MLX90632 PRODUCT SPECIFIC FUNCTIONS

SOFTWARE LIBRARY

1 Contents

2 Introduction

MLX90632 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 MLX90632 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

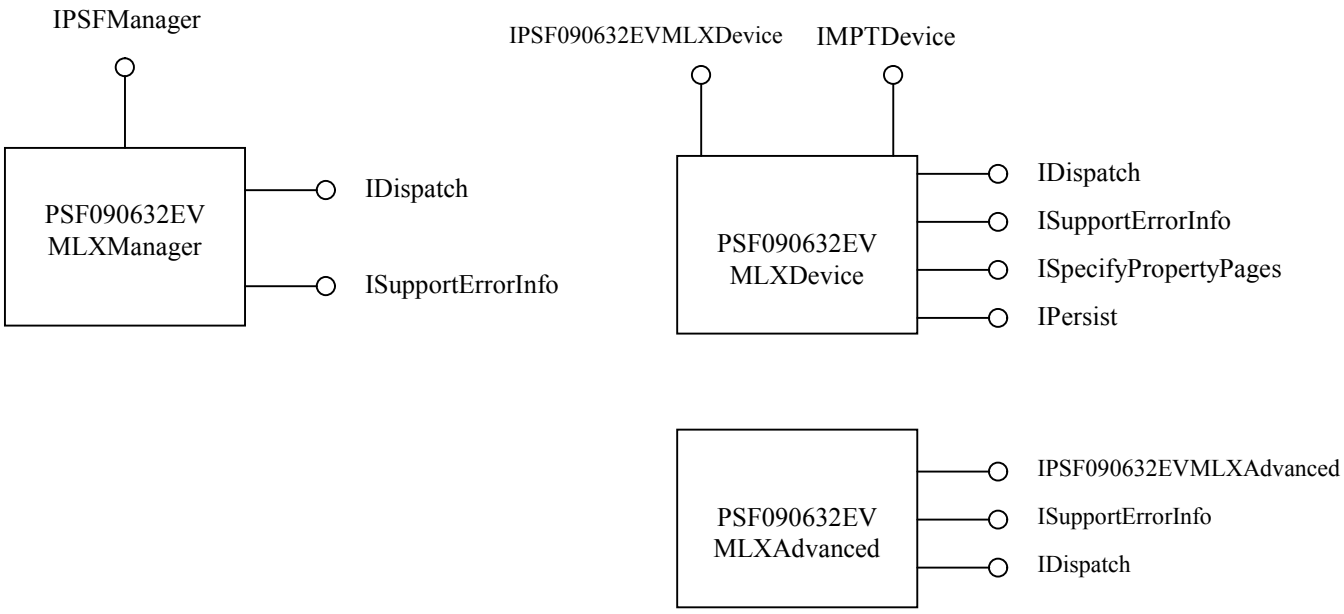
3 Software Structure

3.1 Object Model

MPT object model specifies that a PSF module must expose two COM objects which implement certain COM interfaces. MLX90632 PSF implements these two objects and two additional objects for advanced operations.

- **PSF090632EVMLXManager 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.
- **PSF090632EVMLXDevice object** – implements IPSF090632EVMLXDevice specific interface. However, this interface derives from IMPTDevice standard MPT interface and therefore PSF090632EVMLXDevice also implements the functionality of MPTDevice standard MPT object. In addition to standard IMPTDevice methods, IPSF090632EVMLXDevice 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.
- **PSF090632EVMLXAdvanced object** – implements IPSF090632EVMLXAdvanced library specific interface. This object implements advanced functions that would be rarely used in order to perform specific operations not available with the standard device functions. In general, most of the methods of that object provide direct access to MLX90632 EVB firmware commands.

3.2 Objects with Interfaces



4 PSF090632EVMLXManager 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.

PSF090632EVMLXManager 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_PSF090632EVMLXManager**, or ProgID **"MPT. PSF090632EVMLXManager"**:

```
hRes = ::CoCreateInstance(CLSID_PSF090632EVMLXManager, NULL, CLSCTX_INPROC,
                          IID_IPSFManager, (void**) &pPSFMan);
```

Visual Basic applications should call CreateObject function to instantiate PSF090632EVMLXManager:

```
Set PSFMan = CreateObject("MPT. PSF090632EVMLXManager")
```

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 PSF090632EVMLXDevice objects, one for each connected MLX90632 EVB. The collection is empty if there are no connected evaluation boards.

