //                                    from idle to active clock state
    //           p_clock_polarity       - clock polarity, true =
    //                                    idle state for clock is high
    //           p_auto_output_disable  - auto output disable
    //                                    true = disable output during input,
    //                                    allows SDI and SDO to be shorted
    //                                    for 3-wire communication
    //           p_chip_sel_polarity    - chip select polarity,
    //                                    true = Hi-true, false = Lo-true
    //           p_supply_5V            - turn on 5V source voltage, false
    //                                    means the device is powered externally
    // Description:  Configures PICkit Serial control block for Microwire
    //               Master communication and tells class library
    //               to interpret incoming data as same.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Send_Data(byte p_byte_count,
                             ref byte[] p_data_array,
                             bool p_assert_cs,
                             bool p_de_assert_cs,
                             ref string p_script_view)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:       True if successful, False if not
    //
    // Inputs:        p_byte_count   - number of bytes in p_data_array to send
    //                p_data_array   - reference to byte array that holds
    //                                 data to be written
    //                p_assert_cs    - boolean that says whether or not to
    //                                 prepend the command with a chip select
    //                                 assert
    //                p_de_assert_cs - boolean that says whether or not to
    //                                 append the command with a chip select
    //                                 de-assert
    //                p_script_view  - reference to a string to which will be
    //                                 copied a formatted view of the command
    //
    // Description:   Attempts to perform generic Microwire command using above
    //                parameters.  Regardless of success or failure, the
    //                PICkit Serial status packet is updated after the
    //                write attempt and stored for retrieval by the function
    //                There_Is_A_Status_Error.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Receive_Data(byte p_byte_count,
                                ref byte[] p_data_array,
                                bool p_assert_cs,
                                bool p_de_assert_cs,
                                ref string p_script_view)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    //
    // Inputs:       p_byte_count   - number of bytes in p_data_array to send
    //               p_data_array   - reference to byte array that holds
    //                                data to be written
    //               p_assert_cs    - boolean that says whether or not to
    //                                prepend the command with a chip select
    //                                assert
    //               p_de_assert_cs - boolean that says whether or not to
    //                                append the command with a chip select
    //                                de-assert
    //               p_script_view  - reference to a string to which will be
    //                                copied a formatted view of the command
    //
    // Description:  Attempts to perform generic Microwire data read using above
    //               parameters.  Regardless of success or failure, the
    //               PICkit Serial status packet is updated after the
    //               write attempt and stored for retrieval by the function
    //               There_Is_A_Status_Error.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Set_Microwire_BitRate(double p_Bit_Rate)
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      true if baud rate was successfully changed, false if not
    // Inputs:       double p_Bit_Rate - desired actual bit rate in KHz
    //                                   p_Bit_Rate must be in the range:
    //                 0.61 kHz <= p_Bit_Rate <= 1250 kHz
    // Description:  Sends script to change bit rate in status block - control
    //               block is unchanged.  Bit rate is determined by the equation
    //               bite rate = FOSC / prescale / scale * 1000.0, where
    //               FOSC - 20.0 MHz, prescale = 8, 32, or 128, scale = 1-256.
    //               Since there are multiple ways a given bit rate can be
    //               derived, this function determines the prescale and scale
    //               values that best match the target bit rate.
    //
    ////////////////////////////////////////////////////////////////////////////


public static double Get_Microwire_Bit_Rate()
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      Bit rate of PKSA in kHz, returns zero if an error occured
    //               or if status block is not configured properly
    // Inputs:       None
    // Description:  Read Status Packet from PKSA and calculates bit rate
    //               from bytes 18 and 19 of the status block
    //
    ////////////////////////////////////////////////////////////////////////////
```

```
public static bool Get_Microwire_Status(ref bool p_sample_phase, ref bool p_clock_edge_select,
                                        ref bool p_clock_polarity, ref bool p_auto_output_disable,
                                        ref bool p_SDI_state, ref bool p_SDO_state,
                                        ref bool p_SCK_state, ref bool p_chip_select_state)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:  True if successful, False if not
    // Inputs:   p_sample_phase          - reference to sample phase, true means
    //                                     input data sampled at end of data
    //                                     output time
    //           p_clock_edge_select     - reference to clock edge select,
    //                                     true = transmit occurs on transition
    //                                     from idle to active clock state
    //           p_clock_polarity        - reference to clock polarity, true =
    //                                     idle state for clock is high
    //           p_auto_output_disable   - reference to auto output disable
    //                                     true = disable output during input,
    //                                     allows SDI and SDO to be shorted
    //                                     for 3-wire communication
    //           p_SDI_state             - reference for state of data in,
    //                                     true = SDI Line is High
    //           p_SDO_state             - reference for state of data out,
    //                                     true = SDO Line is High
    //           p_SCK_state             - reference for state of clock,
    //                                     true = Clock Pin is High
    //           p_chip_select_state     - reference for state of chip select,
    //                                     true = Chip Select Line is High
    // Description:  Reads Status Packet from PKSA and sets bool parameters
    //               according to the status block
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Tell_PKSA_To_Use_External_Voltage_Source()
    {
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      true if successfully configures PKSA for external voltage
    //               source, false if not
    // Inputs:       none
    // Description:  toggles control block byte 9, bit 5 (Vsrc)to false, thereby
    //               turning off voltage to attached device
    //
    ///////////////////////////////////////////////////////////////////////////


public static bool Tell_PKSA_To_Power_My_Device()
    {
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      true if successfully configures PKSA for internal (5.0)
    //               voltage source, false if not
    // Inputs:       none
    // Description:  toggles control block byte 9, bit 5 (Vsrc)to true, thereby
    //               turning on voltage to attached device
    //
    ///////////////////////////////////////////////////////////////////////////
```

USART Functions

The following functions are all found in class PICkitS.USART.

```
        public static bool Configure_PICkitSerial_For_USARTAsync()
            ///////////////////////////////////////////////////////////////////////////
            //
            // Returns:       True if successful, False if not
            // Inputs:        None
            // Description:   Configures PICkit Serial control block for USART
            //                Asynchronous communication and tells class library
            //                to interpret incoming data as same.
            //
            ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Configure_PICkitSerial_For_USARTAsync(bool p_aux1_def,
                                                         bool p_aux2_def,
                                                         bool p_aux1_dir,
                                                         bool p_aux2_dir,
                                                         bool p_rcv_dis,
                                                         double p_voltage)
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       p_aux1_def - default state for aux1 (true = on)
    //               p_aux2_def - default state for aux2 (true = on)
    //               p_aux1_dir - direction for aux1 (true = input)
    //               p_aux2_dir - direction for aux2 (true = input)
    //               p_rcv_dis  - async receive disabled (true or false)
    //               p_voltage  - target source voltage, must be in range
    //                            0.0 <= p_voltage <= 5.0
    // Description:  Configures PICkit Serial control block for USART
    //               Asynchronous communication and tells class library
    //               to interpret incoming data as same.
    //
    ////////////////////////////////////////////////////////////////////////////
```

