EVB - PSF - MLX90640

Product Specific Function description Software Library

Contents

1 Introduction

MLX90640 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 MLX90640 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

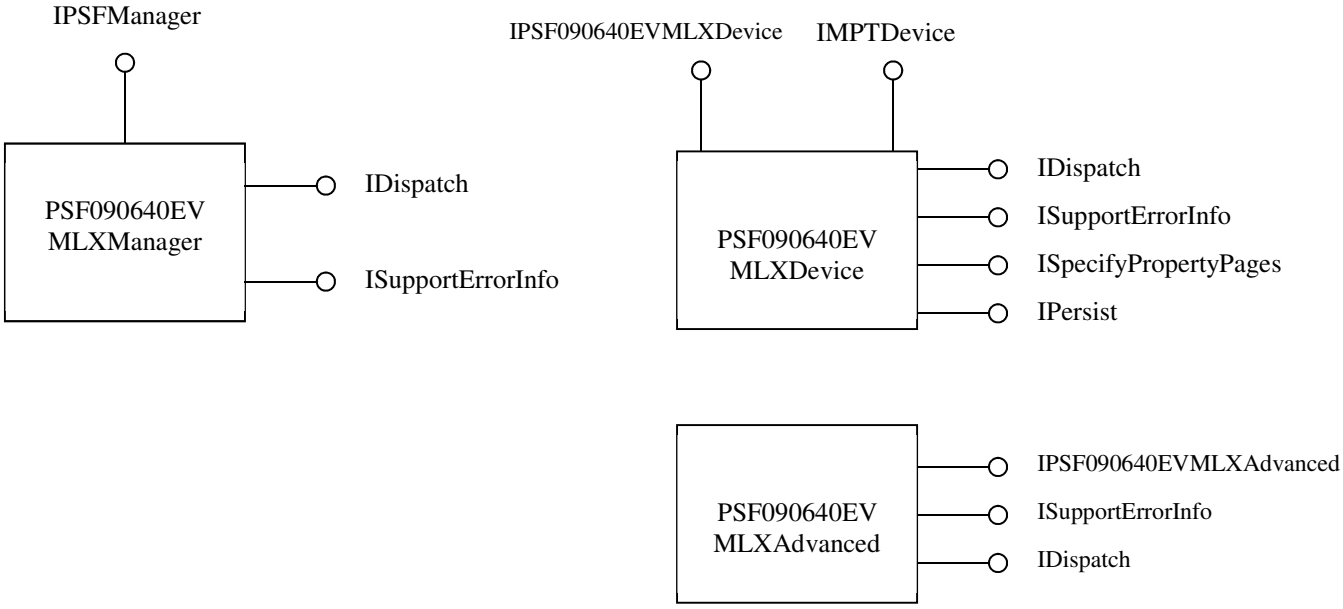
2 Software Structure

2.1 Object Model

MPT object model specifies that a PSF module must expose two COM objects which implement certain COM interfaces. MLX90640 PSF implements these two objects and two additional objects for advanced operations.

- **PSF090640EVMLXManager 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.
- **PSF090640EVMLXDevice object** – implements IPSF090640EVMLXDevice specific interface. However, this interface derives from IMPTDevice standard MPT interface and therefore PSF090640EVMLXDevice also implements the functionality of MPTDevice standard MPT object. In addition to standard IMPTDevice methods, IPSF090640EVMLXDevice 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.
- **PSF090640EVMLXAdvanced object** – implements IPSF090640EVMLXAdvanced 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 MLX90640 EVB firmware commands.

2.2 Objects with Interfaces



3 PSF090640EVMLXManager Object

3.1 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.

PSF090640EVMLXManager 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_PSF090640EVMLXManager, or ProgID "MPT. PSF090640EVMLXManager":

hRes = ::CoCreateInstance(CLSID_PSF090640EVMLXManager, NULL, CLSCTX_INPROC, IID_IPSFManager, (void) &pPSFMan);**

Visual Basic applications should call CreateObject function to instantiate PSF090640EVMLXManager:

Set PSFMan = CreateObject("MPT. PSF090640EVMLXManager")

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 PSF090640EVMLXDevice objects, one for each connected MLX90640 EVB. The collection is empty if there are no connected evaluation boards.