5 PSF090632EVMLXDevice Object

5.1 Background

This object implements standard MPT category **CATID_MLXMPTPSFUSBHIDDevice** as well as library specific **CATID_MLXMPT90632EVBDevice** category. It also declares required specific category **CATID_MLXMPT90632EVBUIModule** 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 PSF090632EVMLXManager object. The following Visual Basic subroutine shows how to instantiate PSF090632EVMLXDevice object by performing device scan on the system:

```
Sub CreateDevice()
   Dim PSFMan As PSF090632EVMLXManager, DevicesCol As ObjectCollection, I As Long
   On Error GoTo lError

   Set PSFMan = CreateObject("MPT.PSF090632EVMLXManager")
   Set DevicesCol = PSFMan.ScanStandalone(dtUSBHID
   If DevicesCol.Count <= 0 Then
      MsgBox ("No EVB90632 devices were found!")
      Exit Sub
   End If

   ' Dev is a global variable of type PSF090632EVMLXDevice
   ‘ 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 PSF090632EVMLXManager, 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.PSF090632EVMLXManager")
      Set DevicesCol = PSFMan.ScanStandalone(dtUSBHID)
      If DevicesCol.Count <= 0 Then
         MsgBox ("No EVB90632 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.PSF090632EVMLXDevice")
    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
```

PSF090632EVMLXDevice 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 PSF090632EVMLXDevice object implements IPSF090632EVMLXDevice library specific interface, which derives from IMPTDevice. The following is a description of its properties and methods.

5.2 Scope of the PSF090632EVMLXDevice object

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

5.3 Advanced Property

5.3.1 Description

This is a read-only property which returns a reference to **PSF090632EVMLXAdvanced** co-object.

5.3.2 Syntax

Visual Basic:

Property Advanced as PSF090632EVMLXAdvanced
Read only

C++:

HRESULT get_Advanced(/*[out][retval]*/ IPSF090632EVMLXAdvanced* pVal);

5.3.3 Parameters

pVal
Address of **IPSF090632EVMLXAdvanced*** pointer variable that receives the interface pointer to the Advanced object. If the invocation succeeds, the caller is responsible for calling **IUnknown::Release()** on the pointer when it is no longer needed.

5.3.4 Return value

Visual Basic:
A reference to the Advanced co-object.

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 pointer.
Any other error code	The operation failed. *pVal contains NULL.

5.4 ChipVersion Property

5.4.1 Description

This property specifies which the version of the device is connected to the board. Its value can be one of the constants defined in the **ChipVersionCodes** enumeration.

5.4.2 Syntax

Visual Basic:
Property ChipVersion as ChipVersionCodes

C++:
HRESULT get_ChipVersion(/*[out][retval]*/ ChipVersionCodes* pVal);
HRESULT set_ChipVersion(/*[in] */ ChipVersionCodes Val);

5.4.3 Parameters

pVal
Address of **ChipVersionCodes** variable that receives the currently selected device version.

Val
A **ChipVersionCodes** constant, specifying the required device version.

5.4.4 Return value

Visual Basic:
A **ChipVersionCodes** value corresponding to the currently selected device version.

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.

5.5 EVBGeneral Property

5.5.1 Description

This property holds a reference to GenericPSFDevice co-object.

5.5.2 Syntax

Visual Basic:

Property EVBGeneral as Object

C++:

HRESULT get_EVBGeneral(/*[out][retval]*/ LPDISPATCH* pVal);
HRESULT set_EVBGeneral(/*[in]*/ LPDISPATCH Value);

5.5.3 Parameters

Value

An **IDispatch*** specifying new EVBGeneral object. Nothing happens if the object is the same instance as the existing one. Otherwise PSF090632EVMLXDevice object releases its current EVBGeneral object and connects to the new one. This also includes replacing of the communication Channel object with the one from the new GenericPSFDevice object.

pVal

Address of **IDispatch*** pointer variable that receives the interface pointer to the EVBGeneral device object. If the invocation succeeds, the caller is responsible for calling **IUnknown::Release()** on the pointer when it is no longer needed.