```
public static bool Configure_PICkitSerial_For_USARTSyncMaster()
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       None
    // Description:  Configures PICkit Serial control block for USART
    //               Synchronous Master communication and tells class library
    //               to interpret incoming data as same.
    //
    ///////////////////////////////////////////////////////////////////////////


public static bool Configure_PICkitSerial_For_USARTSyncMaster(bool p_aux1_def,
                                                              bool p_aux2_def,
                                                              bool p_aux1_dir,
                                                              bool p_aux2_dir,
                                                              bool p_clock_pol,
                                                              double p_voltage)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       p_aux1_def  - default state for aux1 (true = on)
    //               p_aux2_def  - default state for aux2 (true = on)
    //               p_aux1_dir  - direction for aux1 (true = input)
    //               p_aux2_dir  - direction for aux2 (true = input)
    //               p_clock_pol - clock polarity (true = inverted)
    //               p_voltage   - target source voltage, must be in range
    //                             0.0 <= p_voltage <= 5.0
    // Description:  Configures PICkit Serial control block for USART
    //               Asynchronous communication and tells class library
    //               to interpret incoming data as same.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Configure_PICkitSerial_For_USARTSyncSlave()
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       None
    // Description:  Configures PICkit Serial control block for USART
    //               Synchronous Slave communication and tells class library
    //               to interpret incoming data as same.
    //
    ////////////////////////////////////////////////////////////////////////////


public static uint Retrieve_Data_Byte_Count()
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      Number of bytes available to be read
    //
    // Inputs:       None
    //
    // Description:  Retrieves the number of bytes available to be read from
    //               the data buffer in PICkitS.dll.  Typically this command
    //               is followed by the Retrieve_Data command where
    //               the byte count is passed as a parameter.
    //
    ////////////////////////////////////////////////////////////////////////////
```

```
public static bool Retrieve_Data(uint p_byte_count, ref byte[] p_data_array)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    //
    // Inputs:       p_byte_count - number of bytes to retrieve
    //               p_data_array - reference to an array of bytes - must be at
    //                              least as large as p_byte_count
    //
    // Description:  Retrieves p_byte_count bytes from the PICkitS.dll data buffer
    //               and copies them into p_data_array.  After copying the data,
    //               the PICkitS.dll data buffer byte count is decremented by
    //               p_byte_count.
    //
    ///////////////////////////////////////////////////////////////////////////


public static bool Send_Data(byte p_byte_count, ref byte[] p_data_array, ref string p_script_view)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    //
    // Inputs:       p_byte_count  - number of bytes in p_data_array to send
    //                               must be less than 252
    //               p_data_array  - reference to byte array that holds
    //                               data to be written
    //               p_script_view - reference to a string to which will be
    //                               copied a formatted view of the command
    //
    // Description:  Attempts to perform generic USART command using above
    //               parameters.  Regardless of  success or failure, the
    //               PICkit Serial status packet is updated after the
    //               write attempt and stored for retrieval by the function
    //               There_Is_A_Status_Error.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Set_Baud_Rate(ushort p_baud)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      true if baud rate was successfully changed, false if not
    // Inputs:       ushort Baud - desired actual baud rate - not coded value
    // Description:  Reconfigures PKSA control block for new baud rate -
    //               specifically, changes bytes 22 & 23 to coded value using
    //               l_calc_baud = (int)System.Math.Round((double)(CONST.FOSC *
    //                              1000000) / (double)(Baud) / 4.0) - 1;
    //
    ///////////////////////////////////////////////////////////////////////////


public static ushort Get_Baud_Rate()
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      Baud rate of PKSA, returns zero if an error occured
    // Inputs:       None
    // Description:  Read Status Packet from PKSA and calculates baud rate
    //               from bytes 18 and 19 of the control block
    //
    ///////////////////////////////////////////////////////////////////////////


public static bool Get_Source_Voltage(ref double p_voltage, ref bool p_PKSA_power)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      true if voltage was successfully read, false if not
    // Inputs:       p_voltage    - reference to source voltage
    //               p_PKSA_power - reference to bool that reflects whether
    //                              PKSA is powering the usart device
    // Description:  Reads Status Packet from PKSA and sets p_voltage
    //               to the source voltage, then sets p_PKSA_power to true
    //               if PKSA is powering the usart device, false if it is not
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Set_Source_Voltage(double p_voltage)
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      true if voltage was successfully set, false if not
    // Inputs:       p_voltage - target source voltage, must be in range
    //                            0.0 <= p_voltage <= 5.0
    // Description:  Sets source voltage to p_voltage, toggles control block
    //               byte 9, bit 5 (Vsrc)to true, toggles control block
    //               byte 9, bit 6 (Variable Vsrc) to true
    //
    ////////////////////////////////////////////////////////////////////////////


public static bool Tell_PKSA_To_Use_External_Voltage_Source()
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      true if successfully configures PKSA for external voltage
    //               source, false if not
    // Inputs:       none
    // Description:  toggles control block byte 9, bit 5 (Vsrc)to false, thereby
    //               turning off voltage to attached device
    //
    ////////////////////////////////////////////////////////////////////////////
```

```
public static bool Get_Aux_Status(ref bool p_aux1_state,
                                  ref bool p_aux2_state,
                                  ref bool p_aux1_dir,
                                  ref bool p_aux2_dir)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       p_aux1_def  - reference to default state for aux1 (true = on)
    //               p_aux2_def  - reference to default state for aux2 (true = on)
    //               p_aux1_dir  - reference to direction for aux1 (true = input)
    //               p_aux2_dir  - reference to direction for aux2 (true = input)
    // Description:  Reads Status Packet from PKSA and sets bool parameters
    //               according to the status block
    //
    ///////////////////////////////////////////////////////////////////////////


public static bool Set_Aux1_Direction(bool p_dir)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       p_dir  - true to set aux1 direction to input
    //                        false to set to output
    // Description:  Fast command to set direction of Aux1 without having to
    //               alter control block
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Set_Aux2_Direction(bool p_dir)
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       p_dir  - true to set aux2 direction to input
    //                        false to set to output
    // Description:  Fast command to set direction of Aux1 without having to
    //               alter control block
    //
    ////////////////////////////////////////////////////////////////////////////


public static bool Set_Aux1_State(bool p_state)
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       p_state  - true to set aux1 state to a high voltage
    //                          false to reset it to a zero voltage
    // Description:  Fast command to set state of Aux1 without having to
    //               alter control block.  This command only has an effect
    //               when the direction of the aux line is set to output
    //
    ////////////////////////////////////////////////////////////////////////////


public static bool Set_Aux2_State(bool p_state)
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       p_state  - true to set aux2 state to a high voltage
    //                          false to reset it to a zero voltage
    // Description:  Fast command to set state of Aux2 without having to
    //               alter control block.  This command only has an effect
    //               when the direction of the aux line is set to output
    //
    ////////////////////////////////////////////////////////////////////////////
```

LIN Functions

The following functions are all found in class PICkitS.LIN.

