MLX90614 PRODUCT SPECIFIC FUNCTIONS

Software Library

1 Contents

2 Introduction

MLX90614 PSF is MS Windows software library, which meets the requirements for a Product Specific Functions (PSF) module, defined in Melexis Programmable Toolbox (MPT) object model. The library implements in-process COM objects for interaction with MLX90614 EVB firmware. It is designed primarily to be used by MPT Framework application, but also can be loaded as a standalone in-process COM server by other applications that need to communicate with the above-mentioned Melexis hardware.

The library can be utilized in all programming languages, which support ActiveX automation. This gives great flexibility in designing the application with the only limitation to be run on MS Windows OS. In many scripting languages, objects can be directly created and used. In others, though, the first step during implementation is to include the library in your project. The way it can be done depends on the programming language and the specific Integrated Development Environment (IDE) used:

- in C++ it can be imported by #import directive
- in Visual Basic it either can be directly used as pure Object or added as a reference to the project
- in C# it has to be added as a reference to the project
- in NI LabView, for each Automation refnum the corresponding ActiveX class has to be selected
- in NI LabWindows an ActiveX Controller has to be created

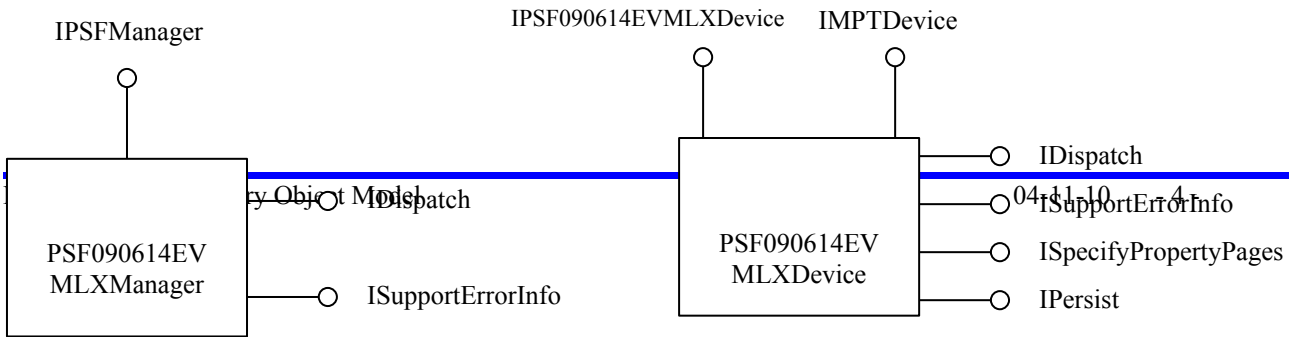
3 Software Structure

3.1 Object Model

MPT object model specifies that a PSF module must expose two COM objects which implement certain COM interfaces. MLX90614 PSF implements these two objects and two additional objects for advanced operations.

- **PSF090614EVMLXManager object** – implements IPSFManager standard MPT interface. This is a standard PSFManager object. MPT Framework and other client applications create a temporary instance of that object, just for device scanning procedure. After that this instance is released.
  This is the first required object. Refer to MPT Developer Reference document for more information about PSFManager object and IPSFManager interface.
- **PSF090614EVMLXDevice object** – implements IPSF090614EVMLXDevice specific interface. However, this interface derives from IMPTDevice standard MPT interface and therefore PSF090614EVMLXDevice also implements the functionality of MPTDevice standard MPT object. In addition to standard IMPTDevice methods, IPSF090614EVMLXDevice interface exposes methods, which are specific to this library. They are described in this document.
  This is the second required COM object. Refer to MPT Developer Reference document for more information about MPTDevice object and IMPTDevice interface.

3.2 Objects with Interfaces



4 PSF090614EVMLXManager Object

4.1 Background

This object is created only once and is destroyed when the library is unmapped from process address space. Each subsequent request for this object returns the same instance.

PSF090614EVMLXManager object implements standard MPT category **CATID_MLXMPTPSFUSBHIDModule**, which is required for automatic device scanning. C++ standalone client applications can create an instance of this object by using the standard COM API CoCreateInstance with class ID **CLSID_PSF090614EVMLXManager**, or ProgID **"MPT. PSF090614EVMLXManager"**:

```
hRes = ::CoCreateInstance(CLSID_PSF090614EVMLXManager, NULL, CLSCTX_INPROC,
                          IID_IPSFManager, (void**) &pPSFMan);
```

Visual Basic applications should call CreateObject function to instantiate PSF090614EVMLXManager:

```
Set PSFMan = CreateObject("MPT. PSF090614EVMLXManager")
```

The primary objective of this instantiation is to call ScanStandalone method. C++:

```
hRes = pPSFMan->ScanStandalone(dtUSBHID, varDevices, &pDevArray);
```

Or in Visual Basic:

```
Set DevArray = PSFMan.ScanStandalone(dtUSBHID)
```

ScanStandalone function returns collection of PSF090614EVMLXDevice objects, one for each connected MLX90614 EVB. The collection is empty if there are no connected evaluation boards.