4 PSF090640EVMLXDevice Object

4.1 Background

This object implements standard MPT category **CATID_MLXMPTPSFUSBHIDDevice** as well as library specific **CATID_MLXMPT90640EVBDevice** category. It also declares required specific category **CATID_MLXMPT90640EVBUIModule** 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 PSF090640EVMLXManager object. The following Visual Basic subroutine shows how to instantiate PSF090640EVMLXDevice object by performing device scan on the system:

```
Sub CreateDevice()
   Dim PSFMan As PSF090640EVMLXManager, DevicesCol As ObjectCollection, I As Long
   On Error GoTo lError

   Set PSFMan = CreateObject("MPT.PSF090640EVMLXManager")
   Set DevicesCol = PSFMan.ScanStandalone(dtUSBHID
   If DevicesCol.Count <= 0 Then
      MsgBox ("No EVB90640 devices were found!")
      Exit Sub
   End If

   ' Dev is a global variable of type PSF090640EVMLXDevice
   ‘ 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 PSF090640EVMLXManager, 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.PSF090640EVMLXManager")
      Set DevicesCol = PSFMan.ScanStandalone(dtUSBHID)
      If DevicesCol.Count <= 0 Then
         MsgBox ("No EVB90640 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.PSF090640EVMLXDevice")
```

```
    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
```

PSF090640EVMLXDevice 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 PSF090640EVMLXDevice object implements IPSF090640EVMLXDevice library specific interface, which derives from IMPTDevice. The following is a description of its properties and methods.

4.2 Scope of the PSF090640EVMLXDevice object

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

4.3 Advanced Property

4.3.1 Description

This is a read-only property which returns a reference to **PSF090640EVMLXAdvanced** co-object.

4.3.2 Syntax

Visual Basic:

Property Advanced as PSF090640EVMLXAdvanced
Read only

C++:

HRESULT get_Advanced(/*[out][retval]*/ IPSF090640EVMLXAdvanced* pVal);

4.3.3 Parameters

pVal
Address of **IPSF090640EVMLXAdvanced*** 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.

4.3.4 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.

4.4 ChipVersion Property

4.4.1 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.

4.4.2 Syntax

Visual Basic:

Property ChipVersion as ChipVersionCodes

C++:

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

4.4.3 Parameters

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

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

4.4.4 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.

4.5 EVBGeneral Property

4.5.1 Description

This property holds a reference to GenericPSFDevice co-object.

4.5.2 Syntax

Visual Basic:

Property EVBGeneral as Object

C++:

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

4.5.3 Parameters

Value
An **IDispatch*** specifying new EVBGeneral object. Nothing happens if the object is the same instance as the existing one. Otherwise PSF090640EVMLXDevice 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.

4.5.4 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.

4.6 Emissivity Property

4.6.1 Description

Specifies Emissivity parameter to be used during object temperature calculation.
Default value is 1.0.

4.6.2 Syntax

Visual Basic:

Property Emissivity as Single

C++:

HRESULT get_Emissivity (/*[out,retval]*/ float* pValue);
HRESULT set_Emissivity(/*[in]*/ float Value);

4.6.3 Parameters

pValue
An address of **float** variable that receives current value of the property.

Value
A **float** specifying new value for the property.

4.6.4 Return value

Visual Basic:
Emissivity parameter used during object temperature calculation.

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.

4.7 UseEmissivityFromEeprom Property

4.7.1 Description

Specifies whether Emissivity property will be overwritten by CodeEmissivity parameter when ReadFullEeprom is called. This option is applicable only for MLX90641.
Default value is False.

4.7.2 Syntax

Visual Basic:
Property UseEmissivityFromEeprom as Boolean

C++:
HRESULT get_UseEmissivityFromEeprom(/*[out,retval]*/ VARIANT_BOOL* pValue);
HRESULT set_UseEmissivityFromEeprom(/*[in]*/ VARIANT_BOOL Value);

4.7.3 Parameters

pValue
An address of **VARIANT_BOOL** variable that receives current value of the property.

Value
A **VARIANT_BOOL** specifying new value for the property. **VARIANT_TRUE** activates the Emissivity update during EEPROM reading, **VARIANT_FALSE** deactivates it.

4.7.4 Return value

Visual Basic:
True if Emissivity will be updated during EEPROM reading or **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.