5.5.4 Return value

Visual Basic:

A reference to the GenericPSFDevice co-object.

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 pointer.
Any other error code	The operation failed. *pVal contains NULL.

5.6 Emissivity Property

5.6.1 Description

This property specifies the emissivity parameter, used during temperature calculation.
The default value is 1.00.

5.6.2 Syntax

Visual Basic:

Property Emissivity as Single

C++:

HRESULT get_ Emissivity(/*[out][retval]*/ float* pVal);
HRESULT set_ Emissivity(/*[in] */float Val);

5.6.3 Parameters

pVal
Address of **float** variable that receives the current value of the emissivity.

Val
A **Single** value, specifying the new emissivity parameter.

5.6.4 Return value

Visual Basic:

A **Single** value of the current emissivity.

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.

5.7 MeasurementRange Property

5.7.1 Description

This property specifies which range to be used during temperature measurements.
The default value is mrtDefault (0).
Note, that some ICs only support the default measurement range. Trying to enable different range on such would return an error.

5.7.2 Syntax

Visual Basic:

Property MeasurementRange as MeasurementRangeType

C++:

HRESULT get_MeasurementRange (/*[out][retval]*/ MeasurementRangeType * pVal);
HRESULT set_MeasurementRange (/*[in] */ MeasurementRangeType Val);

5.7.3 Parameters

pVal
Address of **MeasurementRangeType** variable that receives the current value of the property.

Val
A **MeasurementRangeType** value, specifying the new measurement range.

5.7.4 Return value

Visual Basic:
A **MeasurementRangeType** value of the current emissivity.

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.

5.8 ReadFullEeprom Method

5.8.1 Description

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

5.8.2 Syntax

Visual Basic:
Sub ReadFullEeprom()

C++:
HRESULT ReadFullEeprom();

5.8.3 Parameters

None

5.8.4 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.

5.9 ProgramEeprom Method

5.9.1 Description

Programs the EEPROM of the device. Takes the values from the internal EEPROM cache.
Only the variables that are modified will be programmed.
Note that this method be called only if Advanced.EepromWritable property is True. Otherwise it will immediately return an error.

5.9.2 Syntax

Visual Basic:
Sub ProgramEeprom()

C++:
HRESULT ProgramEeprom();

5.9.3 Parameters

None

5.9.4 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.

5.10 DeviceReplaced Method

5.10.1 Description

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

5.10.2 Syntax

Visual Basic:
Sub DeviceReplaced()

C++:
HRESULT DeviceReplaced();

5.10.3 Parameters

None

5.10.4 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.

5.11 GetEEParameterCode Method

5.11.1 Description

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. **ReadFullEeprom** method could be called before calling **GetEEParameterCode** to update the whole cache. However **GetEEParameterCode** works correctly even if **ReadFullDevice** is not called.

5.11.2 Syntax

Visual Basic:

Function GetEEParameterCode(paramID as ParameterCodesEEPROM) as Long

C++:

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

5.11.3 Parameters

paramID

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

pVal

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

5.11.4 Return 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**.

5.12 SetEEParameterCode Method

5.12.1 Description

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

5.12.2 Syntax

Visual Basic:

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

C++:

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

5.12.3 Parameters

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

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

5.12.4 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.

5.13 GetEEParameterValue Method

5.13.1 Description

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.
Note, that currently no parameter has defined translation.

5.13.2 Syntax

Visual Basic:

```
Function GetEEParameterValue(paramID as ParameterCodesEEPROM)
```

C++:

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

5.13.3 Parameters

paramID
A ParameterCodesEEPROM 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.

5.13.4 Return 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**.

5.14 SetEEParameterValue Method

5.14.1 Description

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.
ProgramEeprom method must be called in order to update the EEPROM of the module with the codes from the cache.
Note, that currently no parameter has defined translation.

5.14.2 Syntax

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

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

5.14.3 Parameters

paramID

A ParameterCodesEEPROM constant specifying the ID of the EEPROM parameter.

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

5.14.4 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.

5.15 GetEEData Method

5.15.1 Description

This method returns the full contents (512 bytes) of EEPROM from the internal cache on the PC. In order to perform a real reading from the device, ReadFullEeprom method must be called first.