```
        public static bool Configure_PICkitSerial_For_LIN()
            ///////////////////////////////////////////////////////////////////////////
            //
            // Returns:      True if successful, False if not
            // Inputs:       None
            // Description:  Configures PICkit Serial, or attached MyDevice control
            //               block for LIN communication and tells class library
            //               to interpret incoming data as same.
            //
            ///////////////////////////////////////////////////////////////////////////
```

```
public delegate void OnReceive(byte masterid, byte[] data, byte length, byte error,
                               ushort baud, double time);
    ///////////////////////////////////////////////////////////////////////////
    //
    // Triggered:      When a frame is received, the DLL is in LISTEN mode and
    //                 the frame is different than what is in the buffer.
    //                                         OR
    //                 When a frame is received and the DLL is in DISPLAY_All mode
    //                                         AND
    //                 In Master mode and sent a whole frame.
    //
    // Paramters:      byte   masterid - Frame ID
    //                 byte[] data     - array of data associated with ID
    //                                 - only contains data, not the ID
    //                 byte   length   - bytecount of data only, not the ID
    //                 ushort baud     - value of baud rate received
    //                                   with frame data
    //                 double time     - time in seconds since last frame break
    //                 byte   error    - Error code: 0 - no error
    //                                               1 - timeout
    //                                               2 - no sync break
    //                                               3 - error resetting timestamp
    //                                               4 - PKSA status error
    //                                               5 - PKSA error event marker
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public delegate void OnAnswer(byte masterid, byte[] data, byte length, byte error,
                              ushort baud, double time);
    ////////////////////////////////////////////////////////////////////////////
    //
    // Triggered:      When data is successfully retrieved as the result of
    //                 the Transmit command of zero length data bytes.
    // Paramters:      byte   masterid - Frame ID
    //                 byte[] data     - array of data associated with ID
    //                                 - only contains data, not the ID
    //                 byte   length   - bytecount of data only, not the ID
    //                 ushort baud     - value of baud rate sent
    //                 double time     - time in seconds since last frame break
    //
    //                 byte   error    - Error code: 0 - no error
    //                                               1 - timeout
    //                                               2 - no sync break
    //                                               3 - error resetting timestamp
    //                                               4 - PKSA status error
    //                                               5 - PKSA error event marker
    //
    ////////////////////////////////////////////////////////////////////////////
```

```
public static bool Set_OnReceive_Timeout(int Timeout)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if timeout is successfully set, false if not
    // Inputs:       int - Timeout in milliseconds
    // Description:  Sets timeout for OnReceive.  If 9 bytes (8 data and
    //               checksum) are not received in this time (time starts
    //               when the frame break is received), then OnReceive will
    //               be triggered with a timeout error.
    //
    //               Setting Timeout to 0xFFFF will
    //               force OnReceive Frame timeouts to be calculated as
    //               a function of the BAUD rate per the equation:
    //
    //               Timeout = 11 bytes * 1sec/(BAUD/10)bytes * 1.5 * 1000,
    //
    //               so 9600 baud would yield a Timeout of 17.
    //
    //               Note that Windows timer resolution is 55 ms, so
    //               Timeout values in this range or lower may not
    //               yield repeatable results.
    //
    ///////////////////////////////////////////////////////////////////////////


public static int Get_OnReceive_Timeout()
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      Frame timeout in milliseconds
    // Inputs:       None
    // Description:  Returns the timeout in milliseconds of the OnReceive
    //               Frame timeout.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool OnReceive_Timeout_Is_Baud_Dependent()
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if the OnReceive timeout is being calculated from
    //               the BAUD rate, false if not.
    // Inputs:       None
    // Description:  Returns the state of a boolean flag that tells the dll
    //               whether or not to calculate the OnReceive timeout from
    //               the Baud rate.
    //
    ////////////////////////////////////////////////////////////////////////////


public static bool SetModeListen()
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       None
    // Description:  Sets internal DLL mode to LISTEN
    //
    ////////////////////////////////////////////////////////////////////////////


public static bool SetModeTransmit()
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       None
    // Description:  Sets internal DLL mode to TRANSMIT
    //
    ////////////////////////////////////////////////////////////////////////////
```

```
public static bool SetModeDisplayAll()
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       None
    // Description:  Sets internal DLL mode to DISPLAY_ALL
    //
    ///////////////////////////////////////////////////////////////////////////


public static bool Transmit_mode_Is_Set()
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if internal DLL mode is set to TRANSMIT, False if not
    // Inputs:       None
    // Description:  Verifies DLL mode is set to TRANSMIT
    //
    ///////////////////////////////////////////////////////////////////////////


public static bool Listen_mode_Is_Set()
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if internal DLL mode is set to LISTEN, False if not
    // Inputs:       None
    // Description:  Verifies DLL mode is set to LISTEN
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool DisplayAll_mode_Is_Set()
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:       True if internal DLL mode is set to DISPLAY_ALL, False if not
    // Inputs:        None
    // Description:   Verifies DLL mode is set to DISPLAY_ALL
    //
    ///////////////////////////////////////////////////////////////////////////


public static void Reset_LIN_Frame_Buffers()
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:       void
    // Inputs:        None
    // Description:   Reinitializes internal DLL Frame buffers to default values
    //
    ///////////////////////////////////////////////////////////////////////////


public static void Reset_Timer()
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:       void
    // Inputs:        None
    // Description:   sets flag in class that the next frame should have
    //                timestamp of zero.  This resets the class timer.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Transmit(byte MasterID,
                            byte[] Data,
                            byte DataByteCount,
                            ref string ErrorString)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if data is successfully sent on the bus AND if correct
    //               data (if required) is returned
    // Inputs:       byte[] Data        - array of bytes that holds
    //                                    optional data and checksum.
    //               byte DataByteCount - number of valid bytes in Data.  Includes
    //                                    data if present, and checksum.
    //                                    DataByteCount = 0 if no data is present.
    //               string ErrorString - reference to a string that will contain
    //                                    an error msg if the Transmit command fails
    //                                    or the data received is not good
    //
    // Example:   ID [D1][D2][CS][--] -    2 data bytes + checksum: DataByteCount = 3
    //            ID [--][--][--][--] -    no data, no checksum: DataByteCount = 0
    // Description:  1. Send Transmit Frame command using Data
    //               2. If DataByteCount > 1 we do not expect to see any data returned
    //                  just send the command
    //               3. If DataByteCount = 0, we expect to see data returned.  Wait
    //                  for TIMEOUT ms after sending the command and gather data.
    //                  If valid data is retrieved, trigger the OnAnswer event
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Change_LIN_BAUD_Rate(ushort Baud)
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      true if baud rate was successfully changed, false if not
    // Inputs:       ushort Baud - desired actual baud rate - not coded value
    // Description:  Reconfigures PKSA control block for new baud rate -
    //               specifically, changes CB bytes 22 & 23 to coded value using
    //               l_calc_baud = (int)System.Math.Round((double)(CONST.FOSC *
    //                              1000000) / (double)(Baud) / 4.0) - 1;
    //
    ////////////////////////////////////////////////////////////////////////////

public static ushort Get_LIN_BAUD_Rate()
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      Baud rate of PKSA, returns zero if an error occured
    // Inputs:       None
    // Description:  Read Status Packet from PKSA and calculates baud rate
    //               from bytes 18 and 19 of the control block
    //
    ////////////////////////////////////////////////////////////////////////////
```

