



## **Reference Guide**

# Microphone SDK API

For PowerScribe<sup>®</sup> SDK Version 3.2

## **Trademarks**

Nuance<sup>®</sup>, the Nuance logo, Dictaphone<sup>®</sup>, and PowerScribe<sup>®</sup> are trademarks or registered trademarks of Nuance Communications, Inc. or its affiliates in the United States and/or other countries. All other trademarks referenced herein are trademarks or registered trademarks of their respective owners.

## **Patents**

The PowerMic II product is the subject of pending U.S and foreign patent applications.

## **Copyright Notice**

This manual is copyrighted and all rights are reserved by Nuance Communications, Inc. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Nuance Communications, Inc., 1 Wayside Road, Burlington, MA 01803.

Copyright © 2007, 2009 Nuance Communications, Inc. All rights reserved.

## **Disclaimer**

Nuance makes no warranty, express or implied, with respect to the quality, reliability, currentness, accuracy, or freedom from error of this document or the product or products referred to herein and specifically disclaims any implied warranties, including, without limitation, any implied warranty of merchantability, fitness for any particular purpose, or non-infringement. Nuance disclaims all liability for any direct, indirect, incidental, consequential, special, or exemplary damages resulting from the use of the information in this document. Mention of any product not manufactured by Nuance does not constitute an endorsement by Nuance of that product.

Published by Nuance Communications, Inc.  
Burlington, Massachusetts, USA

**Visit Nuance Communications, Inc. on the Web at [www.nuance.com](http://www.nuance.com).**

# Contents

<b>Chapter 1: Introducing Microphone SDK Object Model &amp; APIs</b>	<b>1</b>
PowerMic II Microphone SDK Object Model	2
<b>Chapter 2: Microphone SDK API Reference</b>	<b>5</b>
IUSBDeviceMgr	6
DeviceCount	6
Device	7
_IUSBDeviceMgrEvents	9
DeviceConnected()	9
DeviceRemoved()	10
IUSBDevice and IPowerMicII	12
Manufacturer	12
ProductID	14
ProductString	16
VendorID	18
VersionNumber	20
IUSBDevice.	22
USBDeviceType	22
USBDeviceObject	24
IPowerMicII	25
IsConnected()	25
Scan()	26
ScanResult	27
ScannerPresent	28
SetLedState()	29
IPowerMicIIEx.	31
ExclusiveControl()	31
SetProcessID()	33
InterceptEventsForApplication()	34
_IPowerMicIIEvents	35
ButtonPress()	35
ButtonRelease()	38

Connected() .....	41
Removed() .....	42
ScanStarted() .....	43
ScanFinished() .....	44

**Appendix A: Values for PowerMic II Microphone Buttons  
in Microphone SDK API .....45**

PowerMic II Buttons in Microphone SDK .....	46
---	----

**Appendix B: Configuration for Microphone Sharing .....49**

Configuration for Microphone Sharing .....	50
--	----

# *Introducing Microphone SDK Object Model & APIs*

## **Objectives**

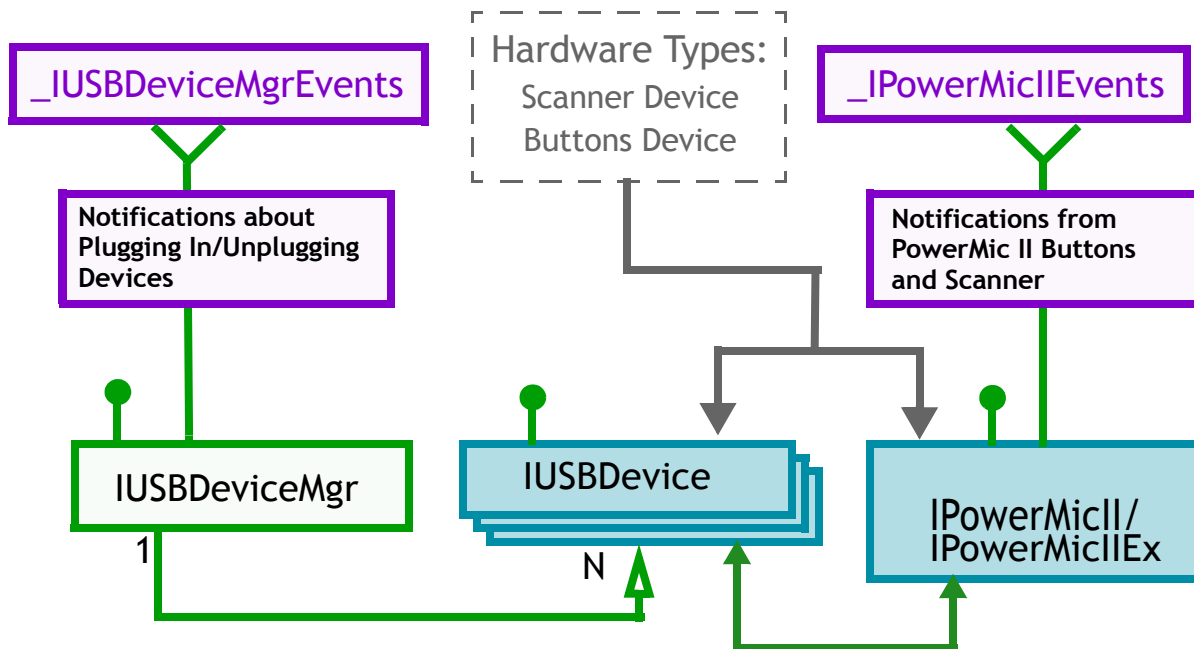
The *Microphone Software Development Kit (SDK)* is a set of application programmer interfaces (APIs) that you use to work with the *PowerMic II* microphone in your dictation and speech-recognition system.

This chapter presents an overview of the object model for the *PowerMic II Microphone SDK* and the types of capabilities that this *SDK* provides in [PowerMic II Microphone SDK Object Model](#).

# PowerMic II Microphone SDK Object Model

*PowerMic II Microphone SDK APIs* are available as a supplement to the full SDK Client.

The object model for these APIs (shown below) contains several types of objects and their related event objects. The **IUSBDevice** object represents any USB device, including a *PowerMic II* device, also represented by the **IPowerMicII** and the **IPowerMicIIEx** objects. Events of the **IUSBDevice** object are the plugging in and unplugging of any USB device, whereas the events of the **IPowerMicII** and the **IPowerMicIIEx** objects are specific to the *PowerMic II*—pressing buttons on the microphone or scanning with the microphone’s scanner.



The **IUSBDeviceMgr** object manages events from all USB devices (including the *PowerMic II*) by providing a **DeviceCount** property to determine how many devices have been connected and a **Device** property to retrieve an object for the device.