5.15.2 Syntax

Visual Basic:
```
Function GetEEData([Format As Long = 1]) as Variant
```

C++:
```
HRESULT GetEEData(/*[in,defaultvalue=1]*/ long Format
                  /*[out,retval]*/ VARIANT* ReadData);
```

5.15.3 Parameters

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

Value	Format
1	Data is an array of bytes. This is the preferred format for Visual Basic applications. This is the default value.
2	Data 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.
4	Data is an array of 16 bit integers.

ReadData
An address of **Variant** variable that will receive the read data. The type of content is specified by Format parameter. The caller is responsible to call VariantClear on that variable when it is no longer needed.

5.15.4 Return value

Visual Basic:

A **Variant**, containing the read data. The type of content is specified by 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.
Any other error code	The operation failed.

5.16 SetEEData Method

5.16.1 Description

This method sets the full contents (512 bytes) of EEPROM to the internal cache on the PC. In order to perform a real programming to the device, ProgramEeprom method must be called afterwards.

5.16.2 Syntax

Visual Basic:

Sub SetEEData(Data as Variant, [Format As Long = 1])

C++:

HRESULT SetEEData(/*[in]*/ VARIANT Data, /*[in,defaultvalue=1]*/ long Format);

5.16.3 Parameters

Data

A **Variant** containing 512 bytes which will be set in the cache. The type of content is specified by Format parameter.

Format

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

Value	Format
1	Data is an array of bytes. This is the preferred format for Visual Basic applications. This is the default value.
2	Data 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.

5.16.4 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.

5.17 SetVdd Method

5.17.1 Description

Sets supply voltage.

5.17.2 Syntax

Visual Basic:

Sub SetVdd(Volt As Single)

C++:

HRESULT SetVdd(/*[in]*/ float Volt);

5.17.3 Parameters

Volt

A **Single (float)** specifying supply voltage. Valid values are between 0 and 5V.

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 MeasureSupply Method

5.18.1 Description

This method will measure the supply voltage and current of the device.

5.18.2 Syntax

Visual Basic:

Function MeasureSupply(Byref Idd as Single) as Single

C++:

HRESULT MeasureSupply(/*[out]*/float* Idd, /*[out, retval]*/ float* Vdd);

5.18.3 Parameters

Idd

An address of **float** variable that will receive the measured supply current in (µA).

Vdd

An address of **float** variable that will receive the measured supply voltage in (V).

5.18.4 Return value

Visual Basic:
A **Single** containing the measured supply voltage.

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.

5.19 ContactTest Method

5.19.1 Description

This method checks if the device is properly connected. A valid I2C read command will be sent and checked for acknowledge. Then the same command will be sent to an invalid address and the result must be NAK.

5.19.2 Syntax

Visual Basic:
Function ContactTest() as Boolean

C++:
HRESULT ContactTest(/*[out, retval]*/ VARIANT_BOOL* pVal);

5.19.3 Parameters

pVal
An address of **VARIANT_BOOL** variable that will receive the result of the contact test.

5.19.4 Return value

Visual Basic:
A **Boolean** containing the result of the contact test.

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.

5.20 ReadMem Method

5.20.1 Description

This method reads a specified sequence of addresses from IC.

5.20.2 Syntax

Visual Basic:

Function ReadMem(Addr As Long, NWords As Long, [Format As Long = 1]) as Variant

C++:

HRESULT ReadMem(/*[in]*/ long Addr, /*[in]*/ long NWords,
/*[in]*/ long Format, /*[out,retval]*/ VARIANT* ReadData);

5.20.3 Parameters

Addr
A **Long** specifying the first RAM address to be read.

NWords
A **Long** specifying the number of words to be read.

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

Value	Format
1	Data is an array of bytes. This is the preferred format for Visual Basic applications. This is the default value.
2	Data 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.
4	Data is an array of 16 bit integers.

ReadData
An address of **Variant** variable that will receive the read data. The type of content is specified by Format parameter. The caller is responsible to call VariantClear on that variable when it is no longer needed.

5.20.4 Return value

Visual Basic:

A **Variant**, containing the read data. The type of content is specified by 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.
Any other error code	The operation failed.

5.21 WriteMem Method

5.21.1 Description

Writes to an address of IC.