5 PSF090614EVMLXDevice Object

5.1 Background

This object implements standard MPT category **CATID_MLXMPTPSFUSBHIDDevice** as well as library specific **CATID_MLXMPT90614EVBDevice** category. It also declares required specific category **CATID_MLXMPT90614EVBUIModule** for identification of required user interface modules.

This object can be created directly with CoCreateInstance/CreateObject or by calling the device scanning procedure ScanStandalone of PSF090614EVMLXManager object. The following Visual Basic subroutine shows how to instantiate PSF090614EVMLXDevice object by performing device scan on the system:

```
Sub CreateDevice()
   Dim PSFMan As PSF090614EVMLXManager, DevicesCol As ObjectCollection, I As Long
   On Error GoTo lError

   Set PSFMan = CreateObject("MPT.PSF090614EVMLXManager")
   Set DevicesCol = PSFMan.ScanStandalone(dtUSBHID
   If DevicesCol.Count <= 0 Then
      MsgBox ("No EVB90614 devices were found!")
      Exit Sub
   End If

   ' Dev is a global variable of type PSF090614EVMLXDevice
   ‘ Select first device from the collection
   Set Dev = DevicesCol(0)
   MsgBox (Dev.Name & " device found on " & Dev.Channel.Name)
   If DevicesCol.Count > 1 Then
      For I = 1 To DevicesCol.Count - 1
         ' We are responsible to call Destroy(True) on the device objects we do not need
         Call DevicesCol(I).Destroy(True)
      Next I
   End If
   Exit Sub

lError:
   MsgBox Err.Description
   Err.Clear
End Sub
```

Developers can also manually connect the device object to a USB HID channel object thus bypassing standard device scanning procedure. The following Visual Basic subroutine allows manual connection along with standard device scanning depending on input parameter bAutomatic:

```
Sub CreateDevice(bAutomatic As Boolean)
   Dim PSFMan As PSF090614EVMLXManager, DevicesCol As ObjectCollection, I As Long
   Dim CommMan As CommManager, Chan As MPTChannel
   On Error GoTo lError

   If bAutomatic Then
      ' Automatic device scanning begins here
      Set PSFMan = CreateObject("MPT.PSF090614EVMLXManager")
      Set DevicesCol = PSFMan.ScanStandalone(dtUSBHID)
      If DevicesCol.Count <= 0 Then
         MsgBox ("No EVB90614 devices were found!")
         Exit Sub
      End If

      If DevicesCol.Count > 1 Then
         For I = 1 To DevicesCol.Count - 1
            'We are responsible to call Destroy(True) on device objects we do not need
            Call DevicesCol(I).Destroy(True)
         Next I
      End If
      Set MyDev = DevicesCol(0)
   Else
```

```
    ' Manual connection begins here
    Set CommMan = CreateObject("MPT.CommManager")
    Set MyDev = CreateObject("MPT.PSF090614EVMLXDevice")
    I = ActiveWorkbook.Names("USB HID Port").RefersToRange.Value2
    Set Chan = CommMan.Channels.CreateChannel(CVar(I), ctUSBHID)
    MyDev.Channel = Chan
    ' Check if an EVB is connected to this channel
    Call MyDev.CheckSetup(False)
  End If
  MsgBox (MyDev.Name & " device found on " & MyDev.Channel.Name)
  Exit Sub

IError:
  MsgBox Err.Description
  Err.Clear
End Sub
```

PSF090614EVMLXDevice object implements IMPTDevice standard MPT interface. Please refer to MPT Developer reference document for description of the properties and methods of this interface.

In addition PSF090614EVMLXDevice object implements IPSF090614EVMLXDevice library specific interface, which derives from IMPTDevice. The following is a description of its properties and methods.

5.2 Scope of the PSF090614EVMLXDevice object

This object supports all needs for a standard user.
With these basic functions, you're able to discover this Melexis Product.

5.3 ReadFullDevice Method

5.3.1 Description

Reads the whole EEPROM of the device. Updates the internal EEPROM cache with values taken from the module.

5.3.2 Syntax

Visual Basic:

```
Sub ReadFullDevice()
```

C++:

```
HRESULT ReadFullDevice();
```

5.3.3 Parameters

None

5.3.4 Return value

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	Meaning
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.4 ProgramDevice Method

5.4.1Description

Programs the EEPROM of the device. Takes the values from the internal EEPROM cache.
Only the variables that are modified will be programmed.