The **IUSBDevice** object provides a **USBDeviceType** property that helps you determine the type of device connected to a USB port, distinguishing between a *PowerMic II* with a scanner and one without a scanner. (The **usbdevUnknown** and **usbdevVecPedal** values are never returned in this version.) The **IUSBDevice** object also provides the **USBDeviceObject** property to retrieve an object for a particular USB device. In this release, the **IUSBDevice** object supports only the *PowerMic II* microphone.

The **\_IUSBDeviceMgrEvents** object provides **DeviceConnected()** and **DeviceRemoved()** methods triggered when those types of events occur.

The **IPowerMicII** object provides methods that make the *PowerMic II* microphone take particular actions or check the status of the microphone, such as **IsConnected()**, **Scan()**, and **SetLedState()**. This object also provides a **ScanResult** property that you use to retrieve the scanned data from the microphone and a **ScannerPresent** property that you use to determine whether or not the microphone has a scanner.

The **IPowerMicIIEx** object provides methods that enable sharing of the *PowerMic II* microphone across applications that have been enabled to listen for *PowerMic II* events.

Both the **IUSBDevice** and **IPowerMicII** objects provide a series of properties that you can use to determine information about the hardware itself, such as **Manufacturer**, **ProductID**, **ProductString**, **VendorID**, and **VersionNumber**.

The **\_IPowerMicIIEvents** object provides methods triggered when a user takes specific actions with the *PowerMic II* microphone, such as **ButtonPress()**, **ButtonRelease()**, **Connected()**, **Removed()**, **ScanStarted()**, and **ScanFinished()**.





# Microphone SDK API Reference

## Objectives

This chapter contains reference information on each *PowerMic II* API. The APIs are organized by these object types:

- [IUSBDeviceMgr](#)
- [\\_IUSBDeviceMgrEvents](#)
- [IUSBDevice and IPowerMicII \(properties that apply to both object types\)](#)
- [IUSBDevice](#)
- [IPowerMicII](#)
- [IPowerMicIIEx](#)
- [\\_IPowerMicIIEvents](#)

The examples shown in this chapter are all COM implementations in C++. However, you can use these APIs in other contexts, including on platforms other than Windows and in languages other than C++.

---

---

# IUSBDeviceMgr

## Properties

---

---

### DeviceCount

#### Purpose:

Read-only property of the **IUSBDeviceMgr** object. Returns the total number of USB devices available on the computer running the application.

#### Prototype:

```
Propget HRESULT DeviceCount( [out, retval] LONG* pVal );
```

#### Parameters:

##### pVal

Pointer to the number of USB devices on the computer. Is never NULL.

#### Example in Visual Basic

To check the number of USB devices connected to the computer running your speech-recognition application, you can check the value of the **DeviceCount** property. First you instantiate the **IUSBDeviceMgr** object, then retrieve its **DeviceCount** value:

```
Dim index As Integer
Dim ncount As Integer

usbmgr = New USBMGRLib.USBDeviceMgr
ncount = usbmgr.DeviceCount
index = 0
```

Once you have the number of devices, you can take action on each device in a loop:

```
While (index < ncount)
    usbdevice = usbmgr.Device(index)
    index = index + 1
```

```
        ... // take action on each device
End While
```

## Example in C++:

To check the number of USB devices connected to the computer running your speech-recognition application, you can check the value of the **DeviceCount** property:

```
long nDcount;

if (&nDcount == IUSBDeviceMgr->DeviceCount())
{
    return nDcount;
}
else
{
    AfxMessageBox( _T("No USB device is connected.\n"));
}
}
```

## Returned Values:

Number of USB devices on the computer.

# Device

## Purpose:

Read-only property of the **IUSBDeviceMgr** object. Returns the USB device object identified by the index number you pass it. That identifier is a number that is unique for the USB port on the computer running the application.

## Prototype:

```
Propget HRESULT Device( [in] LONG index, [out, retval] IUSBDevice** pVal );
```

## Parameters:

### index

Index number of the device on the computer, starting at **0** and progressing to the maximum device number of **DeviceCount - 1**.

### pVal

Returned value. Pointer to the USB device object identified by the index.

## Example in Visual Basic:

After you instantiate the **IUSBDeviceMgr** object, to retrieve an object for the device connected to the USB port of the computer running your application, you call the **Device** property and pass it the number of the device. If you have one microphone, it is device zero (0). If you have more than one, to retrieve the first device, you can start with 0 and loop through all the devices until you find the first one that is not NULL:

```
Dim index As Integer
Dim ncount As Integer

usbmgr = New USBMGRLib.USBDeviceMgr

FindFirstDevice:
    While (index < ncount)
        objDevice = usbmgr.Device(index)
        index = index + 1
        If (objDevice != NULL)
            return objDevice
            GoTo UseDeviceRoutine
        End If
    End While

UseDeviceForDictation:
    rem Execute dictation
```

## Example in C++:

To retrieve an object for the device connected to the USB port of the computer running your application, you can use the **Device** property and pass it the number of the device. If you have one microphone, it is device zero (0):

```
if (&&nMicDeviceObj == IUSBDeviceMgr->Device(0))
{
    return &nMicDeviceObj;
}
else
{
    AfxMessageBox( _T("No USB device is connected.\n");
}
}
```

## Returned Values:

Pointer to the USB device object.

---

---

# **\_IUSBDeviceMgrEvents**

## **Methods**

You should implement this event notification object on the client side. Returned values of the methods of this object are usually S\_OK and you can choose to ignore them.

---

---

## **DeviceConnected()**

### **Purpose:**

Method of the **\_IUSBDeviceMgrEvents** object. The framework calls this method when a new device has been plugged in to a USB port. The method call indicates that the device is available for use.

### **Prototype:**

```
HRESULT DeviceConnected([in] IUSBDevice* pDevice);
```

### **Parameters:**

#### **pDevice**

Pointer to the USB device just plugged in to the computer.

### **Example in C++:**

To take action when a device is plugged in to a USB port, you should take that action in the **DeviceConnected()** method of the **\_IUSBDeviceMgrEvents** object. When you define the class for the object, you should define the **DeviceConnected()** method. To have the method initiate tests of the microphone and display the results in a dialog whenever a device is plugged in, you can have the **DeviceConnected()** method call a function that takes that action:

```
STDMETHODIMP CUsbMgrSink::DeviceConnected(IUSBDevice* pDevice)
{
    if ( !m_pDlg )
        return E_FAIL;
```

```
{
    m_pDevice->InitiateMicrophoneTests();
    m_pDlg->DisplayResults();
    return S_OK;
}
```

### Returned Values:

None.

## DeviceRemoved()

### Purpose:

Method of the **\_IUSBDeviceMgrEvents** object. The framework calls this method when a device has been unplugged from a USB port. The method call indicates that the device is no longer available for use.

### Prototype:

```
HRESULT DeviceRemoved([in] IUSBDevice* pDevice);
```

### Parameters:

#### pDevice

Pointer to the USB device just unplugged from the computer.