5.21.2 Syntax

Visual Basic:

Sub WriteMem(Addr as Long, Data As Long)

C++:

HRESULT WriteMem(/*[in]*/ long Addr, /*[in]*/ long Data);

5.21.3 Parameters

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

Data
A **Long** specifying the data to be written.

5.21.4 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.

5.22 ReadSingleFrame Method

5.22.1 Description

This method reads object and PTAT temperatures from IC.

5.22.2 Syntax

Visual Basic:

Function ReadSingleFrame(Processing As DataProcessingTypes, ByRef PTAT As Long, ByRef IR2 As Long) as Long

C++:

HRESULT ReadSingleFrame(/*[in]*/ DataProcessingTypes Processing, /*[out]*/ long* PTAT, /*[out]*/ long* IR2, /*[out,retval]*/ long* IRData);

5.22.3 Parameters

Processing
A DataProcessingTypes constant, specifying whether raw or compensated data will be returned.

PTAT
An address of **Long** variable that will receive the value of PTAT register.

IR2
An address of **Long** variable that will receive the value of the second IR object temperature register.

IRData
An address of **Long** variable that will receive the value of IR object temperature register.

5.22.4 Return value

Visual Basic:
A **Long**, containing the value of IR object temperature register.

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 ReadSingleFrameEx Method

5.23.1 Description

This method reads object and PTAT temperatures from IC. Additionally it provides the raw data used during calculations.

5.23.2 Syntax

Visual Basic:

```
Function ReadSingleFrameEx(Processing As DataProcessingTypes,
                Format As Long, ByRef RawData as Variant,
                ByRef PTAT As Long, ByRef IR2 As Long) as Long
```

C++:

```
HRESULT ReadSingleFrame(/*[in]*/ DataProcessingTypes Processing,
                /*[in]*/long Format, /*[out]*/ VARIANT* RawData,  /*[out]*/long* PTAT,
                /*[out]*/ long* IR2, /*[out,retval]*/ long* IRData);
```

5.23.3 Parameters

Processing
A DataProcessingTypes constant, specifying whether raw or compensated data will be returned.

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

Value	Format
1	Data is an array of bytes. This is the preferred format for Visual Basic applications. This is the default value.
2	Data 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.
4	Data is an array of 16 bit integers.

RawData

An address of **Variant** variable that will receive the raw data. The type of content is specified by Format parameter. The caller is responsible to call VariantClear on that variable when it is no longer needed.

PTAT
An address of **Long** variable that will receive the value of PTAT register.

IR2
An address of **Long** variable that will receive the value of the second IR object temperature register.

IRData
An address of **Long** variable that will receive the value of IR object temperature register.

5.23.4 Return value

Visual Basic:
A **Long**, containing the value of IR object temperature register.

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.

5.24 CmdReset Method

5.24.1 Description

This method sends a RESET command to the device.

5.24.2 Syntax

Visual Basic:
Sub CmdReset()

C++:
HRESULT CmdReset();

5.24.3 Parameters

None

5.24.4 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.

5.25 ReadChipVersion Method

5.25.1 Description

This method is used to determine the type of the connected device.

5.25.2 Syntax

Visual Basic:

Function ReadChipVersion() as ChipVersionCodes

C++:

HRESULT ReadChipVersion(/*[out,retval]*/ ChipVersionCodes* pValue);

5.25.3 Parameters

pValue

An address of **ChipVersionCodes** variable that will receive an enumeration code, corresponding to the type of the connected device. If there is no connected device or communication with such cannot be established, the returned code is ChipVersionUndefined.

5.25.4 Return value

Visual Basic:

A **ChipVersionCodes** code, corresponding to the type of the connected device. If there is no connected device or communication with such cannot be established, the returned code is ChipVersionUndefined.

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.

6 PSF090632EVMLXAdvanced Object

6.1 Background

This object cannot be created directly; it is only accessible as "Advanced" property of PSF090632EVMLXDevice object.

PSF090632EVMLXAdvanced object implements IPSF090632EVMLXAdvanced library specific interface. The following is a description of its methods.

6.2 Scope of the PSF090632EVMLXAdvanced object

This object implements advanced functions that would be rarely used in order to perform specific operations not available with the standard device functions. In general, most of the methods of that object provide direct access to MLX90632 EVB firmware commands.