5.4.2Syntax

Visual Basic:

Sub ProgramDevice()

C++:

HRESULT ProgramDevice();

5.4.3Parameters

None

5.4.4Return value

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.5 DeviceReplaced Method

5.5.1Description

Informs the object that the sensor is replaced and the EEPROM cache and some internal variables should be invalidated.

5.5.2Syntax

Visual Basic:

Sub DeviceReplaced()

C++:

HRESULT DeviceReplaced();

5.5.3Parameters

None

5.5.4Return value

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.6 GetEEParameterCode Method

5.6.1Description

Returns the code of a particular EEPROM parameter as it is represented in EEPROM. It is optimized because it uses the EEPROM cache maintained by the library. **ReadFullDevice** method could be called before calling **GetEEParameterCode** to update the whole cache. However **GetEEParameterCode** works correctly even if **ReadFullDevice** is not called.

5.6.2Syntax

Visual Basic:

Function GetEEParameterCode(paramID as ParamCodesEEPROM) as Long

C++:

HRESULT GetEEParameterCode(/*[in]*/ ParamCodesEEPROM paramID, /*[out,retval]*/ long* pVal);

5.6.3Parameters

paramID

A **ParamCodesEEPROM** constant specifying the ID of the EEPROM parameter.

pVal

An address of **Long** variable that will receive the return value of the method.

5.6.4Return value

Visual Basic:

A **Long** containing the code of an EEPROM parameter.

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully. *pVal contains a valid value.
Any other error code	The operation failed. *pVal is **0**.

5.7 SetEEParameterCode Method

5.7.1Description

Changes the code of a particular EEPROM parameter. The method works with the EEPROM cache maintained by the library.
ProgramDevice method must be called in order to update the EEPROM of the module with the codes from the cache.

5.7.2Syntax

Visual Basic:

```
Sub SetEEParameterCode(paramID as ParamCodesEEPROM, Value as Long)
```

C++:

```
HRESULT SetEEParameterCode(/*[in]*/ ParamCodesEEPROM paramID, /*[in]*/ long
        Value);
```

5.7.3Parameters

paramID
A ParamCodesEEPROM constant specifying the ID of the EEPROM parameter.

Value
A **Long** containing new code for the parameter.

5.7.4Return value

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	Meaning
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.8 GetEEParameterValue Method

5.8.1Description

Returns the translated value of a particular EEPROM parameter. It first calls GetEEParameterCode method and then translates the code of the parameter into a suitable value.
Translation is not defined for all parameters and this method returns an error if it receives paramID which is not supported.

5.8.2Syntax

Visual Basic:

```
Function GetEEParameterValue(paramID as ParamCodesEEPROM)
```

C++:

HRESULT GetEEParameterValue(/*[in]*/ ParamCodesEEPROM paramID, /*[out,retval]*/ TVariant* pVal);

5.8.3Parameters

paramID
A ParamCodesEEPROM constant specifying the ID of the EEPROM parameter.

pVal
An address of **VARIANT** variable that will receive the return value of the method. The caller is responsible to call VariantClear on that variable when it is no longer needed.

5.8.4Return value

Visual Basic:
A **Variant** containing the translated value of an EEPROM parameter.

C++:
The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully. *pVal contains a valid value.
Any other error code	The operation failed. *pVal is **Empty**.

5.9 SetEEParameterValue Method

5.9.1Description

Changes the value of a particular EEPROM parameter. It first translates the value to a corresponding code and then calls SetEEParameterCode method to modify the parameter in the cache.
Translation is not defined for all parameters and this method returns an error if it receives paramID which is not supported.
ProgramDevice method must be called in order to update the EEPROM of the module with the codes from the cache.

5.9.2Syntax

Visual Basic:
Sub SetEEParameter(paramID as ParamCodesEEPROM, Value)

C++:
HRESULT SetEEParameter(/*[in]*/ ParamCodesEEPROM paramID, /*[in]*/ TVariantInParam Value);

5.9.3Parameters

paramID
A ParamCodesEEPROM constant specifying the ID of the EEPROM parameter.

Value
A **VARIANT** containing new value for the parameter.

5.9.4Return value

C++:
The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.10CmdCheckModuleVersion Method

5.10.1Description

This method checks the required power supply of the connected module(s) and returns an array of populated SMBus addresses. After successful execution, the bus is remained powered and the modules in SMBus mode.

5.10.2Syntax

Visual Basic:
Function CmdCheckModuleVersion(pVolt As Byte, [Format As Long = 1])

C++:
HRESULT CmdCheckModuleVersion(/*[out]*/ unsigned char* pVolt, /*[in, defaultvalue(1)]*/ long Format, /*[out, retval]*/ VARIANT* pVal);

5.10.3Parameters

pVolt
An address of **Byte** variable that will receive the operating voltage of the connected module(s). Meaning of the value is as follows: 0-no module(s); 3 – 3V module; 5 – 5V module.