### Example:

To take action when a device is unplugged from a USB port, you should take that action in the **DeviceRemoved()** method of the **\_IUSBDeviceMgrEvents** object. First, you should define the class for the object, then define the **DeviceRemoved()** method. To have the method check whether or not the device removed was a *PowerMic II*, you can have the method test the **USBDeviceType** property value of the device passed in to the method (**pDevice**) and if the device is a **usbdevPowerMic2** type device, then take appropriate action:

```
STDMETHODIMP CUsbMgrSink::DeviceRemoved(IUSBDevice* pDevice)
{
    ...
    pDevice->get_USBDeviceType(&pType);
    if( pType == usbdevPowerMic2 )
    {
        AfxMessageBox( _T("Microphone disconnected. \n");
        m_pDlg->StopMicDialogDisplay();
        return S_OK;
    }
}
```

```
    return E_FAIL;  
}
```

**Returned Values:**

None.

---

---

# IUSBDevice and IPowerMicII

## Properties Applicable to Both Objects

---

---

### Manufacturer

#### Purpose:

Get property of the **IUSBDevice** or **IPowerMicII** object. Returns a pointer to a string containing the name of the manufacturer of the device plugged in to the USB port.

#### Prototype:

Propget HRESULT Manufacturer([out, retval] BSTR\* pVal);

#### Parameters:

##### pVal

Returned value. Pointer to a string containing the name of the product manufacturer.

#### Example in Visual Basic:

You can retrieve the microphone information from the hardware using the properties of the **IPowerMicII** object. For instance, to retrieve the name of the manufacturer so that you can later display the name in a dialog, you would use the **Manufacturer** property of the object and retrieve the returned string inside the routine that updates the GUI:

```
Private Sub FillData(ByVal pDevice As USBMGRLib.IUSBDevice)

    Dim strMfr As String
    ...

    strMfr = pDevice.Manufacturer

    ...
    TextBox4.Text = strMfr
    ...

```



End Sub

### Example 1 in C++:

You can retrieve the microphone information from the hardware using the properties of the **IPowerMicII** object. For instance, to retrieve the name of the manufacturer so that you can later display the name in a dialog, you would use the **Manufacturer** get property of the object and pass it the pointer to retrieve the returned string, then call a user-defined function to update the GUI:

```
void CUsbmgr_testDlg::FillData( IUSBDevice *pDevice )
{
    if ( !pDevice )
        return;

    HRESULT hr;
    CComBSTR bstrManufacturer;

    hr = pDevice->get_Manufacturer( &bstrManufacturer );

    if ( FAILED(hr) )
        return;

    m_strManufacturer = bstrManufacturer;

    UpdateData( FALSE );
}
```

### Example 2 in C++:

To retrieve the name of the microphone manufacturer, you can create a function that calls the get property **Manufacturer** and pass it a pointer to return the string in:

```
BSTR* function DisplayManufacturuer(IPowerMicII* pDevice)
{
    pDevice->get_Manufacturer(&bstrManuf);
    if (bstrManuf)
    {
        AfxMessageBox( _T("Microphone Manufacturer is " + pManuf + "\n"));
        return bstrManuf;
    }
    return E_FAIL;
}
```

### Returned Values:

Pointer to the string of the product manufacturer.

# ProductID

## Purpose:

Get property of the **IUSBDevice** or **IPowerMicII** object. Returns the numeric product ID for the device plugged in to the USB port.

## Prototype:

Propget HRESULT ProductID([out, retval] USHORT\* pVal);

## Parameters:

### pVal

Returned value. Pointer to the product ID number.

## Example in Visual Basic:

You can retrieve the microphone information from the hardware using the properties of the **IPowerMicII** object. For instance, to retrieve the identifier for the product so that you can later display the name in a dialog, you would use the **ProductID** property of the object and retrieve the returned short integer inside the routine that updates the GUI:

```
Private Sub FillData(ByVal pDevice As USBMGRLib.IUSBDevice)

    Dim nProductID As Short

    nProductID = pDevice.ProductID

    ...
    TextBox2.Text = nProductID.ToString
    ...

End Sub
```

## Example 1 in C++:

You can set the retrieve the microphone information from the hardware using the properties of the **IPowerMicII** object. For instance, to retrieve the numeric product ID to later display the number in a dialog, you would use the **ProductID** get property of the object and pass it the pointer to retrieve the returned number, then call a user-defined function to update the GUI:

```
void CUsbmgr_testDlg::FillData( IUSBDevice *pDevice )
{
    if ( !pDevice )
        return;

    HRESULT hr;
    USHORT nProductID;
```

```
hr = pDevice->get_ProductID( &nProductID );

if ( FAILED(hr) )
    return;

m_nProductID = nProductID;

UpdateData( FALSE );
}
```

### Example 2 in C++:

To retrieve the numeric microphone product ID, you can create a function that calls the get property **ProductID** and pass it a pointer to retrieve the returned numeric value:

```
USHORT* function DisplayProductID(IPowerMicII* pDevice)
{
    pDevice->get_ProductID( &nProductID);
    if ( nProductID )
    {
        AfxMessageBox( _T("Microphone Product ID is " + nProductID + "\n"));
        m_pDlg->OpenPowerMicIIDlg();
        return nProductID;
    }
    return E_FAIL;
}
```

### Returned Values:

Pointer to the product ID number.

# ProductString

## Purpose:

Get property of the **IUSBDevice** or **IPowerMicII** object. Retrieves a string containing the product name of the device plugged in to the USB port.

## Prototype:

```
Propget HRESULT ProductString([out, retval] BSTR* pVal);
```

## Parameters:

### pVal

Returned value. Pointer to the string containing the product name.

## Example in Visual Basic:

You can retrieve the microphone information from the hardware using the properties of the **IPowerMicII** object. For instance, to retrieve the name of the product so that you can later display the name in a dialog, you would use the **ProductString** property of the object and retrieve the returned string inside the routine that updates the GUI:

```
Private Sub FillData(ByVal pDevice As USBMGRLib.IUSBDevice)

    Dim strproduct As String

    strproduct = pDevice.ProductString

    ...

    TextBox5.Text = strproduct

End Sub
```

## Example 1 in C++:

You can set the retrieve the microphone information from the hardware using the properties of the **IPowerMicII** object. For instance, to retrieve the name of the product in a string, you would use the **ProductString** get property of the object and pass it the pointer to retrieve the string returned, then call a user-defined function to update the GUI:

```
void CUsbmgr_testDlg::FillData( IUSBDevice *pDevice )
{
    if ( !pDevice )
        return;

    HRESULT hr;
    CComBSTR bstrProductString;
```

```
hr = pDevice->get_ProductString( &bstrProductString );

if ( FAILED(hr) )
    return;

m_strProductString = bstrProductString;

UpdateData( FALSE );
}
```

### Example 2 in C++:

To retrieve a string containing the name of the microphone, you can create a function that calls the get property **ProductString** and pass it a pointer to receive the string returned:

```
BSTR* function DisplayProductString(IPowerMicII* pDevice)
{
    pDevice->get_ProductString(&bstrProductString);
    if (bstrProductString)
    {
        AfxMessageBox( _T("Microphone Product is " + bstrProductString + "\n"));
        m_pDlg->OpenPowerMicIIDlg();
        return bstrProductString;
    }
    return E_FAIL;
}
```

### Returned Values:

Pointer to the product name string.