4.8 ReadFullEeprom Method

4.8.1 Description

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

4.8.2 Syntax

Visual Basic:

Sub ReadFullEeprom()

C++:

HRESULT ReadFullEeprom();

4.8.3 Parameters

None

4.8.4 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.

4.9 ProgramEeprom Method

4.9.1 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.

4.9.2 Syntax

Visual Basic:

Sub ProgramEeprom()

C++:

HRESULT ProgramEeprom();

4.9.3 Parameters

None

4.9.4 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.

4.10 DeviceReplaced Method

4.10.1 Description

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

4.10.2 Syntax

Visual Basic:

```
Sub DeviceReplaced()
```

C++:

```
HRESULT DeviceReplaced();
```

4.10.3 Parameters

None

4.10.4 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.

4.11 GetEEParameterCode Method

4.11.1 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 must be called before calling **GetEEParameterCode** to update the whole cache.

4.11.2 Syntax

Visual Basic:

```
Function GetEEParameterCode(paramID as ParameterCodesEEPROM) as Long
```

C++:

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

4.11.3 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.

4.11.4 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**.

4.12 SetEEParameterCode Method

4.12.1 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.

4.12.2 Syntax

Visual Basic:
Sub SetEEParameterCode(paramID as ParameterCodesEEPROM, Value as Long)

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

4.12.3 Parameters

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

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

4.12.4 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.

4.13 GetEEIdxParameterCode Method

4.13.1 Description

Returns the code of a particular **indexed** EEPROM parameter as it is represented in EEPROM. It is optimized because it uses the EEPROM cache maintained by the library. **ReadFullEeprom** method must be called before calling **GetEEIdxParameterCode** to update the whole cache.
Note that this method is applicable only to parameters with defined index range in **ParameterCodesEEPROM** table.

4.13.2 Syntax

Visual Basic:

Function GetEEIdxParameterCode(paramID as ParameterCodesEEPROM, Index as Long) as Long

C++:

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

4.13.3 Parameters

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

Index
A **Long** number, specifying the requested index.

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

4.13.4 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**.

4.14 SetEEIdxParameterCode Method

4.14.1 Description

Changes the code of a particular indexed 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.
Note that this method is applicable only to parameters with defined index range in **ParameterCodesEEPROM** table.

4.14.2 Syntax

Visual Basic:

```
Sub SetEEIdxParameterCode(paramID as ParameterCodesEEPROM, Index as Long,
        Value as Long)
```

C++:

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

4.14.3 Parameters

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

Index
A **Long** number, specifying the requested index.

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

4.14.4 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.

4.15 GetEEParameterValue Method

4.15.1 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.

4.15.2 Syntax

Visual Basic:

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

C++:

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

4.15.3 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.

4.15.4 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**.

4.16 SetEEParameterValue Method

4.16.1 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.

4.16.2 Syntax

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

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

4.16.3 Parameters

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

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

4.16.4 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.

4.17 GetEE1Data Method

4.17.1 Description

This method returns the full contents (832*2 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.

4.17.2 Syntax

Visual Basic:

```
Function GetEE1Data([Format As Long = 1]) as Variant
```

C++:

```
HRESULT GetEE1Data(/*[in,defaultvalue=1]*/ long Format
                   /*[out,retval]*/ VARIANT* ReadData);
```

4.17.3 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.

4.17.4 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.

4.18 SetEE1Data Method

4.18.1 Description

This method sets the full contents (832*2 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.

4.18.2 Syntax

Visual Basic:

Sub SetEE1Data(Data as Variant, [Format As Long = 1])

C++:

HRESULT SetEE1Data(/*[in]*/ VARIANT Data, /*[in,defaultvalue=1]*/ long Format);

4.18.3 Parameters

Data

A **Variant** containing 832*2 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 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.

4.18.4 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.

4.19 SetVdd Method

4.19.1 Description

Sets supply voltage.

4.19.2 Syntax

Visual Basic:

Sub SetVdd(Volt As Single)

C++:

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

4.19.3 Parameters

Volt

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

4.19.4 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.

4.20 MeasureVdd Method

4.20.1 Description

This method will measure the supply voltage of the device.

4.20.2 Syntax

Visual Basic:

Function MeasureVdd() as Single

C++:

HRESULT MeasureVdd(/*[out, retval]*/ float* pVal);

4.20.3 Parameters

pVal

An address of **float** variable that will receive the measured supply voltage.