```
public static bool Configure_PICkitSerial_For_LINSlave_Mode(byte p_array_byte_count,
                                                            ref byte[] p_profile_array,
                                                            ref string p_result_str,
                                                            bool p_autobaud,
                                                            ref int p_error_code)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       p_array_byte_count - number of bytes in p_profile_array
    //               p_profile_array    - reference to array of bytes that
    //                                    contain the slave profile
    //               p_result_str       - reference to string that will contain
    //                                    error description if result is false
    //               p_autobaud         - boolean that indicates if the autobaud
    //                                    bit should be set in control block
    //                                    true = set, false = not set
    //               p_error_code       - flag that details the reason for
    //                                    function failure:
    //                                    0 - no failure, LIN slave implemented
    //                                    1 - p_array_byte_count larger than what
    //                                        size of CBUF3
    //                                    2 - write error
    //                                    3 - verification error
    // Description:  Configures PICkit Serial control block for LIN Slave
    //               communication and tells class library to interpret
    //               incoming data as same.
    //
    // IMPORTANT -   Use this function to load a slave profile to the PKSA
    //               with DEFAULT configurations.  All configuration parameters
    //               will be reset to their default values after this
    //               function call.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Add_LIN_Slave_Profile_To_PKS(byte p_array_byte_count,
                                                ref byte[] p_profile_array,
                                                ref string p_result_str,
                                                ref int p_error_code)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       p_array_byte_count - number of bytes in p_profile_array
    //               p_profile_array    - reference to array of bytes that
    //                                    contain the slave profile
    //               p_result_str       - reference to string that will contain
    //                                    error description if result is false
    //               p_error_code       - flag that details the reason for
    //                                    function failure:
    //                                    0 - no failure, LIN slave implemented
    //                                    1 - p_array_byte_count larger than what
    //                                        size of CBUF3
    //                                    2 - write error
    //                                    3 - verification error
    // Description:  Configures PICkit Serial control block for LIN Slave
    //               communication and tells class library to interpret
    //               incoming data as same.
    //
    // IMPORTANT -   Use this function AFTER the PKSA has already been
    //               configured for LIN.  This function will not change
    //               other LIN configuration parameters - it will only
    //               add the slave profile, and erase any existing slave
    //               profile.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Write_Slave_Profile(byte p_masterid,
                                       ref byte[] p_data,
                                       byte p_byte_count,
                                       ref byte p_error_code)
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if able to write slave profile data associated with
    //               p_masterid, false if not.
    //
    //               p_masterid   - frame header id
    // Inputs:       p_data       - array of bytes that holds frame
    //                              data
    //               p_byte_count - number of bytes in p_data to write,
    //                              must be less than 245
    //               p_error_code - flag that details the reason for function
    //                              failure:
    //                              0 - no failure, data retrieved OK
    //                              1 - verification failed
    //                              2 - write error
    //                              3 - byte count too large
    //
    // Description:  Writes p_data to slave profile p_masterid.  If an error
    //               code of 1 (verification error) is returned, the user
    //               should call Read_Slave_Profile to get detailed information
    //               about the slave profile
    //
    // IMPORTANT -   This function allows you to change the data associated with
    //               an existing slave profile only.  You must first create a
    //               slave profile in the PKSA by invoking a call to
    //               Configure_PICkitSerial_For_LINSlave_Mode.
    //
    ////////////////////////////////////////////////////////////////////////////
```

```
public static bool Read_Slave_Profile(byte p_masterid, ref byte[] p_data,
                                       byte p_expected_byte_count,
                                       ref byte p_actual_byte_count,
                                       ref byte p_error_code)
            ///////////////////////////////////////////////////////////////////////////
            //
            // Returns:      True if able to read slave profile data associated with
            //               p_masterid, false if not.
            //
            //               p_masterid            - frame header id
            // Inputs:       p_data                - array of bytes that holds retrieved
            //                                       data
            //               p_expected_byte_count - number of expected bytes associated
            //                                       with this frame header
            //               p_actual_byte_count   - number of bytes actually returned
            //               p_error_code - flag that details the reason for function
            //                              failure:
            //                              0 - no failure, data retrieved OK
            //                              1 - timeout, no data retrieved within
            //                                  two seconds
            //                              2 - write error
            //                              3 - retrieved data byte count > p_expected_byte_count
            //                              4 - p_masterid does not match returned value
            //
            // Description:  Retrieves data associated with p_masterid in slave profile.
            //               If p_actual_byte_count is less than or equal to
            //               p_expected_byte_count, no error is returned.  If
            //               p_actual_byte_count > p_expected_byte_count, error code 3
            //               is returned.
            //
            ///////////////////////////////////////////////////////////////////////////
```

```
public static byte Number_Of_Bytes_In_CBUF3(ref byte p_used_bytes,
                                             ref byte p_unused_bytes)
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      Total number of bytes that are in CBUF3
    //
    // Inputs:       p_used_bytes   - # of bytes in CBUF3 in use
    //               p_unused_bytes - # of bytes unused in CBUF3
    //
    // Description:  Issues a status request and calculates the total number
    //               of bytes that are in CBUF3.  Total number of bytes is the
    //               sum of p_used_bytes and p_unused_bytes.  You can use this
    //               function to see how much room your LIN slave profile is
    //               using.
    //
    ////////////////////////////////////////////////////////////////////////////
```

Device Functions

The following functions are all found in class PICkitS.Device and are all communication mode independent.