# VendorID

## Purpose:

Get property of the **IUSBDevice** or **IPowerMicII** object. Returns a pointer to the numeric ID of vendor for the device plugged in to the USB port.

## Prototype:

Propget HRESULT VendorID([out, retval] USHORT\* pVal);

## Parameters:

### pVal

Returned value. Pointer to the product vendor ID number.

## Example in Visual Basic:

You can retrieve the microphone information from the hardware using the properties of the **IPowerMicII** object. For instance, to retrieve the numeric identifier for the vendor so that you can later display the identifier in a dialog, you would use the **VendorID** property of the object and retrieve the returned short integer inside the routine that updates the GUI:

```
Private Sub FillData(ByVal pDevice As USBMGRLib.IUSBDevice)

    Dim nvendorID As Short

    nvendorID = pDevice.VendorID

    TextBox1.Text = nvendorID.ToString
    ...

End Sub
```

## Example 1 in C++:

You can set the retrieve the microphone information from the hardware using the properties of the **IPowerMicII** object. For instance, to retrieve the numeric ID of the vendor, you would use the **VendorID** get property of the object and pass it the pointer to retrieve the value returned, then call a user-defined function to update the GUI:

```
void CUsbmgr_testDlg::FillData( IUSBDevice *pDevice )
{
    if ( !pDevice )
        return;

    HRESULT hr;
    USHORT nVendorID;
```

```
hr = pDevice->get_VendorID( &nVendorID );

if ( FAILED(hr) )
    return;

m_nVendorID = nVendorID;

UpdateData( FALSE );
}
```

### Example 2 in C++:

To retrieve a string containing the ID of the microphone's vendor, you can create a function that calls the get property **VendorID** and pass it a pointer to return the string in:

```
BSTR* function DisplayVendorID(IPowerMicII* pDevice)
{
    pDevice->get_VendorID(&nVendorID);
    if (nVendorID)
    {
        AfxMessageBox( _T("Microphone Vendor ID is " + nVendorID + "\n"));
        m_pDlg->OpenPowerMicIIDlg();
        return nVendorID;
    }
    return E_FAIL;
}
```

### Returned Values:

Pointer to the vendor ID number.

# VersionNumber

## Purpose:

Get property of the **IUSBDevice** or **IPowerMicII** object. Returns a pointer to the version number the device plugged in to the USB port.

## Prototype:

Propget HRESULT VersionNumber([out, retval] USHORT\* pVal);

## Parameters:

### pVal

Returned value. Pointer to the product version number.

## Example in Visual Basic:

You can retrieve the microphone information from the hardware using the properties of the **IPowerMicII** object. For instance, to retrieve the version of the software in the microphone so that you can later display the version in a dialog, you would use the **VersionNumber** property of the object and retrieve the returned short integer inside the routine that updates the GUI:

```
Private Sub FillData(ByVal pDevice As USBMGRLib.IUSBDevice)

    Dim nVersionNumber As Short

    nVersionNumber = pDevice.VersionNumber

    ...
    TextBox3.Text = nVersionNumber.ToString
    ...

End Sub
```

## Example 1 in C++:

You can set the retrieve the microphone information from the hardware using the properties of the **IPowerMicII** object. For instance, to retrieve the version number of the microphone, you would use the **VersionNumber** get property of the object and pass it the pointer to retrieve the value returned, then call a user-defined function to update the GUI:

```
void CUsbmgr_testDlg::FillData( IUSBDevice *pDevice )
{
    if ( !pDevice )
        return;

    HRESULT hr;
```



```
USHORT nVersionNumber;

hr = pDevice->get_VersionNumber( &nVersionNumber );

if ( FAILED(hr) )
    return;

m_nVersionNumber = nVersionNumber;

UpdateData( FALSE );
}
```

### Example 2 in C++:

To retrieve version of the microphone, you can create a function that calls the get property **VersionNumber** and pass it a pointer to retrieve the returned value:

```
USHORT* function DisplayVersionNumber(IPowerMicII* pDevice)
{
    pDevice->get_VersionNumber(&nVersionNumber);
    if (nVersionNumber)
    {
        AfxMessageBox( _T("Microphone Version is " + nVersionNumber + "\n"));
        m_pDlg->OpenPowerMicIIDlg();
        return nVersionNumber;
    }
    return E_FAIL;
}
```

### Returned Values:

Pointer to the product version number.

---

---

# IUSBDevice

## Exclusive Properties

---

---

## USBDeviceType

### Purpose:

Get property of the **IUSBDevice** object. Returns the device type of the device plugged in to the USB port.

### Prototype:

Propget HRESULT USBDeviceType ([out, retval] USBDevicesType\* pVal);

### Parameters:

#### pVal

Returned value. Pointer to the type of device.

USBDevicesType Constant	Equivalent Numeric Value	Comments
usbdevUnknown	0	Never returned in this version.
usbdevPowerMic2	1	Returned when the <i>PowerMic II</i> has no scanner.
usbdevPowerMic2Scanner	2	Returned when the <i>PowerMic II</i> has a scanner.
usbdevVecPedal	3	Never returned in this version.

### Example in Visual Basic:

To determine whether or not the device currently connected to the machine running the application is a *PowerMic II*, after you instantiate the device manager, then retrieve the device, you can retrieve the value of the **USBDeviceType** property of the **IUSBDevice** object

and check to be sure the device (**usbdevice**) is a **usbdevPowerMic2** type device, then take appropriate action:

```
Private Sub InitializePMII()
    Dim index As Integer
    Dim devtype As USBMGRLib.USBDevicesType
    Dim ncount As Integer
    usbmgr = New USBMGRLib.USBDeviceMgr
    ncount = usbmgr.DeviceCount
    index = 0

    While (index < ncount)
        usbdevice = usbmgr.Device(index)
        index = index + 1
        devtype = usbdevice.USBDeviceType
        If (devtype = USBMGRLib.USBDevicesType.usbdevPowerMic2) Then
            rem Take Appropriate Action
        End If
    End While
    ...
End Sub
```

### Example in C++:

To determine whether or not the device currently connected to the machine running the application is a *PowerMic II*, you can get the **USBDeviceType** property value and check to be sure the device (**pDevice**) is a **usbdevPowerMic2** type device, then take appropriate action:

```
function ConnectedDeviceTypeIsPowerMicII(IUSBDevice* pDevice)
{
    pDevice->get_USBDeviceType(&nType);
    if( nType == usbdevPowerMic2 )
    {
        m_pDlg->OpenPowerMicIIDlg();
        return S_OK;
    }
    return E_FAIL;
}
```

### Returned Values:

Pointer to the type of device.