4.20.4 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.

4.21 ContactTest Method

4.21.1 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.

4.21.2 Syntax

Visual Basic:

Function ContactTest() as Boolean

C++:

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

4.21.3 Parameters

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

4.21.4 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.

4.22 ReadMem Method

4.22.1 Description

This method reads the specified memory area from IC.

4.22.2 Syntax

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

C++:
HRESULT ReadRam(/*[in]*/ long Addr, /*[in]*/ long NWords,
/*[in]*/ long Format, /*[out,retval]*/ VARIANT* ReadData);

4.22.3 Parameters

Addr
A **Long** specifying the first 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.

4.22.4 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.

4.23 WriteMem Method

4.23.1 Description

Writes a word in memory of IC.

4.23.2 Syntax

Visual Basic:

Sub WriteMem(Addr as Long, Data As Long)

C++:

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

4.23.3 Parameters

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

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

4.23.4 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.

4.24 ReadSingleFrame Method

4.24.1 Description

This method reads a frame with the whole IR array as well as PTAT and Cyclop sensors from IC.

4.24.2 Syntax

Visual Basic:

Function ReadSingleFrame(Processing As DataProcessingTypes, Format As Long, ByRef PTAT As Long, ByRef Cyclop As Long) as Variant

C++:

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

4.24.3 Parameters

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

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.

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

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

IRData
An address of **Variant** variable that will receive the content of IR sensors array. 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.

4.24.4 Return value

Visual Basic:

A **Variant**, containing IR sensors array. 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.

4.25 SendStart Method

4.25.1 Description

This method sends Start Conversion command to the device.

4.25.2 Syntax

Visual Basic:

Sub SendStart()

C++:

HRESULT SendStart();

4.25.3 Parameters

None

4.25.4 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.

4.26 StartDAQ Method

4.26.1 Description

This method initiates continuous data acquisition on the evaluation board. After starting, ReadDAQ must be called regularly in order to receive data. At the end, StopDAQ must be called to terminate acquisition process on EVB.
Note, that this method **doesn't send Start command**, but just reads whatever is in ICs RAM. Thus if the device is configured in Step mode, successive reads will give the same result.

4.26.2 Syntax

Visual Basic:

Sub StartDAQ(FramePeriodUS As Long, Processing As DataProcessingTypes)

C++:

HRESULT StartDAQ(/*[in]*/ long FramePeriodUS, /*[in]*/ DataProcessingTypes Processing);

4.26.3 Parameters

FramePeriodUS
A **long** specifying the delay between successive frames reads in [us]. It must be bigger than 0.

Processing
A DataProcessingTypes constant, specifying the type of data acquisition to start. The meaningful modes are listed below:

Constant	**DAQ type**
1237	MLX90640 32x24 pixels frame measured in continuous mode.
1238	MLX90641 16x12 pixels frame measured in continuous mode.

4.26.4 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.

4.27 ReadDAQ Method

4.27.1 Description

This method reads all the buffered data on the EVB. If no data has been buffered yet, it'll return an empty Variant.

4.27.2 Syntax

Visual Basic:

Function ReadDAQ(ByVal err As Byte, [Format As Long = 1]) as Variant

C++:

```
HRESULT ReadDAQ(/*[out]*/ unsigned char* err, /*[in,defaultvalue=1]*/ long Format,
        /*[out,retval]*/ VARIANT* pData);
```

4.27.3 Parameters

err

An address of **Byte** variable that will receive an error code (0=no error, 1=I2C NAK, 2=EVB buffer overflow)

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.

pData

An address of **Variant** variable that will receive the buffered 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.

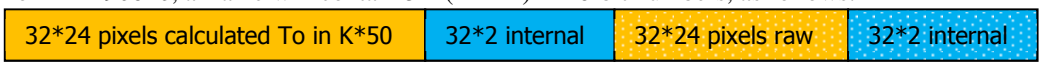
4.27.4 Return value

Visual Basic:

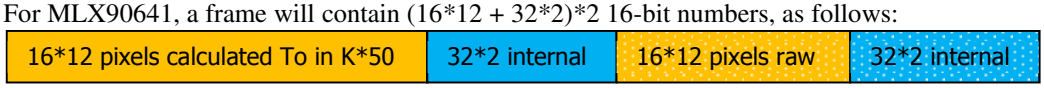
A **Variant**, containing the buffered data. If it's not empty, the result will be an array of Variants. Each of the later will be the data of an IR frame.