6.3 Logging Property

6.3.1 Description

Specifies whether logging information is generated while working with the library, mostly for the solving process.

6.3.2 Syntax

Visual Basic:

Property Logging as Boolean

C++:

HRESULT get_Logging(/*[out,retval]*/ VARIANT_BOOL* pValue);
HRESULT set_Logging(/*[in]*/ VARIANT_BOOL Value);

6.3.3 Parameters

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.

6.3.4 Return 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.

6.4 QuietCheck Property

6.4.1 Description

Specifies whether connection and configuration check, performed in front of each high level method, can show warning and confirmation messages or will directly return an error message.

6.4.2 Syntax

Visual Basic:

Property QuietCheck as Boolean

C++:

HRESULT get_QuietCheck(/*[out,retval]*/ VARIANT_BOOL* pValue);
HRESULT set_QuietCheck(/*[in]*/ VARIANT_BOOL Value);

6.4.3 Parameters

pValue
An address of **VARIANT_BOOL** variable that receives current value of the property. **VARIANT_TRUE** means that checks are "quiet", **VARIANT_FALSE** means that warnings can be shown.

Value
A **VARIANT_BOOL** specifying new value for the property. **VARIANT_TRUE** suppress dialogs, **VARIANT_FALSE** allows them.

6.4.4 Return value

Visual Basic:

True if checks are "quiet", **False** if warnings can be shown.

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.

6.5 EepromWritable Property

6.5.1 Description

This property specifies whether EEPROM of the device can be written. By default its value is **False**, meaning EEPROM cannot be written.

6.5.2 Syntax

Visual Basic:

Property EepromWritable as Boolean

C++:

HRESULT get_ EepromWritable(/*[out,retval]*/ VARIANT_BOOL* pValue);
HRESULT set_ EepromWritable(/*[in]*/ VARIANT_BOOL Value);

6.5.3 Parameters

pValue
An address of **VARIANT_BOOL** variable that receives current value of the property. **VARIANT_TRUE** means that calls to ProgramEeprom method will write to the device, **VARIANT_FALSE** means that such calls will return an error.

Value
A **VARIANT_BOOL** specifying new value for the property. **VARIANT_TRUE** enables writing, **VARIANT_FALSE** disables it.

6.5.4 Return value

Visual Basic:

True if ProgramEeprom method will write to the device, **False** if it will return an error.

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.

6.6 GetSetting Method

6.6.1 Description

Returns the value of a particular setting.

6.6.2 Syntax

Visual Basic:

Function GetSetting(settingID as SettingCodes)

C++:

HRESULT GetSetting(/*[in]*/ SettingCodes settingID, /*[out,retval]*/ TVariant* pVal);

6.6.3 Parameters

settingID
A SettingCodes constant specifying the ID of the setting.

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.

6.6.4 Return value

Visual Basic:
A **Variant** containing the value of a setting.

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 is **Empty**.

6.7 SetSetting Method

6.7.1 Description

Changes the value of a particular setting. Sets an associated internal variable. The setting is also sent immediately to MLX90632 evaluation board.

NOTE: If necessary, the changes can be saved in the profile with a subsequent call to SaveProfile or SaveProfileAs methods.

6.7.2 Syntax

Visual Basic:
Sub SetSetting(settingID as SettingCodes, Value)

C++:
HRESULT SetSetting(/*[in]*/ SettingCodes settingID, /*[in]*/ TVariantInParam Value);

6.7.3 Parameters

settingID
A SettingCodes constant specifying the ID of the setting to modify.

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

6.7.4 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.

6.8 OpenProfile Method

6.8.1 Description

Opens the specified file and updates the settings.

6.8.2 Syntax

Visual Basic:

Sub OpenProfile(FileName as String)

C++:

HRESULT OpenProfile(/*[in]*/ BSTR FileName);

6.8.3 Parameters

FileName
A **String** specifying the path of the file to open.

6.8.4 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.

6.9 SaveProfile Method

6.9.1 Description

Saves the settings into a previously opened profile. This function fails if there is not a profile in use.