Format
A **long** specifying the format of the returned data in pVal. Possible values are:

Value	**Format**
1	Return value is an array of bytes. This is the preferred format for Visual Basic applications. This is the default value.
2	Return value is an ANSI string packed in bstrVal member of *pvarID. This is the preferred format for C++ applications because of the best performance. It is a binary data so the string can contain zeroes and may not be zero terminated. Callers can get its real length by calling SysStringByteLen API on bstrVal member.

pVal
An address of **VARIANT** variable that will receive an array of SMB addresses of the modules populated on the bus. The caller is responsible to call VariantClear on that variable when it is no longer needed.

5.10.4Return value

Visual Basic:

A **Variant** containing an array of SMB addresses of the modules populated on the bus. Representation of the data depends on Format parameter.

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	Meaning
S_OK	The operation completed successfully. *pVal contains a valid value.
Any other error code	The operation failed. *pVal contains an empty variant.

5.11CmdSendRequest Method

5.11.1Description

Switches the operating mode to SMBus.

5.11.2Syntax

Visual Basic:

Sub CmdSendRequest()

C++:

HRESULT CmdSendRequest();

5.11.3Parameters

None

5.11.4Return value

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	Meaning
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.12CmdReportAddresses Method

5.12.1Description

This method returns an array of populated SMBus addresses.

5.12.2Syntax

Visual Basic:

Function CmdReportAddresses([Format As Long = 1])

C++:

HRESULT CmdReportAddresses(/*[in, defaultvalue(1)]*/ long Format,
/*[out, retval]*/ VARIANT* pVal);

5.12.3Parameters

Format

A **long** specifying the format of the returned data in pVal. Possible values are:

Value	Format
1	Return value is an array of bytes. This is the preferred format for Visual Basic applications. This is the default value.
2	Return value is an ANSI string packed in bstrVal member of *pvarID. This is the preferred format for C++ applications because of the best performance. It is a binary data so the string can contain zeroes and may not be zero terminated. Callers can get its real length by calling SysStringByteLen API on bstrVal member.

pVal

An address of **VARIANT** variable that will receive an array of SMB addresses of the modules populated on the bus. The caller is responsible to call VariantClear on that variable when it is no longer needed.

5.12.4Return value

Visual Basic:

A **Variant** containing an array of SMB addresses of the modules populated on the bus. Representation of the data depends on Format parameter.

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	Meaning
S_OK	The operation completed successfully. *pVal contains a valid value.
Any other error code	The operation failed. *pVal contains an empty variant.

5.13CheckPosition Method

5.13.1Description

This method checks whether the module is connected properly and exits with an error if it is not.

5.13.2Syntax

Visual Basic:

Sub CheckPosition()

C++:

HRESULT CheckPosition();

5.13.3Parameters

None

5.13.4Return value

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	Meaning
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.14SetSMBAddress Method

5.14.1Description

Sets the SMB address of the module which will be accessed with the next commands.

5.14.2Syntax

Visual Basic:

Sub SetSMBAddress(Addr As Byte)

C++:

HRESULT SetSMBAddress (/*[in]*/ unsigned char Addr);

5.14.3Parameters

Addr

A **Byte** specifying the SMB address of the module which will be accessed with the next commands.

5.14.4Return value

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	Meaning
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.15CmdSetSMBSpeed Method

5.15.1Description

Sets SMBus speed.

5.15.2Syntax

Visual Basic:

Sub CmdSetSMBSpeed(Speed As Byte)

C++:
HRESULT CmdSetSMBSpeed(/*[in]*/ unsigned char Speed);

5.15.3Parameters

Speed
A **Byte** specifying the SMBus speed.

5.15.4Return value

C++:
The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.16CmdSetSupplyVoltage Method

5.16.1Description

Sets supply voltage.

5.16.2Syntax

Visual Basic:
Sub CmdSetSupplyVoltage(Volt As Byte)

C++:
HRESULT CmdSetSupplyVoltage(/*[in]*/ unsigned char Volt);

5.16.3Parameters

Volt
A **Byte** specifying supply voltage. Meaning is as follows: 0->no supply; 1->5V; 2->2.45V; 3->3V.

5.16.4Return value

C++:
The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.17 CmdRestartModule Method

5.17.1 Description

Restarts the connected module by turning the power off and on.

5.17.2 Syntax

Visual Basic:

Sub CmdRestartModule()

C++:

HRESULT CmdRestartModule();

5.17.3 Parameters

None

5.17.4 Return value

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.18 CmdSetSleepMode Method

5.18.1 Description

Activates or deactivates sleep mode.

5.18.2 Syntax

Visual Basic:

Sub CmdSetSleepMode(btOn As Byte)

C++:

HRESULT CmdSetSleepMode(/*[in]*/ unsigned char btOn);

5.18.3 Parameters

btOn

A **Byte** specifying whether the connected module to be placed in sleep mode (0) or awoken (1).