```
        public static bool Initialize_PICkitSerial()
            ////////////////////////////////////////////////////////////////////////////
            //
            // Returns:      True if successful, False if not
            // Inputs:       None
            // Description:  Attempts to establish communication with PICkit Serial
            //               device and initialize communication threads used by
            //               class library.  If multiple devices are attached to
            //               host PC, function will only initialize first device
            //               it finds.
            //
            ////////////////////////////////////////////////////////////////////////////



        public static bool Initialize_PICkitSerial(ushort USBIndex)
            ////////////////////////////////////////////////////////////////////////////
            //
            // Returns:      True if successful, False if not
            // Inputs:       ushort - USBIndex  0 based index of which USB port you
            //                        wish to communicate with
            // Description:  Attempts to establish communication with PICkit Serial
            //               device and initialize communication threads used by
            //               class library.
            //
            ////////////////////////////////////////////////////////////////////////////
```

```
public static bool Initialize_MyDevice(ushort USBIndex,
                                       ushort ProductID)
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       ushort - USBIndex  0 based index of which USB port you
    //                        wish to communicate with
    //               ushort - ProductID  ID of device type
    // Description:  Attempts to establish communication with MyDevice
    //               and initialize communication threads used by
    //               class library.
    //
    ////////////////////////////////////////////////////////////////////////////


public static ushort How_Many_PICkitSerials_Are_Attached()
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      number of PKSA's attached to computer
    // Inputs:       None
    // Description:  Polls USB devices for PKSA productID and vendorID
    //               looks for a maximum of 30 devices
    //
    ////////////////////////////////////////////////////////////////////////////


public static ushort How_Many_Of_MyDevices_Are_Attached(ushort ProductID)
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      number of Microchip devices with ProductID attached to computer
    // Inputs:       None
    // Description:  Polls USB devices for ProductID and Microchip vendorID
    //               looks for a maximum of 30 devices
    //
    ////////////////////////////////////////////////////////////////////////////
```

```
public static void Terminate_Comm_Threads()
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:        void
    // Inputs:         none
    // Description:    Closes communication threads inside of dll.  Use this
    //                 when switching between USB ports, but do not wish to
    //                 close host application
    //
    ////////////////////////////////////////////////////////////////////////////


public static bool ReEstablish_Comm_Threads()
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:        True if successful, False if not
    // Inputs:         none
    // Description:    Re-establishes communication threads inside of dll.  Use
    //                 this when switching between USB ports, but do not wish to
    //                 close host application
    //
    ////////////////////////////////////////////////////////////////////////////
```

```
public static int Get_Script_Timeout()
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      Time, in ms, the dll will wait for any script to complete
    // Inputs:       None
    // Description:  When sending any script to the PICkit Serial Analyzer,
    //               the dll will wait for the script to complete and then
    //               verify there were no status errors.  The script timeout
    //               is the maximum time the dll will wait for the script to
    //               finish.  The default value is 3000 (three seconds).
    //
    ////////////////////////////////////////////////////////////////////////////

public static void Set_Script_Timeout(int p_time)
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      void
    // Inputs:       int p_time  -  time in ms the dll will wait for any
    //                              script to complete
    // Description:  When sending any script to the PICkit Serial Analyzer,
    //               the dll will wait for the script to complete and then
    //               verify there were no status errors.  The script timeout
    //               is the maximum time the dll will wait for the script to
    //               finish.  The default value is 3000 (three seconds).
    //
    ////////////////////////////////////////////////////////////////////////////

public static void Cleanup()
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      Void
    // Inputs:       None
    // Description:  Shuts down communication threads and closes file handles.
    //               Must be performed prior to closing host application.
    //
    ////////////////////////////////////////////////////////////////////////////
```

```
public static bool Clear_Comm_Errors()
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:       True if successful, false if not
    // Inputs:        None
    // Description:   Attempts to clear status flags set during a read or
    //                write error by issuing a commreset then a warm reset while
    //                preserving control block contents.  Also resets I2C Slave
    //                buffers.  Issue this function call after a read or write
    //                failure (and you have collected failure status via
    //                Get_Status if desired).
    //
    //                NOTE 1:  This command differs from Reset_Control_Block in
    //                         that power out of the PKSA device is not
    //                         interrupted as a result of this call, as is the
    //                         case in Reset_Control_Block.  This function is
    //                         essentially a cleaner implementation of
    //                         Reset_Control_Block.
    //
    //                NOTE 2:  PKSA firmware version 0202 or greater is required
    //                         for this command to work.  No errors are generated
    //                         by callling this command on an older version of
    //                         the firmware, but the PKSA status flags may not be
    //                         successfully reset and the PKSA power output may
    //                         be interrupted.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Reset_Control_Block()
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       None
    // Description:  Attempts to clear status flags set during a read or
    //               write error by issuing cold then warm resets while
    //               preserving control block contents.  Issue this function
    //               call after a read or write failure (and you have
    //               collected failure status via There_Is_A_Status_Error
    //               if desired).
    //
    ////////////////////////////////////////////////////////////////////////////


public static void Flash_LED1_For_2_Seconds()
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      void
    // Inputs:       None
    // Description:  Kicks off thread that flashes LED1 (Busy) for ~ 2 seconds
    //               Used to help identify which PKSA you are communicating
    //               with in multiple PKSA configurations.
    //
    ////////////////////////////////////////////////////////////////////////////
```

```
public static bool Get_Buffer_Flush_Parameters(ref bool p_flush_on_count,
                                               ref bool p_flush_on_time,
                                               ref byte p_flush_byte_count,
                                               ref double p_flush_interval)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      true if successfull, false if not
    // Inputs:       bool p_flush_on_count   - tells PKSA whether or not to
    //                                         flush buffer on byte count
    //               bool p_flush_on_time    - tells PKSA whether or not to
    //                                         flush buffer on time interval
    //               byte p_flush_byte_count - PKSA will flush buffer when
    //                                         byte count has reached this
    //                                         value if p_flush_on_count = true
    //               double p_flush_interval - time, in ms, between forced
    //                                         flushing of buffer if
    //                                         p_flush_on_time = true
    //                                         (.409 <= p_flush_interval <= 104.3)
    // Description:  Gets parameters that control when CBUF2 is flushed.  CBUF2
    //               contains data from the PKSA that is ready for the host.
    //               You can set the buffer to flush when a certain number of
    //               bytes have arrived, when a certain time interval has
    //               expired, or both.  The coded value of flush time is set by:
    //               value = p_flush_interval/.409
    //
    //               Default values are:
    //               p_flush_on_count = true
    //               p_flush_on_time = true
    //               p_flush_byte_count = 0x0A
    //               p_flush_interval = 104.3ms (raw value 0xFF)
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Set_Buffer_Flush_Parameters(bool p_flush_on_count,
                                               bool p_flush_on_time,
                                               byte p_flush_byte_count,
                                               double p_flush_interval)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:       true if successfull, false if not
    // Inputs:        bool p_flush_on_count   - tells PKSA whether or not to
    //                                          flush buffer on byte count
    //                bool p_flush_on_time    - tells PKSA whether or not to
    //                                          flush buffer on time interval
    //                byte p_flush_byte_count - PKSA will flush buffer when
    //                                          byte count has reached this
    //                                          value if p_flush_on_count = true
    //                double p_flush_interval - time, in ms, between forced
    //                                          flushing of buffer if
    //                                          p_flush_on_time = true
    //                                          (.409 <= p_flush_interval <= 104.3)
    // Description:   Sets parameters that control when CBUF2 is flushed.  CBUF2
    //                contains data from the PKSA that is ready for the host.
    //                You can set the buffer to flush when a certain number of
    //                bytes have arrived, when a certain time interval has
    //                expired, or both.  The coded value of flush time is set by:
    //                value = p_flush_interval/.409
    //
    //                Default values are:
    //                p_flush_on_count = true
    //                p_flush_on_time = true
    //                p_flush_byte_count = 0x0A
    //                p_flush_interval = 104.3ms (raw value 0xFF)
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static double Get_Buffer_Flush_Time()
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      Flush time (in ms) of PKSA CBUF2
    // Inputs:       None
    // Description:  Retrieves byte 4 of the control block and returns:
    //               value = byte4 * .409.  Valid values will be in the range
    //               of .409 to 104.3.  Values greater than 104.3 are
    //               to be considered an error.
    //
    ////////////////////////////////////////////////////////////////////////////