6.9.2 Syntax

Visual Basic:

Sub SaveProfile()

C++:

HRESULT SaveProfile();

6.9.3 Parameters

None

6.9.4 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.

6.10 SaveProfileAs Method

6.10.1 Description

Saves the settings into the specified file.

6.10.2 Syntax

Visual Basic:

Sub SaveProfileAs(FileName as String)

C++:

HRESULT SaveProfileAs(/*[in]*/ BSTR FileName);

6.10.3 Parameters

FileName
A **String** specifying the path of the file.

6.10.4 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.

6.11 I2CWriteRead Method

6.11.1 Description

This method is for general I2C master-to-slave communication. It could be used for write or read transmissions, and also for write then read. Start and Stop conditions are generated respectively at the beginning and the end of the transmission. A Repeated Start Condition is inserted between Write-then-Read requests.

6.11.2 Syntax

Visual Basic:

Function I2CWriteRead(DevAddr As Byte, WriteData as Variant, Format As Long, NReadBytes As Long, err As Byte) as Variant

C++:

HRESULT I2CWriteRead(/*[in]*/ unsigned char DevAddr, /*[in]*/ VARIANT WriteData, /*[in]*/ long Format, /*[in]*/ long NReadBytes, /*[out]*/ unsigned char* err, /*[out,retval]*/ VARIANT* ReadData);

6.11.3 Parameters

DevAddr
A **Byte** specifying the address of the slave device.

WriteData
A **Variant** specifying the data to be send by the master. The content of the variant depends on the Format parameter.

Format
A **long** specifying the format of data in WriteData and ReadData. Possible values are:

Value	Format
1	Data is an array of bytes. This is the preferred format for Visual Basic applications. This is the default value.
2	Data 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.
4	Data is an array of 16 bit integers.

NReadBytes
A **Long** specifying the number of bytes to read after (eventual) writing.

err
An address of **Byte** variable that will receive an error code (0=no error, 1=I2C NAK).

ReadData
An address of **Variant** variable that will receive the read data. The type of content is specified by Format parameter. The caller is responsible to call VariantClear on that variable when it is no longer needed.

6.11.4 Return value

Visual Basic:

A **Variant**, containing the read data. The type of content is specified by 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.
Any other error code	The operation failed.

7 Enumeration constants

7.1 ParameterCodesEEPROM 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
CodeCONTROL	1	16	-	
CodeI2C_ADDRESS	2	16	-	
CodeWAIT_TIME0	3	16	-	
CodeWAIT_TIME1	4	16	-	
CodeWAIT_TIME2	5	16	-	
CodeWAIT_TIME3	6	16	-	
CodeRES_CTRL_CH1_0	7	16	-	
CodeRES_CTRL_CH1_1	8	16	-	
CodeRES_CTRL_CH2_0	9	16	-	
CodeRES_CTRL_CH2_1	10	16	-	
CodeRES_CTRL_CH3_0	11	16	-	
CodeRES_CTRL_CH3_1	12	16	-	
CodeMEAS0	13	16	-	
CodeMEAS1	14	16	-	
CodeMEAS2	15	16	-	
CodeMEAS3	16	16	-	
CodeMEAS4	17	16	-	
CodeMEAS5	18	16	-	
CodeMEAS6	19	16	-	
CodeMEAS7	20	16	-	
CodeMEAS8	21	16	-	
CodeMEAS9	22	16	-	
CodeMEAS10	23	16	-	
CodeMEAS11	24	16	-	
CodeMEAS12	25	16	-	
CodeMEAS13	26	16	-	
CodeMEAS14	27	16	-	
CodeMEAS15	28	16	-	
CodeMEAS16	29	16	-	
CodeMEAS17	30	16	-	
CodeMEAS18	31	16	-	
CodeMEAS19	32	16	-	
CodeMEAS20	33	16	-	
CodeMEAS21	34	16	-	
CodeMEAS22	35	16	-	
CodeMEAS23	36	16	-	