The type of the frame variant is specified by Format parameter. Each pixel is represented by a 16 bit integer number. The frame contains pixel data twice – first time with calculated temperatures in Kelvin * 50 and second time the raw data.

For MLX90640, a frame will contain 32*(24 + 2)*2 16-bit numbers, as follows:



For MLX90641, a frame will contain (16*12 + 32*2)*2 16-bit numbers, as follows:



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.

4.28 StopDAQ Method

4.28.1 Description

This method will terminate data acquisition process running on EVB. If DAQ hasn't been started, there will be no effect.

4.28.2 Syntax

Visual Basic:

Sub StopDAQ()

C++:

HRESULT StopDAQ();

4.28.3 Parameters

None

4.28.4 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.

4.29 FilterInterlacing Method

4.29.1 Description

This method specifies if de-interlacing filter will be applied during data acquisition (DAQ).
By default, de-interlacing filter is disabled.
Note that this filter is applicable only for MLX90640.

4.29.2 Syntax

Visual Basic:

Sub FilterInterlacing(Enable As Boolean)

C++:

HRESULT FilterInterlacing(/*[in]*/ VARIANT_BOOL Enable);

4.29.3 Parameters

Enable
A **Boolean**, specifying if de-interlacing filter will be applied during data acquisition of MLX90640.

4.29.4 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.

4.30 FilterThresholdAveraging Method

4.30.1 Description

This method specifies if averaging filter will be applied during data acquisition (DAQ).
By default, averaging filter is disabled.

4.30.2 Syntax

Visual Basic:

Sub FilterThresholdAveraging(Enable As Boolean, Depth as Long, Threshold as Single)

C++:

HRESULT FilterThresholdAveraging(/*[in]*/ VARIANT_BOOL Enable, /*[in]*/ long Depth, /*[in]*/ float Threshold);

4.30.3 Parameters

Enable
A **Boolean**, specifying if averaging filter will be applied during data acquisition.

Depth
A **Long**, specifying the depth of the filter.

Threshold
A **Single**, temperature difference (in degC) which if met would reset the filter to the current temperature.

4.30.4 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.

4.31 FilterTgc Method

4.31.1 Description

This method specifies if TGC filter will be applied during data acquisition (DAQ).
By default, TGC filter is disabled.

4.31.2 Syntax

Visual Basic:

Sub FilterTgc(Enable As Boolean, Depth as Long)

C++:

HRESULT FilterTgc(/*[in]*/ VARIANT_BOOL Enable, /*[in]*/ long Depth);

4.31.3 Parameters

Enable
A **Boolean**, specifying if TGC filter will be applied during data acquisition.

Depth
A **Long**, specifying the depth of the filter.

4.31.4 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.

4.32 UploadFirmwareFromDFU Method

4.32.1 Description

This method writes new firmware on the evaluation board.

4.32.2 Syntax

Visual Basic:

Sub UploadFirmwareFromDFU(DfuFilename As String)

C++:

HRESULT UploadFirmwareFromDFU(/*[in]*/ BSTR DfuFilename);

4.32.3 Parameters

DfuFilename
A **String**, specifying the full path to a DFU file, containing the firmware to be written.

4.32.4 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 PSF090640EVMLXAdvanced Object

5.1 Background

This object cannot be created directly; it is only accessible as "Advanced" property of PSF090640EVMLXDevice object.

PSF090640EVMLXAdvanced object implements IPSF090640EVMLXAdvanced library specific interface. The following is a description of its methods.

5.2 Scope of the PSF090640EVMLXAdvanced 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 MLX90640 EVB firmware commands.

5.3 Logging Property

5.3.1 Description

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

5.3.2 Syntax

Visual Basic:

Property Logging as Boolean

C++:

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

5.3.3 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.

5.3.4 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.

5.4 QuietCheck Property

5.4.1 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.

5.4.2 Syntax

Visual Basic:

Property QuietCheck as Boolean

C++:

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

5.4.3 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.

5.4.4 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.