public static bool Set_Buffer_Flush_Time(double p_time)
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      true if successfully sets CBUF2 flush time
    // Inputs:       double p_time - time in ms (.409 <= p_time <= 104.3)
    // Description:  Sets bit 4 of the control block to flush the PKSA
    //               CBUF2 buffer on p_time intervals (if not empty).
    //               Value of byte 4 is set by: value = p_time / .409
    //
    ////////////////////////////////////////////////////////////////////////////
```

```
public static bool Get_Status_Packet(ref byte[] p_status_packet)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if the status packet is successfullu updated,
    //               False if there is an error updating the status packet.
    // Inputs:       p_status_packet - Reference to an array of bytes.
    //                                 The array must be at least 65 bytes
    //                                 long to hold the status packet data.
    // Description:  Issues a command to update the Status Packet stored in
    //               the class library and copy it to p_status_packet.
    //               Call this function to get detailed information regarding
    //               the status block and control block of the PICkit Serial.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool There_Is_A_Status_Error(ref uint p_error)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if there is an error in the status block, false if not
    // Inputs:       p_error - reference to an unsigned 32 bit integer
    // Description:  Copies exec errors into byte 0
    //               Copies comm errors into byte 1
    //               Copies mode errors into byte 2 (i2c, spi, usart, lin, etc)
    //               see relavent sections of the firmware specs for details
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Get_PKSA_FW_Version(ref ushort p_version, ref string p_str_fw_ver)
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      true if able to read PKSA firmware version, false if not
    // Inputs:       ushort p_version    - reference to value that will contain
    //                                     the firmware version if successful
    //               string p_str_fw_ver - reference to a string that will
    //                                     hold the hex representation of the
    //                                     firmware version if successful
    // Description:  Attempts to obtain the firmware version of the attached
    //               PICkit Serial Analyzer by reading a status packet and
    //               interpretting bytes 3 and 4 as the version number.
    //
    ////////////////////////////////////////////////////////////////////////////

public static bool Get_PickitS_DLL_Version(ref string p_version)
    ////////////////////////////////////////////////////////////////////////////
    //
    // Returns:      true if able to read PKSA DLL version, false if not
    // Inputs:       string p_str_fw_ver - reference to a string that will hold
    //                                     the decimal representation of the
    //                                     DLL version if successful
    // Description:  Attempts to obtain the PICkit Serial DLL version  by
    //               reading the FileVersionInfo associated with the DLL.
    //
    ////////////////////////////////////////////////////////////////////////////
```

USBRead Functions

The following delegate is found in class PICkitS.USBRead.

```
        public static event DataNotifier DataAvailable;
            ///////////////////////////////////////////////////////////////////////////
            //
            // Description:  A delegate that is triggered whenever data is available
            //               to be read from the dll.  Typically used in asynchronous
            //               USART communication.  The user should register
            //               DataAvailable during initialization and associate the
            //               trigger with their own routine to retrieve data.  See
            //               the USART Example GUI.doc for an example.
            //
            ///////////////////////////////////////////////////////////////////////////
```

mTouchCap Functions

The following functions are all found in class PICkitS.mTouchCap and are relevant only to devices using I2C Microchip mTouch Capacitive firmware.

```
        public static bool ReadRawAvg(byte p_slave_addr, byte p_index,
                                      byte p_num_sensors, ref byte[] p_data_array)
            ////////////////////////////////////////////////////////////////////////////
            //
            // Returns:      True if successful, False if not
            // Inputs:       byte p_slave_addr   - I2C slave address of mTouch device
            //               byte p_index        - Zero based index of sensor to be read
            //               byte p_num_sensors  - How many sensors worth of data to read
            //               byte[] p_data_array - reference to byte array that will
            //                                     store retrieved data - must be at
            //                                     least as large as p_num_sensors * 4
            //
            // Description:  Reads raw and avg data for p_num_sensors, starting
            //               at p_index.  Data is returned in p_data_array in the format:
            //               [raw1U][raw1L][raw2U][raw2L]..[avg1U][avg1L][avg2U][avg2L]..
            //
            ////////////////////////////////////////////////////////////////////////////

        public static bool ReadNumSensors(byte p_slave_addr, ref byte p_num_sensors)
            ////////////////////////////////////////////////////////////////////////////
            //
            // Returns:      True if successful, False if not
            // Inputs:       byte p_slave_addr   - I2C slave address of mTouch device
            //               byte p_num_sensors  - How many sensors associated with
            //                                     device at this slave address
            //
            // Description:  Queries mTouch device at p_slave_address.  Device should
            //               respond with the number of sensors associated with that
            //               address.  That value is written into p_num_sensors.
            //
            ////////////////////////////////////////////////////////////////////////////
```

```
  public static bool ReadAllData(byte p_slave_addr, byte p_index,
                                 byte p_num_sensors, ref byte[] p_data_array)
      ////////////////////////////////////////////////////////////////////////////
      //
      // Returns:      True if successful, False if not
      // Inputs:       byte p_slave_addr   - I2C slave address of mTouch device
      //               byte p_index        - Zero based index of sensor to be read
      //               byte p_num_sensors  - How many sensors worth of data to read
      //               byte[] p_data_array - reference to byte array that will
      //                                     store retrieved data - must be at
      //                                     least as large as p_num_sensors * 8
      //
      // Description:  Reads raw, avg, trip, and guardband data for p_num_sensors,
      //               starting at p_index.  Data is returned in p_data_array
      //               in the format:
      //         [raw1U][raw1L][raw2U][raw2L]..[avg1U][avg1L][avg2U][avg2L]..
      //         [tri1U][tri1L][tri2U][tri2L]..[gba1U][gba1L][gba2U][gba2L]..
      //
      ////////////////////////////////////////////////////////////////////////////

public static bool WriteTripGuardband(byte p_slave_addr, byte p_index,
                                      ushort p_trip, ushort p_guardband)
      ////////////////////////////////////////////////////////////////////////////
      //
      // Returns:      True if successful, False if not
      // Inputs:       byte p_slave_addr  - I2C slave address of mTouch device
      //               byte p_index       - Zero based index of sensor to write to
      //               ushort p_trip      - trip value to write
      //               ushort p_guardband - guardband value to write
      //
      // Description:  Attempts to write trip and guarband values to sensor
      //               p_index and slave address p_slave_address.  If successful,
      //               the user must still verify the results by calling
      //               ReadAllData and checking the returned trip and guardband
      //               values agains p_trip and p_guardband.
      ////////////////////////////////////////////////////////////////////////////
```