Constant	Value	Bits	Translation value	Description
CodeMEAS24	37	16	-	
CodeMEAS25	38	16	-	
CodeMEAS26	39	16	-	
CodeMEAS27	40	16	-	
CodeMEAS28	41	16	-	
CodeMEAS29	42	16	-	
CodeMEAS30	43	16	-	
CodeMEAS31	44	16	-	
CodeCUST_CRC16	45	16	-	
CodeAlpha_appli	46	16	-	
CodeTO_Offset_appli	47	16	-	
CodeTRIM0	501	16	-	R/O
CodeTRIM1	502	16	-	R/O
CodeTRIM2	503	16	-	R/O
CodeTRIM3	504	16	-	R/O
CodeI2C_CONFIG	505	16	-	R/O
CodeID0	506	16	-	R/O
CodeID1	507	16	-	R/O
CodeID2	508	16	-	R/O
CodeIDCRC	509	16	-	R/O
CodeMLX_CRC16	510	16	-	R/O
CodeVERSIONS	511	16	-	R/O
CodeDSP_VERSION	512	8	-	R/O
CodeTRIM_VERSION	513	8	-	R/O
CodePTAT_REF	514	32	-	R/O
CodePTAT_GAIN	515	32	-	R/O
CodePTAT_TC2	516	32	-	R/O
CodePTAT_OFFSET	517	32	-	R/O
CodeAa	518	32	-	R/O
CodeAb	519	32	-	R/O
CodeBa	520	32	-	R/O
CodeBb	521	32	-	R/O
CodeCa	522	32	-	R/O
CodeCb	523	32	-	R/O
CodeDa	524	32	-	R/O
CodeDb	525	32	-	R/O
CodeK_TA	526	32	-	R/O
CodeTA0	527	32	-	R/O
CodeAlpha	528	32	-	R/O
CodeKsTA	529	32	-	R/O
CodeKsTO	530	32	-	R/O
CodeBeta_TA	531	16	-	R/O
CodeBeta_IR	532	16	-	R/O
CodeVDDMON_OFFSET	533	16	-	R/O
CodePRODUCT_CODE	534	16	-	R/O

Constant	Value	Bits	Translation value	Description
CodeACCURACY_RANGE	535	5	-	R/O
CodePACKAGE	536	3	-	R/O
CodeFDV	537	2	-	R/O

7.2 SettingCodes enumeration

The following constants specify different settings. They are used by GetSetting and SetSetting methods.

Constant	Value	Type	Default value	Description
SettingTpor	1	(Long) long	10000 [µs]	Power On Reset delay
SettingTreset	2		1000 [µs]	Delay to keep Vdd off for resetting
SettingTclock	3	(Single) float	1.0 [µs]	I2C clock speed
SettingTstart	4	(Single) float	1.0 [µs]	I2C Start condition delay
SettingTstop	5	(Single) float	1.0 [µs]	I2C Stop condition delay
SettingTwrdelay	6	(Single) float	1.0 [µs]	Delay between the write and read I2C commands
SettingI2CaddrIR	7	(Byte) unsigned char	58	The address of IR device
SettingTEEwrite	8	(Long) long	6000 [µs]	Delay for writing to an EEPROM address
SettingTholdData	9	(Single) float	0.05 [µs]	I2C delay between SCL and SDA edges

7.3 ChipVersionCodes enumeration

The following constants specify different versions of the device. They are used by ChipVersion property.

Constant	Value	Description
ChipVersionUndefined	0	
ChipVersion90632AAA	1	

7.4 DataProcessingTypes enumeration

The following constants specify different types of data processing. They are used by ReadSingleFrame method.

Constant	Value	Description

Constant	Value	Description
dptRawData	0	Received data will be the same as read from IC
dptAbsoluteToFromPC	2	Received data will be absolute To. Conversion will be made on PC.

7.5 MeasurementRangeTypes enumeration

The following constants specify different types of measurement range. They are used by **MeasurementRange** property.

Constant	Value	Description
mrtDefault	0	Use the default measurement range of the connected IC
mrtExtended	1	Use extended measurement range of the connected IC (Optional)

8 History records

Version	Date	Comments
1.9.0	Mar 31, 2020	Added new property MeasurementRange
1.8.3	Aug 2, 2019	Added PRODUCT_CODE, ACCURACY_RANGE, PACKAGE and FDV parameters
1.7.0	Feb 7, 2019	Added new method ReadSingleFrameEx
1.6.0	May 15, 2018	Added Emissivity (default 1.0) property
1.5.0	May 16, 2017	Initial release

9 Disclaimer

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