# USBDeviceObject

## Purpose:

Get property of the **IUSBDevice** object. Returns a pointer to the device plugged in to the USB port.

## Prototype:

Propget HRESULT USBDeviceObject ([out, retval] IDispatch\*\* pVal);

## Parameters:

### pVal

Returned value. IDispatch (object) pointer to the device plugged in to the USB port.

## Example in Visual Basic:

After you have determined that the device type is a *PowerMic II*, you can retrieve an object for the microphone using the **USBDeviceObject** property:

```
Private Sub InitializePMII()  
    ' Initialize PowerMic if connected  
    Dim index As Integer  
    ...  
    usbmgr = New USBMGRLib.USBDeviceMgr  
    ...  
    usbdevice = usbmgr.Device(index)  
    ...  
    usbpm2 = usbdevice.USBDeviceObject  
End Sub
```

## Example in C++:

To retrieve a pointer to the device currently plugged in to the USB port of the machine running the application, you can retrieve the **USBDeviceObject** property value and return the pointer:

```
IDispatch* function RetrieveAudioDeviceObject(IUSBDevice* pDevice)  
{  
    pDevice->get_USBDeviceObject(&pMicObject);  
    if( &pMicObject )  
    {  
        m_pDlg->OpenPowerMicIIDlg();  
        return &pMicObject;  
    }  
    return E_FAIL;  
}
```

## Returned Values:

Pointer to the string of the product manufacturer.

---

---

# IPowerMicII

## Exclusive Methods and Properties

---

---

### IsConnected()

#### Purpose:

Method of the **IPowerMicII** object. Returns 1 if the USB device is connected and ready to use, returns 0 if the device is not connected or not available to use.

#### Prototype:

```
HRESULT IsConnected([out,retval] USHORT* bVal);
```

#### Parameters:

##### **bVal**

**True** if the microphone is connected, **False** if it is not.

#### Example in C++:

To determine whether the microphone is connected to the computer running the application, you can call the **IsConnected()** method of the **IPowerMicII** object in an **if** statement:

```
if ( pMicDeviceObject->IsConnected() )
{
    if (pMicDeviceObject->ScannerPresent)
    {
        // Start Scanning
        pMicDeviceObject->Scan();
        BSTR* bsScannedString = pMicDeviceObject->ScanResults();
    }
}
```

#### Returned Values:

Connected status of the microphone.

# Scan()

## Purpose:

Method of the **IPowerMicII** object. Initiates decoding a bar code with the scanner on the *PowerMic II* microphone.

## Prototype:

HRESULT Scan(void);

## Parameters:

None.

## Example in C++:

Once you determine the microphone is connected to the computer running the application and that the microphone has a scanner, you can scan a bar code by calling the **Scan()** method of the **IPowerMicII** object:

```
if ( pMicDeviceObject->IsConnected() )
{
    if (pMicDeviceObject->ScannerPresent)
    {
        // Start Scanning
        pMicDeviceObject->Scan() ;
        BSTR* bsScannedString = pMicDeviceObject->ScanResults();
    }
}
```

## Returned Values:

None.

# ScanResult

## Purpose:

Property of the **IPowerMicII** object. Returns a string containing the results of the last scan.

## Prototype:

```
HRESULT ScanResult([out, retval] BSTR* pVal);
```

## Parameters:

### pVal

Returned value. String containing the results of the last scan.

## Example in C++:

After you call the **Scan()** method of the **IPowerMicII** object, you retrieve the results of the scan using the **ScanResult** property of the same object:

```
if ( pMicDeviceObject->IsConnected())
{
    if (pMicDeviceObject->ScannerPresent)
    {
        // Start Scanning
        pMicDeviceObject->Scan();
        BSTR* bsScannedString = pMicDeviceObject->ScanResult();
    }
}
```

## Returned Values:

String containing the results of the last scan.

# ScannerPresent

## Purpose:

Property of the **IPowerMicII** object. Returns a BOOL value that reflects whether or not the scanner is present.

## Prototype:

```
HRESULT ScannerPresent([out, retval] VARIANT_BOOL *pVal);
```

## Parameters:

### pVal

VARIANT\_BOOL pointer that receives VARIANT\_TRUE if the scanner is present and VARIANT\_FALSE if it is not.

## Example in C++:

Once you determine the microphone is connected to the computer running the application, you can check to see if that microphone has a scanner using the **ScannerPresent** method of the **IPowerMicII** object:

```
if ( pMicDeviceObject->IsConnected() )
{
    if (pMicDeviceObject->ScannerPresent)
    {
        // Start Scanning
        pMicDeviceObject->Scan();
        BSTR* bsScannedString = pMicDeviceObject->ScanResults();
    }
}
```

## Returned Values:

Connected status of the microphone.



# SetLedState()

## Purpose:

Method of the **IPowerMicII** object. Sets the state of the LED light on the *PowerMic II* to either turn off both red and green diodes or to turn on one of these combinations of diodes:

- Both a green and a red diodes
- Only the green diode
- Only the red diode.

## Prototype:

```
HRESULT SetLedState([in] MicLedStateType state);
```

## Parameters:

### state

**MicLedStateType** constant or its numeric equivalent that indicates color you want the LED light on the *PowerMic II* to display during dictation. You can have the LED light turn red, green, or fluctuate between red and green if you want it to be on during dictation. Otherwise, you can have it turn off (not light up at all during dictation).

tagMicLedStateType Constant	Equivalent Numeric Value	Explanation
micLedGreenRed	0x0300	Both diodes on.
micLedGreen	0x0200	Only the green diode on.
micLedRed	0x0100	Only the red diode on.
micLedOff	0x0000	Both diodes off.

## Example 1 in C++:

When the **DICTATE** button on the *PowerMic II* microphone has been pressed, you can set the LED on the microphone to green by calling the **SetLedState()** method of the **IPowerMicII** object and passing it the **micLedGreen** constant:

```
STDMETHODIMP CPowerMicSink::ButtonPressed(PMII_MICBUTTONS btncode)
{
    switch(btncode)
    {
        case BTN_DICTATE:
            pMicDeviceObject->SetLedState(micLedGreen);
            pMicDeviceObject->StartDictation();
        }
    }
}
```

## Example 2 in C++:

You can allow the user of your application to set the color that the LED on the microphone displays during dictation by calling the **SetLedState()** method of the **IPowerMicII** object and passing it the constant that corresponds to the color the user chooses from a combo box. Once the user makes the selection, you can call the **SetLedState()** method in the handler for the combo box:

```
BOOL CUsbmgr_testDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    ...

    LPCTSTR pszDescription[] = { _T("Off"), _T("On - red"), _T("On - green"),
        _T("On - red & green") };
    DWORD dwData[] = { (DWORD)micLedOff, (DWORD)micLedRed, (DWORD)micLedGreen,
        (DWORD)micLedGreenRed };
    for ( int i = 0; i < sizeof(dwData)/sizeof(DWORD); i++ )
    {
        int nItem = m_ctlLight.AddString( pszDescription[i] );
        ATLASSTERT( nItem != CB_ERR );
        m_ctlLight.SetItemData( nItem, dwData[i] );
    }

    m_ctlLight.SetCurSel( 0 );
    ...
}

void CUsbmgr_testDlg::OnCbnSelchangeComboLight()
{
    int nItem = m_ctlLight.GetCurSel();
    if ( nItem == CB_ERR )
        return;

    if( m_pPowerMic )
    {
        MicLedStateType _type = (MicLedStateType)m_ctlLight.GetItemData( nItem );
        HRESULT hr = m_pPowerMic->SetLedState( _type );
    }
}
```

## Returned Values:

Success if the setting succeeds.