5.5 EepromWritable Property

5.5.1 Description

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

5.5.2 Syntax

Visual Basic:

Property EepromWritable as Boolean

C++:

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

HRESULT set_ EepromWritable(/*[in]*/ VARIANT_BOOL Value);

5.5.3 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.

5.5.4 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.

5.6 GetSetting Method

5.6.1 Description

Returns the value of a particular setting.

5.6.2 Syntax

Visual Basic:
Function GetSetting(settingID as SettingCodes)

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

5.6.3 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.

5.6.4 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**.

5.7 SetSetting Method

5.7.1 Description

Changes the value of a particular setting. Sets an associated internal variable. The setting is also sent immediately to MLX90640 evaluation board.

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

5.7.2 Syntax

Visual Basic:

Sub SetSetting(settingID as SettingCodes, Value)

C++:

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

5.7.3 Parameters

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

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

5.7.4 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.

5.8 OpenProfile Method

5.8.1 Description

Opens the specified file and updates the settings.

5.8.2 Syntax

Visual Basic:

Sub OpenProfile(FileName as String)

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

5.8.3 Parameters

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

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 SaveProfile Method

5.9.1 Description

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

5.9.2 Syntax

Visual Basic:
Sub SaveProfile()

C++:
HRESULT SaveProfile();

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 SaveProfileAs Method

5.10.1 Description

Saves the settings into the specified file.

5.10.2 Syntax

Visual Basic:

Sub SaveProfileAs(FileName as String)

C++:

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

5.10.3 Parameters

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

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 I2CWriteRead Method

5.11.1 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.

5.11.2 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);

5.11.3 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.

5.11.4 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.

6 Enumeration constants

6.1 ParameterCodesEEPROM enumeration

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

Constant	Value	Bits	Translation value	Description	90640AAA	90640CAA	90641AAA	90641CAA
CodeOscTrim	1	16	-		√	√	√	√
CodeAnalogTrim	2	16	-		√	√	√	√
CodeConfiguration	3	16	-		√	√	√	√
CodeI2CAddr	4	16	-		√	√	√	√
CodeCropPageAddr	5	16	-		√	√	√	√
CodeCropCellAddr	6	16	-		√	√	√	√
CodeControl1	7	16	-		√	√	√	√
CodeControl2	8	16	-		√	√	√	√
CodeI2CConf	9	16	-		√	√	√	√
CodeID1	10	16	-		√	√	√	√

Constant	Value	Bits	Translation value	Description	90640AAA	90640CAA	90641AAA	90641CAA
CodeID2	11	16	-		√	√	√	√
CodeID3	12	16	-		√	√	√	√
CodeDeviceOptions	13	16	-		√	√	√	√
CodeAnalogTrim2	14	16	-			√		√
CodeScale_Occ_rem	15	4	-		√	√		
CodeScale_Occ_col	16	4	-	Indexed [0 to 31]	√	√		
CodeScale_Occ_row	17	4	-	Indexed [0 to 23]	√	√		
CodeAlpha_PTAT	18	»	-		4	4	11	11
CodePix_os_average	19	16	-		√	√		
CodeOCC_row	20	4	-		√	√		
CodeOCC_column	21	4	-		√	√		
CodeScale_Acc_rem	22	4	-		√	√		
CodeScale_Acc_col	23	4	-	Indexed [0 to 31]	√	√		
CodeScale_Acc_row	24	4	-	Indexed [0 to 23]	√	√		
CodeAlpha_scale	25	4	-		√	√		
CodePix_sens_average	26	16	-		√	√		
CodeACC_row	27	4	-		√	√		
CodeACC_column	28	4	-		√	√		
CodeGAIN	29	»	-		16	16	11	11
CodePTAT_25	30	»	-		16	16	11	11
CodeKt_PTAT	31	»	-		10	10	11	11
CodeKv_PTAT	32	»	-		6	6	11	11
CodeVdd_25	33	»	-		8	8	11	11
CodeK_Vdd	34	»	-		8	8	11	11
CodeKv_Avg_RE_CE	35	4	-		√	√		
CodeKv_Avg_RO_CE	36	4	-		√	√		
CodeKv_Avg_RE_CO	37	4	-		√	√		
CodeKv_Avg_RO_CO	38	4	-		√	√		
CodeKta_Avg_RE_CO	39	8	-		√	√		
CodeKta_Avg_RO_CO	40	8	-		√	√		
CodeKta_Avg_RE_CE	41	8	-		√	√		
CodeKta_Avg_RO_CE	42	8	-		√	√		
CodeKta_scale2	43	4	-		√	√		
CodeKta_scale1	44	4	-		√	√		
CodeKv_scale	45	4	-		√	√		
CodeRes_control	46	2	-		√	√		
CodeAlpha_CP_P0	47	10	-		√	√		
CodeAlpha_CP_P1_P0	48	6	-		√	√		
CodeOffset_CP_P0	49	10	-		√	√		
CodeOffset_CP_P1_P0	50	6	-		√	√		
CodeKta_CP	51	»	-		8	8	6	6
CodeKv_CP	52	»	-		8	8	6	6
CodeTGC	53	»	-		8	8	9	9
CodeKsTa	54	»	-		8	8	11	11
CodeKsTo_R1	55	8	-		√	√		
CodeKsTo_R2	56	8	-		√	√		