```
public static bool ReadDevToolData(byte p_slave_addr, byte p_num_bytes,
                                   ref byte[] p_data_array)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       byte p_slave_addr   - I2C slave address of mTouch device
    //               byte p_num_bytes    - How many bytes of data to read
    //               byte[] p_data_array - reference to byte array that will
    //                                     store retrieved data
    //
    // Description:  Reads device specific data for associated p_slave_addr
    ///////////////////////////////////////////////////////////////////////////
```

mTouch2 Functions

The following functions are all found in class PICkitS.mTouch2 and are relevant only to devices using Microchip mTouch2 (Touch Sense 2) firmware.

```
    public static bool Configure_PICkitSerial_For_MTouch2()
            ////////////////////////////////////////////////////////////////////////////
            //
            // Returns:      True if successful, False if not
            // Inputs:       None
            // Description:  Configures PICkit Serial control block for
            //               mTouch2 communication and tells class library
            //               to interpret incoming data as same.
            //
            ////////////////////////////////////////////////////////////////////////////

    public static bool Send_MT2_RESET_Command()
            ////////////////////////////////////////////////////////////////////////////
            //
            // Returns:      true if command was successfully sent, false if not
            // Inputs:       None
            // Description:  Sends MT2_RESET command.
            //
            ////////////////////////////////////////////////////////////////////////////

    public static bool Send_MT2_ARCHIVE_Command()
            ////////////////////////////////////////////////////////////////////////////
            //
            // Returns:      true if command was successfully sent, false if not
            // Inputs:       None
            // Description:  Sends MT2_ARCHIVE command.
            //
            ////////////////////////////////////////////////////////////////////////////
```

```
public static bool Send_MT2_COMM_TAG_WR_USE_USB_Command(bool p_enable)
        ///////////////////////////////////////////////////////////////////////////
        //
        // Returns:      true if command was successfully sent, false if not
        // Inputs:       bool    p_enable  -  enable or disable USB input for
        //                                     trip and guardband
        // Description:  Sends MT2_COMM_TAG_WR_USE_USB command.  Sending a p_enable
        //               value of true enables the firmware to accept user defined
        //               values for Trip and Guardband.  Sending a value of
        //               false causes the firmware to use default values for
        //               Trip and Guardband.
        //
        ///////////////////////////////////////////////////////////////////////////


public static bool Create_User_Defined_Sensor_Group(byte p_sensor_count,
                                                    ref byte[] p_sensor_array)
        ///////////////////////////////////////////////////////////////////////////
        //
        // Returns:      true if user defined sensor group is successfully created,
        //               false if not
        // Inputs:       byte   p_sensor_count  -  number of sensors in group
        //               byte[] p_sensor_array  -  reference to array that contains
        //                                          sensor id's to be placed in group
        // Description:  Sends MT2_WR_USERGROUP command to create a user defined
        //               group of sensor id's.  Data from these sensors can then
        //               be obtained by using the user-defined group id 0x80.  See
        //               the mTouch2 firmware documentation for more detail.
        //
        ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Read_User_Defined_Sensor_Group(ref byte p_sensor_count,
                                                  ref byte[] p_sensor_array)
        ///////////////////////////////////////////////////////////////////////////
        //
        // Returns:      true if user defined sensor group is successfully read,
        //               false if not
        // Inputs:       byte   p_sensor_count  -  reference to number of sensors
        //                                         in group
        //               byte[] p_sensor_array  -  reference to array to place
        //                                         sensor id's in group
        // Description:  Sends MT2_RD_USERGROUP command to obtain status of user
        //               defined group.  This command does not read data from the
        //               user defined group, but simply verifies the count and
        //               id's of sensors in the group.
        //
        ///////////////////////////////////////////////////////////////////////////


public static bool Write_Trip_Value(byte p_sensor_id, ushort p_trip)
        ///////////////////////////////////////////////////////////////////////////
        //
        // Returns:      true if trip value was successfully written, false if not
        //
        // Inputs:       byte sensor_id  -  id of individual sensor to write to
        //               ushort p_trip   -  value of trip
        //
        // Description:  Writes the trip value p_trip to sensor p_sensor_id,
        //               then reads back value for verification.
        //
        ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Write_Gdbnd_Value(byte p_sensor_id, ushort p_gdbnd)
        ///////////////////////////////////////////////////////////////////////////
        //
        // Returns:      true if guardband value was successfully written,
        //               false if not
        // Inputs:       byte sensor_id  -  id of individual sensor to write to
        //               ushort p_gdbnd  -  value of guardband
        //
        // Description:  Writes the guardband value p_gdbnd to sensor p_sensor_id,
        //               then reads back value for verification.
        //
        ///////////////////////////////////////////////////////////////////////////


public static bool Get_Trip_and_Gdbnd_Data(byte p_sensor_id, int p_num_sensors,
                                           ref ushort[] p_trip, ref ushort[] p_gdbnd)
        ///////////////////////////////////////////////////////////////////////////
        //
        // Returns:      true if trip and guardband values are successfully
        //               retrieved, false if not
        // Inputs:       byte sensor_id     -  id of individual sensor or group id
        //               int p_num_sensors  - number of sensors for this group
        //               ushort[]           - p_trip reference to array that will
        //                                    hold trip data
        //               ushort[]           - p_gdbnd reference to array that will
        //                                    hold guardband data
        //
        // Description:  Reads trip and guardband data associated with p_sensor_id.
        //               p_sensor_id can either be an individual sensor or a group
        //               id.
        //
        ///////////////////////////////////////////////////////////////////////////
```