---

---

# IPowerMicIIEx

## Exclusive Methods and Properties

---

---

### ExclusiveControl()

#### Purpose:

Method of the **IPowerMicII** object. Method for assigning exclusive access to microphone events. For your application to take exclusive control over the microphone, it should call the **ExclusiveControl()** method. After a call to this method, only the application that called it can receive the *PowerMic II* events.

You set the operation *ExclusiveControlOperation* argument to **exclControlSet** to take control of the microphone:

```
m_Microphone.ExclusiveControl(ul_ID, ExclusiveControlOperation.exclControlSet)
```

While your application has exclusive control over the microphone, no other application can use it until you later call the method again and pass it **exclControlRelease** to release the microphone from exclusive use by your application:

```
m_Microphone.ExclusiveControl(ul_ID, ExclusiveControlOperation.exclControlRelease)
```

#### Prototype:

```
HRESULT ExclusiveControl([in] ULONG ulAdviseCookie, [in]ExclusiveControlOperation operation );
```

#### Parameters:

##### **ulAdviseCookie**

Unsigned long. Cookie that the **Advise()** method returned.

##### **ExclusiveControlOperation**

Operation that you should take on the microphone, either securing it for exclusive use of the application (**exclControlSet**) or releasing it (**exclControlRelease**).

### **Example in C++:**

To secure the microphone for exclusive use of your application, you can call the **ExclusiveControl()** method of the **IPowerMicII** object:

```
pMicDeviceObject->ExclusiveControl (ul_ID, exclControlSet);
```

### **Returned Values:**

None.

---

# SetProcessID()

## Purpose:

Method of the **IPowerMicII** object. Method for connecting process to calling application. The new **SetProcessID()** method binds the cookie returned by the **Advise()** method with the process ID of the application that called **Advise()**. Your client must call this method after calling **Advise()** in order to receive *PowerMic II* events. If the client application does not call this method, it does not receive notifications about the *PowerMic* events unless the application has exclusive control of the microphone due to a previous call of **ExclusiveControl()**.

## Prototype:

```
HRESULT SetProcessID([in] ULONG ulAdviseCookie, [in] ULONG ulProcessID);
```

## Parameters:

### **ulAdviseCookie**

Unsigned long. Cookie that the **Advise()** method returned.

### **ulProcessID**

Unsigned long. Process ID of the application that called the **Advise()** method.

## Example in C++:

To associate a process with the application that called the **Advise()** method, you pass the cookie **Advise()** returns and the process ID of the application to the **SetProcessID()** method of the **IPowerMicII** object:

```
pMicDeviceObject->SetProcessID(ulCookie, ulProcessID)
```

## Returned Values:

None.

# InterceptEventsForApplication()

## Purpose:

Method of the **IPowerMicIIEx** object. Method to enable a client application to intercept events intended for other applications.

## Prototype:

HRESULT InterceptEventsForApplication([in] ULONG dwCookie, BSTR bstrAppName, [in] BOOL bIntercept);

## Parameters:

### **dwCookie**

Unsigned long. Cookie that the **Advise()** method returns.

### **bstrAppName**

String containing the name of the application from which events should be intercepted.

### **bIntercept**

True to intercept events or False to stop intercepting.

## Example in C++:

To intercept events intended for the `usbmgr_test` application, the client code would call:

```
pMicDeviceObject->InterceptEventsForApplication(ul_ID, appName, true);
```

To stop intercepting, the client would call:

```
pMicDeviceObject->InterceptEventsForApplication(ul_ID, appName, false);
```

## Returned Values:

None.

---

---

# \_IPowerMicIIEvents

## Properties

---

---

## ButtonPress()

### Purpose:

Method of the **IPowerMicIIEvents** object. The framework calls this method when any single button on the *PowerMic II* has been pressed.

### Prototype:

HRESULT ButtonPress([in] PMII\_MICBUTTONS btncode);

### Parameters:

#### btncode

PMII\_MICBUTTONS constant or its equivalent numeric value that reflects the button pressed on the *PowerMic II*.

<b>PMII_MICBUTTONS Constant</b>	<b>Equivalent Numeric Value</b>
BTN_TRANSCRIBE	0x0001
BTN_TABBACKWARD	0x0002
BTN_DICTATE	0x0004
BTN_TABFORWARD	0x0008
BTN_REWIND	0x0010
BTN_FASTFORWARD	0x0020
BTN_STOPPLAY	0x0040
BTN_CUSTOMLEFT	0x0080
BTN_ENTERSELECT	0x0100

PMII_MICBUTTONS Constant	Equivalent Numeric Value
BTN_CUSTOMRIGHT	0x0200
BTN_SCAN	0x0400

### Example in Visual Basic:

To have your application react when a *PowerMic II* button has been pressed, your code should include a **ButtonPress()** method of the **IPowerMicIIEvents** object. First, you establish the *PowerMic II* object, then create the method of that object, having it receive the **btncode** as an argument, the **PMII\_MICBUTTONS** constant from the table above that indicates which button was pressed.

```
Private Sub usbpm2_ButtonPress(ByVal btncode As USBMGRLib.PMII_MICBUTTONS)
    Handles usbpm2.ButtonPress
    Select Case btncode
        Case USBMGRLib.PMII_MICBUTTONS.BTN_DICTATE
            DgnMicBtn1.MicState = DNSTools.DgnMicStateConstants.dgmicOn
    End Select
End Sub
```

### Example in C++:

To have your application react when a *PowerMic II* button has been pressed, your code should include a **ButtonPress()** method of the **IPowerMicIIEvents** object. First, you establish the events object, then create the method of that object, having it receive the **btncode** as an argument, the **PMII\_MICBUTTONS** constant from the table above that indicates which button was pressed.

The **ButtonPress()** method would test the **btncode** to determine the particular button that has been pressed, then take appropriate action.