Constant	Value	Bits	Translation value	Description	90640AAA	90640CAA	90641AAA	90641CAA
CodeKsTo_R3	57	8	-		√	√		
CodeKsTo_R4	58	8	-		√	√		
CodeScale_KsTo	59	4	-		√	√		
CodeCT1	60	4	-		√	√		
CodeCT2	61	4	-		√	√		
CodeTemp_Step	62	2	-		√	√		
CodePixel_Kta	63	»	-	Indexed [0 to 767] for 90640 Indexed [0 to 191] for 90641	3	3	6	6
CodePixel_Alpha	64	6	-	Indexed [0 to 767]	√	√		
CodePixel_Offset	65	6	-	Indexed [0 to 767]	√	√		
CodeScale_occ_os	66	6	-				√	√
CodePix_os_avg	67	11	-				√	√
CodeKta_avg	68	11	-				√	√
CodeKta_scale_2	69	5	-				√	√
CodeKta_scale_1	70	6	-				√	√
CodeKv_avg	71	11	-				√	√
CodeKv_scale_2	72	5	-				√	√
CodeKv_scale_1	73	6	-				√	√
CodeScale_row	74	5	-	Indexed [0 to 5]			√	√
Coderow_max	75	11	-	Indexed [0 to 5]			√	√
CodeEmissivity	76	11	-				√	√
CodeAlpha_CP	77	11	-				√	√
CodeAlpha_CP_scale	78	11	-				√	√
CodeOffset_CP	79	11	-				√	√
CodeKta_CP_scale	80	5	-				√	√
CodeKv_CP_scale	81	5	-				√	√
CodeCalib_res_cont	82	2	-				√	√
CodeKsTo_scale	83	11	-				√	√
CodeKsTo	84	11	-				√	√
CodePixel_os	85	11	-	Indexed [0 to 191]			√	√
CodePixel_Sensitivity	86	11	-	Indexed [0 to 191]			√	√
CodePixel_Kv	87	5	-	Indexed [0 to 191]			√	√

6.2 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	(Long) long	1000 [µs]	Delay to keep Vdd off for resetting
SettingTclock	3	(Single) float	1.0 [µs]	IR I2C clock speed
SettingTstart	4	(Single) float	1.0 [µs]	IR I2C Start condition delay
SettingTstop	5	(Single) float	1.0 [µs]	IR I2C Stop condition delay

Constant	Value	Type	Default value	Description
SettingTwrdelay	6	(Single) float	1.0 [µs]	IR delay between the write and read I2C commands
SettingI2Caddr	7	(Byte) unsigned char	51	The address of IR device
SettingTEEwrite	8	(Long) long	5000 [µs]	EEPROM write delay

6.3 ChipVersionCodes enumeration

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

Constant	Value	Description
ChipVersionUndefined	0	Chip version not recognized
ChipVersion90640AAA	1	Chip version is 90640AAA
ChipVersion90640CAA	2	Chip version is 90640CAA
ChipVersion90641AAA	3	Chip version is 90641AAA
ChipVersion90641CAA	4	Chip version is 90641CAA

6.4 DataProcessingTypes enumeration

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

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.
Note, that in current release, StartDaq is expecting constants not listed in this enumeration. See StartDaq method description.		

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