```
public static bool Get_Sensor_Data(byte p_sensor_id, byte p_num_sensors,
                                   ref ushort[] p_raw, ref ushort[] p_avg,
                                   ref ushort[] p_trip, ref ushort[] p_gdbnd,
                                   ref byte[] p_detect)
        ///////////////////////////////////////////////////////////////////////////
        //
        // Returns:      true if correct sensor data is obtained, false if not
        //
        // Inputs:       byte      p_sensor_id   - id of desired sensor or group
        //               byte      p_num_sensors - number of sensors associated
        //                                         with p_sensor_id
        //               ushort[]  p_raw         - reference to array that will
        //                                         hold raw data
        //               ushort[]  p_avg         - reference to array that will
        //                                         hold avg data
        //               ushort[]  p_trip        - reference to array that will
        //                                         hold trip data
        //               ushort[]  p_gdbnd       - reference to array that will
        //                                         hold guardband data
        //               byte[]    p_detect      - reference to array that will
        //                                         hold detect data
        //
        // Description:  Sends MT2_RD command and waits up to 500ms for response.
        //               If command returns matching sensor id and number of sensors,
        //               data is copied into passed arrays and a value of true is
        //               returned.  Otherwise a value of false is returned.
        //
        ///////////////////////////////////////////////////////////////////////////
```

mTouchLV Functions

```
public static bool InitializePksaForMtouchLV()
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:       True if successful, False if not
    // Inputs:        None
    //
    // Description:   Initializes the PKSA Dll read and write threads and
    //                configures the PKSA for I2C(mTouchLV) communication.
    //                Also optimizes PKSA configuration for minimum response
    //                time and sets the DLL global script timeout to 100ms.
    //
    ///////////////////////////////////////////////////////////////////////////

public static void Cleanup()
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:       Void
    // Inputs:        None
    // Description:   Shuts down communication threads and closes file handles.
    //                Must be performed prior to closing host application.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool SelectKeypad(byte p_selection)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       byte p_selection   - 0x00 buttons
    //                                    0x01 matrix
    //
    // Description:  Directs firmware to poll data for buttons or matrix,
    //               then verifies firmware status.  p_selection values other
    //               than 0x00 or 0x01 will cause the function to fail, and
    //               the firmware to remain in its existing state.
    //
    ///////////////////////////////////////////////////////////////////////////

public static bool SelectMode(byte p_mode)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       byte p_mode    - 0x00 Fastest
    //                                0x01 Suggested
    //                                0x02 MaxAccuracy
    //
    // Description:  Directs firmware to operate in one of three modes, trading
    //               speed for accuracy, then verifies firmware status.  p_mode
    //               values other than 0x00, 0x01 or 0x02 will cause the
    //               function to fail, and the firmware to remain in its
    //               existing state.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool ReadRawSensorData(int p_num_reads,
                                     ref string[] p_data_array,
                                     ref string p_status_str)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:       True if successful, False if not
    // Inputs:        byte p_num_reads      - How many times to read all
    //                                        NUM_SENSORS sensors
    //                string[] p_data_array - reference to string array that
    //                                        will store retrieved data plus
    //                                        timestamp.  Each array element
    //                                        will consist of a coma delimited
    //                                        string consisting of a timestamp
    //                                        followed by rawdata.  Array must
    //                                        be at least as large as
    //                                        p_num_reads.
    //                string p_status_str   - reference to string that will hold
    //                                        read status, including error
    //                                        information should the function fail
    //
    // Description:   Reads raw data for for all NUM_SENSORS sensors by calling
    //                ReadBuffer p_num_reads times.  Prior to each read, the
    //                ReadBufferIsReady function is called to verify data is
    //                available.  If data is not available, the ReadBufferIsReady
    //                function is repeatedly called until data is available, or
    //                100 attempts have failed - in which case a False is returned.
    //                If p_num_reads = 0, function calls WriteDefaultSettings()
    //                and p_status_str is set to "PASS" or "FAIL" depending on
    //                result.
    //
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool ReadBuffer(ref byte[] p_data_array)
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if data is read, False if not
    // Inputs:       byte[] p_data_array - reference to byte array that will
    //                                     store retrieved data - must be at
    //                                     least 69 byes long
    //
    // Description:  Returns three sets of consecutive sensor data.
    //               There are 11 sensors so will return 3 * 11 * 2 bytes,
    //               plus a two byte timestamp and a one byte buffer status flag.
    //               Data is of format:
    //
    //               0      buff_ready_status               <1= good, 0= fail>
    //               1      timestamp_buffer1L              <only 1 timestamp>
    //               2      timestamp_buffer1H              <per buffer, not per set>
    //               3      set_0_raw0L
    //               4      set_0_raw0H
    //               5      set_0_raw1L
    //               6      set_0_raw1H
    //               …
    //               25 set_1_raw0L                         <start of set1 data>
    //               …
    //               47 set_2_raw0L                         <start of set2 data>
    //               …
    //               67 set_2_raw10L
    //               68 set_2_raw10H
    //
    //               NOTE - It is the calling function's responsibility to
    //                      verify the value of the status byte.
    ///////////////////////////////////////////////////////////////////////////
```

```
public static bool WriteDefaultSettings()
    ///////////////////////////////////////////////////////////////////////////
    //
    // Returns:      True if successful, False if not
    // Inputs:       None
    //
    // Description:  Reinitializes the device's capacitive sensing:
    //               - Re-establish the average with current sensor values
    //               - Use SUGGESTED mode
    //               - Use 8-Button Keypad
    //
    ///////////////////////////////////////////////////////////////////////////
```

Firmware Functions

```
    public static bool Load_Firmware(string p_file_name, int p_device_type,
                                     ref string p_error_str, ref int p_error_code)

            ///////////////////////////////////////////////////////////////////////////
            //
            // Returns:      True if successful, False if not
            // Inputs:       p_file_name - firmware hex file name, including full path
            //               p_device_type - 1 for PKSA device
            //                               2 for LIN device
            //               p_error_str - reference to a string that will contain
            //                             a detailed error msg if the function fails
            //               p_error_code - flag that details the reason for
            //                              function failure:
            //                              0 - no failure, firmware correctly loaded
            //                              1 - error reading hex file
            //                              2 - error entering programming mode
            //                              3 - error clearing flash
            //                              4 - error writing flash
            //                              5 - error writing config bytes
            //                              6 - error reading flash
            //                              7 - error reading config bytes
            //                              8 - error verifying flash
            //                              9 - error verifying config bytes
            //                              10 - error issuing reset/exiting programming mode
            //
            // Description:  Attempts to download firmware from hex file p_file_name
            //               into PKSA or LIN device.  The steps are:
            //               - read hex file
            //               - force device into programming mode
            //               - verify device is in programming mode
            //               - clear old flash
            //               - write new flash
            //               - write new config bytes
            //               - read new flash
```

```
//                 - verify new flash
//                 - read new config bytes
//                 - verify new config bytes
//                 - reset device to force out of bootloader mode
//
//                 Typical times for this function to complete are:
//                 PKSA: about 60 seconds
//                 LIN:  about 30 seconds
//
// NOTE:           After the function has completed, the PKSA must be
//                 configured for a specific protocol, but does not need
//                 to be re-initialized.  For example, you do not need to
//                 call Device.Initialize_PICkitSerial(), but you will need
//                 to call I2CM.Configure_PICkitSerial_For_I2CMaster().
///////////////////////////////////////////////////////////////////////////
```