The code for the method might have a case for each possible button that could be pressed on the microphone. For example, in the code below, the dialog (**m\_pDlg**) displays a check box that corresponds to each button that could be pressed on the microphone. The check box that corresponds to the button pressed is set to checked (TRUE):

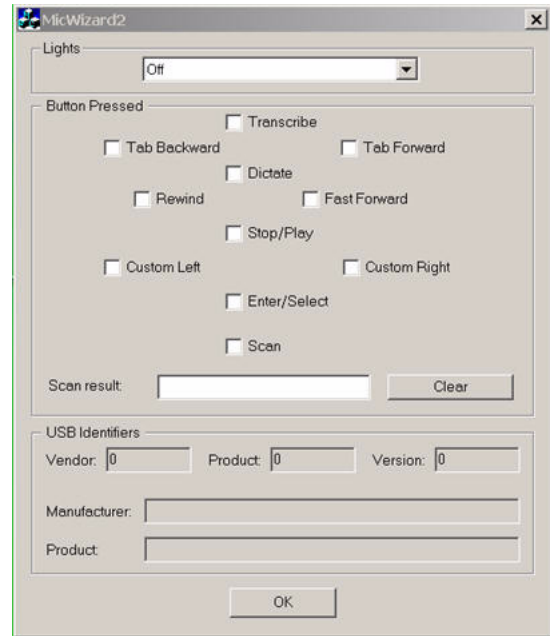
```
STDMETHODIMP CPowerMicSink::ButtonPress( PMII_MICBUTTONS btncode )
{
    ATLTRACE( "Button press codebutton = %x\n", btncode );
    switch( btncode )
    {
        case BTN_TRANSCRIBE:
```



```
        m_pDlg->m_bTranscribe = TRUE;
        break;
    case BTN_TABBACKWARD:
        m_pDlg->m_bTabBackward = TRUE;
        break;
    case BTN_DICTATE:
        m_pDlg->m_bDictate = TRUE;
        break;
    case BTN_TABFORWARD:
        m_pDlg->m_bTabForward = TRUE;
        break;
    case BTN_REWIND:
        m_pDlg->m_bRewind = TRUE;
        break;
    case BTN_FASTFORWARD:
        m_pDlg->m_bFastForward = TRUE;
        break;
    case BTN_STOPPLAY:
        m_pDlg->m_bStopPlay = TRUE;
        break;
    case BTN_CUSTOMLEFT:
        m_pDlg->m_bCustomLeft = TRUE;
        break;
    case BTN_ENTERSELECT:
        m_pDlg->m_bEnterSelect = TRUE;
        break;
    case BTN_CUSTOMRIGHT:
        m_pDlg->m_bCustomRight = TRUE;
        break;
    case BTN_SCAN:
        m_pDlg->m_bScan = TRUE;
        break;
    }
    m_pDlg->UpdateData( FALSE );

    if( btncode == BTN_SCAN )
        m_pDlg->Scan();

    return S_OK;
}
```



**Returned Values:**

None.

# ButtonRelease()

## Purpose:

Method of the **IPowerMicIIEvents** object. The framework calls this method when a previously pressed button on the *PowerMic II* has been released.

## Prototype:

HRESULT ButtonRelease([in] PMII\_MICBUTTONS btncode);

## Parameters:

### btncode

**tagPMII\_MICBUTTONS** constant or its equivalent numeric value that reflects the button pressed on the *PowerMic II*.

<b>tagPMII_MICBUTTONS Constant</b>	<b>Equivalent Numeric Value</b>
BTN_TRANSCRIBE	0x0001
BTN_TABBACKWARD	0x0002
BTN_DICTATE	0x0004
BTN_TABFORWARD	0x0008
BTN_REWIND	0x0010
BTN_FASTFORWARD	0x0020
BTN_STOPPLAY	0x0040
BTN_CUSTOMLEFT	0x0080
BTN_ENTERSELECT	0x0100
BTN_CUSTOMRIGHT	0x0200
BTN_SCAN	0x0400

## Example in Visual Basic:

To have your application react when a *PowerMic II* button that has been pressed is released, your code should include a **ButtonRelease()** method of the **IPowerMicIIEvents** object.

First, you establish the *PowerMic II* object, then create the method of that object, having it receive the **btncode** as an argument, the **PMII\_MICBUTTONS** constant from the table above that indicates which button was released.

```
Private Sub usbpm2_ButtonRelease(ByVal btncode As USBMGRLib.PMII_MICBUTTONS)
    Handles usbpm2.ButtonRelease
    Select Case btncode
```

```
        Case USBMGRLib.PMII_MICBUTTONS.BTN_DICTATE
            DgnMicBtn1.MicState = DNSTools.DgnMicStateConstants.dgnmicOff
        End Select
    End Sub
```

## Example in C++:

To have your application react when a *PowerMic II* button that has been pressed is released, your code should include a **ButtonRelease()** method of the **IPowerMicIIEvents** object. First, you establish the events object, like the **CPowerMicSink** shown below:

```
CPowerMicSink::CPowerMicSink(void)
{
    m_pDlg = NULL;
}

CPowerMicSink::~CPowerMicSink(void)
{
    m_pDlg = NULL;
    TRACE( _T("CPowerMicSink::~CPowerMicSink() -> object destroyed!\n") );
}
```

Second, you create the method of the events object, having it receive the **btncode** as an argument, a **PMII\_MICBUTTONS** constant from the table above that indicates which button was released. The **ButtonPress()** method would test the **btncode** to determine the particular button that has been released, then take appropriate action.

The code for the method might have a case for each possible button that could be released on the microphone. For example, in the code below, the dialog (**m\_pDlg**) displays a check box that corresponds to each button that could have been pressed on the microphone. The check box that corresponds to the button just released is set to unchecked (**FALSE**):

```
STDMETHODIMP CPowerMicSink::ButtonRelease( PMII_MICBUTTONS btncode )
{
    ATLTRACE( "Button release codebutton = %x\n", btncode );
    switch( btncode )
    {
        case BTN_TRANSCRIBE:
            m_pDlg->m_bTranscribe = FALSE;
            break;
        case BTN_TABBACKWARD:
            m_pDlg->m_bTabBackward = FALSE;
            break;
        case BTN_DICTATE:
            m_pDlg->m_bDictate = FALSE;
            break;
        case BTN_TABFORWARD:
            m_pDlg->m_bTabForward = FALSE;
            break;
        case BTN_REWIND:
            m_pDlg->m_bRewind = FALSE;
            break;
        case BTN_FASTFORWARD:
            m_pDlg->m_bFastForward = FALSE;
    }
}
```

```
        break;
    case BTN_STOPPLAY:
        m_pDlg->m_bStopPlay = FALSE;
        break;
    case BTN_CUSTOMLEFT:
        m_pDlg->m_bCustomLeft = FALSE;
        break;
    case BTN_ENTERSELECT:
        m_pDlg->m_bEnterSelect = FALSE;
        break;
    case BTN_CUSTOMRIGHT:
        m_pDlg->m_bCustomRight = FALSE;
        break;
    case BTN_SCAN:
        m_pDlg->m_bScan = FALSE;
        break;
    }
    m_pDlg->UpdateData( FALSE );

    return S_OK;
}
```

### **Returned Values:**

None.