5.18.4Return value

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.19SetApplicationMode Method

5.19.1Description

This method sets the module in application mode.

5.19.2Syntax

Visual Basic:

Sub SetApplicationMode()

C++:

HRESULT SetApplicationMode();

5.19.3Parameters

None

5.19.4Return value

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.20SetCalibrationMode Method

5.20.1Description

This method sets the module in calibration mode.

5.20.2Syntax

Visual Basic:

Sub SetCalibrationMode()

C++:

HRESULT SetCalibrationMode();

5.20.3Parameters

None

5.20.4Return value

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.21CmdReadRam Method

5.21.1Description

This method reads a word from the specified RAM address of the module.

5.21.2Syntax

Visual Basic:

Function CmdReadRam(Addr as Byte) As Long

C++:

HRESULT CmdReadRam (/*[in]*/ unsigned char Addr, /*[out, retval]*/ long* pVal);

5.21.3Parameters

Addr

A **Byte** specifying the address to be read.

pVal

An address of **Long** variable that will receive data read from the module. Only 16 LSB are meaningful.

5.21.4Return value

Visual Basic:

A **Long** containing the data read from the module. Only 16 LSB are meaningful.

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully. *pVal contains valid value.
Any other error code	The operation failed. *pVal contains 0.

5.22 CmdWriteRam Method

5.22.1 Description

This method writes a data word to the specified RAM address of the module.

5.22.2 Syntax

Visual Basic:

Sub CmdWriteRam(Addr As Byte, Val As Long)

C++:

HRESULT CmdWriteRam(/*[in]*/ unsigned char Addr, /*[in]*/ long Val);

5.22.3 Parameters

Addr
A **Byte** specifying the address to be written.

Val
A **Long** containing the data to be written. Only 16 LSB are meaningful.

5.22.4 Return value

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.23 CmdReadEeprom Method

5.23.1 Description

This method reads a word from the specified EEPROM address of the module.

5.23.2 Syntax

Visual Basic:

Function CmdReadEeprom(Addr as Byte) As Long

C++:

HRESULT CmdReadEeprom (/*[in]*/ unsigned char Addr, /*[out, retval]*/ long* pVal);

5.23.3 Parameters

Addr
A **Byte** specifying the address to be read.

pVal
An address of **Long** variable that will receive data read from the module. Only 16 LSB are meaningful.

5.23.4Return value

Visual Basic:
A **Long** containing the data read from the module. Only 16 LSB are meaningful.

C++:
The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully. *pVal contains valid value.
Any other error code	The operation failed. *pVal contains 0.

5.24CmdWriteEeprom Method

5.24.1Description

This method writes a data word to the specified EEPROM address of the module.

5.24.2Syntax

Visual Basic:
Sub CmdWriteEeprom(Addr As Byte, Val As Long)

C++:
HRESULT CmdWriteEeprom (/*[in]*/ unsigned char Addr, /*[in]*/ long Val);

5.24.3Parameters

Addr
A **Byte** specifying the address to be written.

Val
A **Long** containing the data to be written. Only 16 LSB are meaningful.

5.24.4Return value

C++:
The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.25 CmdEraseEeprom Method

5.25.1 Description

This method erases the specified EEPROM address of the module. Usage of this method is not mandatory, i.e. it is not necessary to call it before CmdWriteEeprom.

5.25.2 Syntax

Visual Basic:

Sub CmdEraseEeprom(Addr As Byte)

C++:

HRESULT CmdEraseEeprom (/*[in]*/ unsigned char Addr);

5.25.3 Parameters

Addr
A **Byte** specifying the address to be erased.

5.25.4 Return value

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.26 CmdCapturePWM Method

5.26.1 Description

This method captures two sequential periods of PWM signal from the module.

5.26.2 Syntax

Visual Basic:

Sub CmdCapturePWM(pPeriod1 As Single, pPulse1 As Single, pPeriod2 As Single, pPulse2 As Single)

C++:

HRESULT CmdCapturePWM(/*[out]*/ float* pPeriod1, /*[out]*/ float* pPulse1, /*[out]*/ float* pPeriod2, /*[out]*/ float* pPulse2);

5.26.3 Parameters

Period1
An address of **Single** variable that will receive the duration of the first period in [s].

Pulse1
An address of **Single** variable that will receive the duration of the first pulse in [s].

Period2
An address of **Single** variable that will receive the duration of the second period in [s].

Pulse2
An address of **Single** variable that will receive the duration of the second pulse in [s].

5.26.4Return value

C++:
The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.27CapturePWM Method

5.27.1Description

This method captures two sequential periods of PWM signal from the module and returns corresponding temperature values. Conversion is done according to data specified in EEPROM.

5.27.2Syntax

Visual Basic:
Sub CapturePWM(pTemp1 As Single, pTemp2 As Single)

C++:
HRESULT CapturePWM(/*[out]*/ float* pTemp1, /*[out]*/ float* pTemp2);

5.27.3Parameters

pTemp1
An address of **Single** variable that will receive the first temperature [ºC].

pTemp2
An address of **Single** variable that will receive the second temperature [ºC].

5.27.4Return value

C++:
The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**

S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.28 ResetHardware Method

5.28.1 Description

Resets the attached device. Also exits from the bootloader mode.

5.28.2 Syntax

Visual Basic:

Sub ResetHardware()

C++:

HRESULT ResetHardware();

5.28.3 Parameters

None

5.28.4 Return value

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.29 GetMainHardwareID Method

5.29.1 Description

Sends GetHardwareID_Main command to the attached device and returns the response.

5.29.2 Syntax

Visual Basic:

Function GetMainHardwareID([Format as Long = 1])

C++:

HRESULT GetMainHardwareID (/*[in]*/ long Format, /*[out][retval]*/ VARIANT* pvarID);

5.29.3 Parameters

Format
A **long** specifying the format of the returned data in pvarID. Possible values are:

Value	Format
1	Return value is an array of bytes. This is the preferred format for Visual Basic applications. This is the default value.
2	Return value is an ANSI string packed in bstrVal member of *pvarID. This is the preferred format for C++ applications because of the best performance. It is a binary data so the string can contain zeroes and may not be zero terminated. Callers can get its real length by calling SysStringByteLen API on bstrVal member.

pvarID
An address of VARIANT variable that will receive the return value of the method. The caller is responsible to call VariantClear on that variable when it is no longer needed.

5.29.4Return value

Visual Basic:
A **Variant** containing the hardware ID.

C++:
The return value obtained from the returned HRESULT is one of the following:

Return value	Meaning
S_OK	The operation completed successfully. *pvarID contains a valid value.
Any other error code	The operation failed. *pvarID contains zero.

5.30GetSoftwareID Method

5.30.1Description

Sends GetSoftwareID command to the attached device and returns the response.

5.30.2Syntax

Visual Basic:
Function GetSoftwareID([Format as Long = 1])

C++:
HRESULT GetSoftwareID (/*[in]*/ long Format, /*[out][retval]*/ VARIANT* pvarID);

5.30.3Parameters

Format
A **long** specifying the format of the returned data in pvarID. Possible values are:

Value	Format
1	Return value is an array of bytes. This is the preferred format for Visual Basic applications. This is the default value.
2	Return value is an ANSI string packed in bstrVal member of *pvarID. This is the preferred format for C++ applications because of the best performance. It is a binary data so the string can contain zeroes and may not be zero terminated. Callers can get its real length by calling SysStringByteLen API on bstrVal member.

pvarID
An address of VARIANT variable that will receive the return value of the method. The caller is responsible to call VariantClear on that variable when it is no longer needed.

5.30.4Return value

Visual Basic:
A **Variant** containing the software ID.

C++:
The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully. *pvarID contains a valid value.
Any other error code	The operation failed. *pvarID contains zero.

5.31SendCommand Method

5.31.1Description

Sends the requested command to the EVB and returns the response.

5.31.2Syntax

Visual Basic:
Function SendCommand(Cmd as Byte, [vParameters], [Format as Long = 1])

C++:
HRESULT SendCommand((/*[in]*/ unsigned char Cmd, /*[in][optional]*/ VARIANT vParameters, /*[in]*/ long Format, /*[out][retval]*/ VARIANT* pvRes);

5.31.3Parameters

Cmd
A **Byte** specifying the code of the command to send.

vParameters
A **VARIANT** containing optional command parameters. In case the command does not have parameters it must be an empty variant. The value of the *Format* parameter is ignored in the latter case. Optional, the default is an empty variant.

Format
A **long** specifying the format of the returned data in pvarID. Possible values are:

Value	**Format**
1	Return value is an array of bytes. This is the preferred format for Visual Basic applications. This is the default value.
2	Return value is an ANSI string packed in bstrVal member of *pvarID. This is the preferred format for C++ applications because of the best performance. It is a binary

data so the string can contain zeroes and may not be zero terminated. Callers can get its real length by calling SysStringByteLen API on bstrVal member.

pvRes
An address of VARIANT variable that will receive the return value of the method. The caller is responsible to call VariantClear on that variable when it is no longer needed.

5.31.4Return value

Visual Basic:
A **Variant** containing the response from the EVB.

C++:
The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully. *pvRes contains a valid value.
Any other error code	The operation failed. *pvRes contains an empty variant.

5.32EnterBootLoader Method

5.32.1Description

Sends Goto_BootLoader command to the attached device.

5.32.2Syntax

Visual Basic:
Function EnterBootLoader() As Byte

C++:
HRESULT EnterBootLoader(/*[out, retval]*/ unsigned char* pbtMode);

5.32.3Parameters

pbtMode
An address of byte variable that will receive the mode of the bootloader software after executing the command: 1 – start-up mode, 2 – programming mode, 0 – an error has occurred.

5.32.4Return value

Visual Basic:
A **Byte** containing the mode of the bootloader software after executing the command: 1 – start-up mode, 2 – programming mode, 0 – an error has occurred.

C++:
The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully. *pbtMode contains a valid value.

Any other error code	The operation failed. *pbtMode contains zero.

5.33 ExitBootLoader Method

5.33.1 Description

Sends Exit_BootLoader command to the attached device.

5.33.2 Syntax

Visual Basic:

```
Sub ExitBootLoader()
```

C++:

```
HRESULT ExitBootLoader();
```

5.33.3 Parameters

None

5.33.4 Return value

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	Meaning
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.34 BLUploadIntelHexFile Method

5.34.1 Description

Uploads Hex file with firmware into the attached device. First the device is set in bootloader mode. Then the lines from the file are sent. At the end ExitBootloader command is issued in order to start-up the newly uploaded firmware.

5.34.2 Syntax

Visual Basic:

```
Sub BLUploadIntelHexFile(FileName As String, Progress As Object, [vHint])
```

C++:

```
HRESULT BLUploadIntelHexFile(/*[in]*/BSTR FileName,
                             /*[in]*/LPDISPATCH Progress,
                             /*[in,opt]*/VARIANT vHint);
```

5.34.3Parameters

FileName
Specifies full path name of the Hex file.

Progress
Object that implements IMPTProgressCallback interface. It should have implementation of methods OnStart, OnProgress and OnEnd. **Nothing** (**NULL**) can be passed if the callback is not needed.

vHint
A Variant that is sent back to callback object as parameter in Onxxx methods.

5.34.4Return value

C++:
The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.35BLSendIntelHexLine Method

5.35.1Description

Sends SendIntelHexFile command to the attached device. The EVB must be in bootloader programming mode in order to execute this command properly.

5.35.2Syntax

Visual Basic:
Sub BLSendIntelHexLine(vHLine, [Format As Long = 1])

C++:
HRESULT BLSendIntelHexLine(/*[in]*/VARIANT vHLine, /*[in]*/long Format);

5.35.3Parameters

vHLine
Specifies one line of a Hex file to be programmed.

Format
A **long** specifying the format of the data in **vHLine**. Possible values are:

Value	**Format**
1	vHLine is an array of bytes. This is the default value.
2	vHLine is an ANSI string packed in bstrVal member. This is the preferred format for C++ applications because of the best performance. It is a binary data so the string can contain zeroes and may not be zero terminated.
3	vHLine is an Unicode string.

5.35.4Return value

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.36BLVerifyIntelHexFile Method

5.36.1Description

Compares firmware that is currently in the EVB with one in selected hex file. First the device is set in bootloader mode. Then each line from the file is compared with the corresponding area from the EVBs program memory. At the end ExitBootloader command is issued.

5.36.2Syntax

Visual Basic:

```
Sub BLVerifyIntelHexFile(FileName As String, Progress As Object, [vHint])
```

C++:

```
HRESULT BLVerifyIntelHexFile (/*[in]*/BSTR FileName,
                              /*[in]*/LPDISPATCH Progress,
                              /*[in,opt]*/VARIANT vHint);
```

5.36.3Parameters

FileName
Specifies full path name of the Hex file.

Progress
Object that implements IMPTProgressCallback interface. It should have implementation of methods OnStart, OnProgress and OnEnd. **Nothing** (**NULL**) can be passed if the callback is not needed.

vHint
A Variant that is sent back to callback object as parameter in Onxxx methods.

5.36.4Return value

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.37 BLVerifyIntelHexLine Method

5.37.1 Description

Compares one line from the hex file with corresponding data of the firmware currently present in the EVB.

5.37.2 Syntax

Visual Basic:

Sub BLVerifyIntelHexLine(vHLine, [Format As Long = 1])

C++:

HRESULT BLVerifyIntelHexLine(/*[in]*/VARIANT vHLine, /*[in]*/long Format);

5.37.3 Parameters

vHLine

Specifies one line of a Hex file to be compared.

Format

A **long** specifying the format of the data in **vHLine**. Possible values are:

Value	Format
1	vHLine is an array of bytes. This is the default value.
2	vHLine is an ANSI string packed in bstrVal member. This is the preferred format for C++ applications because of the best performance. It is a binary data so the string can contain zeroes and may not be zero terminated.
3	vHLine is an Unicode string.

5.37.4 Return value

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	Meaning
S_OK	The operation completed successfully.
Any other error code	The operation failed.

5.38 ResponseTimeout Property

5.38.1 Description

This property gets/sets the time (in [ms]) that is allowed to elapse before signaling timeout for a particular operation. During this period attached device should start sending the response/acknowledge for the operation or else the communication layer will generate an error.

5.38.2 Syntax

Visual Basic:

Property ResponseTimeout as Long

C++:

HRESULT get_ResponseTimeout(/*[out,retval]*/ long* pValue);
HRESULT set_ResponseTimeout(/*[in]*/ long Value);

5.38.3Parameters

pValue
An address of **long** variable that receives current value (in [ms])of the property.

Value
A **Long** specifying new value (in [ms]) for the property.

5.38.4Return value

Visual Basic:

A **Long** containing current value (in [ms])of the property.

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully. *pValue contains valid value.
Any other error code	The operation failed.

5.39CommunicationLog Property

5.39.1Description

This property specifies whether logging information is generated from the attached communication channel.

5.39.2Syntax

Visual Basic:

Property CommunicationLog as Boolean

C++:

HRESULT get_CommunicationLog(/*[out,retval]*/ VARIANT_BOOL* pValue);
HRESULT set_CommunicationLog(/*[in]*/ VARIANT_BOOL Value);

5.39.3Parameters

pValue
An address of **VARIANT_BOOL** variable that receives current value of the property. **VARIANT_TRUE** means that logging is active, **VARIANT_FALSE** means inactive.

Value
A **VARIANT_BOOL** specifying new value for the property. **VARIANT_TRUE** activates the logging, **VARIANT_FALSE** deactivates it.

5.39.4Return value

Visual Basic:

True if logging is active, **False** otherwise.

C++:

The return value obtained from the returned HRESULT is one of the following:

Return value	**Meaning**
S_OK	The operation completed successfully. *pValue contains valid value.
Any other error code	The operation failed.

6 Enumeration constants

6.1 ParamCodesEEPROM enumeration

The following constants refer to parameters in EEPROM. They are used by **GetEEParameterCode**, **SetEEParameterCode**, **GetEEParameterValue** and **SetEEParameterValue** methods.
Parameters with translation value '-' are not supported by **GetEEParameterValue** and **SetEEParameterValue** methods.

Constant	Value	Bits	Translation value	Description
CodeTomax	1	16	float [ºC]	
CodeTomin	2	16	float [ºC]	
CodePWMControl	3	16	-	
CodeTarange	4	16	-	
CodeKemissivity	5	16	-	
CodeConfig1	6	16	-	
CodeSlaveAddr	7	16	-	
CodeThermoShock	8	16	-	
CodeMovingAverage	9	16	-	
CodeID1	10	16	-	
CodeID2	11	16	-	
CodeID3	12	16	-	
CodeID4	13	16	-	
CodePWMExtended	14	1	-	
CodePWMEnable	15	1	-	
CodePWMSDA	16	1	-	
CodeThermoRelay	17	1	-	
CodePWMPeriodRep	18	5	Byte [times]	
CodePWMPeriod	19	7	float [s]	
CodeTamin	20	8	float [ºC]	
CodeTamax	21	8	float [ºC]	
CodeIIRFilt	22	3	-	
CodePTAT	23	1	-	
CodePWMData	24	2	-	
CodeDualZone	25	1	-	
CodeDAlpha	26	1	-	
CodeFIRFilt	27	3	-	
CodeGain	28	3	-	
CodeVirDta	29	1	-	
CodeT1	30	8	-	
CodeT2	31	8	-	

7 Disclaimer

Devices sold by Melexis are covered by the warranty and patent indemnification provisions appearing in its Term of Sale. Melexis makes no warranty, express, statutory, implied, or by description regarding the information set forth herein or regarding the freedom of the described devices from patent infringement. Melexis reserves the right to change specifications and prices at any time and without notice. Therefore, prior to designing this product into a system, it is necessary to check with Melexis for current information. This product is intended for use in normal commercial applications. Applications requiring extended temperature range, unusual environmental requirements, or high reliability applications, such as military, medical life-support or life-sustaining equipment are specifically not recommended without additional processing by Melexis for each application.
The information furnished by Melexis is believed to be correct and accurate. However, Melexis shall not be liable to recipient or any third party for any damages, including but not limited to personal injury, property damage, loss of profits, loss of use, interrupt of business or indirect, special incidental or consequential damages, of any kind, in connection with or arising out of the furnishing, performance or use of the technical data herein. No obligation or liability to recipient or any third party shall arise or flow out of Melexis' rendering of technical or other services.

website at:
www.melexis.com

Or for additional information contact Melexis Direct:

Europe and Japan:
Phone: +32 13 67 04 95
E-mail: sales_europe@melexis.com

All other locations:
Phone: +1 603 223 2362
E-mail: sales_usa@melexis.com

QS9000, VDA6.1 and ISO14001 Certified