# Connected()

## Purpose:

Method of the **IPowerMicEvents** object. The framework calls this method when a *PowerMic II* has been connected to the computer running the application.

## Prototype:

HRESULT Connected();

## Parameters:

None.

## Example in C++:

To have your application react when a *PowerMic II* microphone is plugged in to the computer running your application, your code should include a **Connected()** method of the **IPowerMicEvents** object where it takes appropriate action:

```
STDMETHODIMP CPowerMicSink::Connected()  
{  
    return S_OK;  
}
```

## Returned Values:

None.

# Removed()

## Purpose:

Method of the **IPowerMicIIEvents** object. The framework calls this method when the *PowerMic II* has been disconnected from the computer running the application.

## Prototype:

HRESULT Removed();

## Parameters:

None.

## Example in C++:

To have your application react when a *PowerMic II* microphone is unplugged from the computer running your application, your code should include a **Removed()** method of the **IPowerMicIIEvents** object where it takes appropriate action:

```
STDMETHODIMP CPowerMicSink::Removed()  
{  
    return S_OK;  
}
```

## Returned Values:

None.

# ScanStarted()

## Purpose:

Method of the **IPowerMicIIEvents** object. The framework calls this method when the scan has been started using the *PowerMic II* microphone.

## Prototype:

```
HRESULT ScanStarted(void);
```

## Parameters:

None.

## Example in C++:

To have your application react when a *PowerMic II* microphone starts scanning a bar code, your application should include a **ScanStarted()** method of the **IPowerMicIIEvents** object where it takes appropriate action:

```
STDMETHODIMP CPowerMicSink::ScanStarted()  
{  
    ATLTRACE( "##### CPowerMicSink::ScanStarted\n" );  
    return S_OK;  
}
```

## Returned Values:

None.

# ScanFinished()

## Purpose:

Method of the **IPowerMicIIEvents** object. The framework calls this method when the *PowerMic II* has finished scanning a bar code or scannable text.

## Prototype:

```
HRESULT ScanFinished([in] BSTR pbstrData, [in] BOOL pbSuccess);
```

## Parameters:

### pbstrData

String containing the data scanned.

### pbSuccess

**True** if the scan completed successfully, **False** if it did not.

## Example in C++:

To have your application react when a *PowerMic II* microphone finishes scanning a bar code, your application should include a **ScanStarted()** method of the **IPowerMicIIEvents** object where it takes appropriate action:

```
STDMETHODIMP CPowerMicSink::ScanFinished(BSTR bstrData, long bSuccess)
{
    USES_CONVERSION;
    ATLTRACE( "##### result = %s, %d", W2T(bstrData), bSuccess );
    CString str;

    if(bSuccess)
    {
        m_pDlg->m_EditScanData.SetWindowText( W2T(bstrData) );
    }
    else
    {
        m_pDlg->m_EditScanData.SetWindowText( _T("Scan timeout") );
    }
    return S_OK;
}
```

## Returned Values:

None.


















# *Values for PowerMic II Microphone Buttons in Microphone SDK API*

This appendix shows information for working with the buttons on the *PowerMic II* microphone in the *Microphone SDK API*, indicating:

- Name used to refer to each button
- Default function(s) of each button
- *Microphone SDK API* method that is equivalent to pressing the button
- *PMII\_MICBUTTONS* value received by the **ButtonPress()** event handler method when the button is pressed and by the **ButtonRelease()** event handler method when the button is released

## PowerMic II Buttons in Microphone SDK



btncode Values for ButtonPress(), ButtonRelease()		
Microphone Button	btncode PMII_MICBUTTONS Value	Function
<b>MICROPHONE</b> 	Not applicable.	Talk into the PowerMic II microphone.
<b>TRANSCRIBE</b> 	<b>BTN_TRANSCRIBE</b>	Press to end current dictation/display transcribed text. Press to position insertion point or select a sentence.
<b>Dictate</b> 	<b>BTN_DICTATE</b>	Press to begin recording your report.
<b>TAB BACK or NAVIGATE BACKWARD</b> 	<b>BTN_TABBACKWARD</b>	Press to move to previous dialog box or field. After dictating, press to end dictation/display transcribed text.
<b>TAB OR NAVIGATE FORWARD</b> 	<b>BTN_TABFORWARD</b>	Press to move to next dialog box or field. Press to select text after transcribing.
<b>REW</b> 	<b>BTN_REWIND</b>	Press to rewind recorded audio. Press to scroll <i>down</i> through displayed list.
<b>FF</b> 	<b>BTN_FASTFORWARD</b>	Press to fast forward through recorded audio. Press to scroll <i>up</i> through displayed list.
<b>STOP/PLAY</b> 	<b>BTN_STOPPLAY</b>	Press to start or stop audio playback.
<b>ENTER/SELECT Customizable ENTER Button</b> 	<b>BTN_ENTERSELECT</b>	Press to take the custom programmed action.
<b>Customizable LEFT (Left of ENTER)</b> 	<b>BTN_CUSTOMLEFT</b>	Press to take the custom programmed action.
<b>Customizable RIGHT (Right of ENTER)</b> 	<b>BTN_CUSTOMRIGHT</b>	Press to take the custom programmed action.
<b>LEFT/RIGHT MOUSE BUTTONS</b> 	Not applicable.	Use the same way you use your mouse buttons.
<b>POINTING DEVICE</b> 	Not applicable.	Use to position the insertion point and select objects, just as you would with the mouse.
<b>SCAN (if no scanner, Customizable Scan Button)</b> 	<b>BTN_SCAN</b>	On mic with scanner, scans MRN or accession number barcode. On mic without scanner, takes programmed action.
<b>TRIGGER (on back of Microphone)</b> 	Not applicable.	Use the same way you use the left mouse button.

## Dictation Tips & Techniques

- Dictation Preparation — Collect your thoughts and plan your words before speaking.
  - Position the microphone between 2 and 5 inches from your mouth, but not touching. Keep the microphone at a constant distance from your mouth.
  - At the start of your dictation, press the **DICTATE** button and pause briefly before you start speaking.
  - Turn the microphone off when you are not speaking or if you move away from it.
  - Speak naturally at your normal rate.
  - Speak at a normal rate, not too quickly or too slowly. You should not:  
T a a a l k s l o o o w w l y y ..... Or. Say. Only. One. Word. At. A. Time.
  - Speak at a normal, constant volume.
  - Speak as you would to someone sitting across the desk from you. Do not speak too loudly or too softly.
  - Enunciate properly — Fast dictation is acceptable as long as the words are spoken clearly and not slurred.
  - Word beginning and ending sounds are important for recognition accuracy.
  - Small words — Take more time when saying small words. Don't run the words together. For example, "and there was," "there is no," and similar small word phrases.
  - Small words — Add vocal inflection to increase or emphasize where the words are important.
  - Pause slightly before and after small words such as *a* and *the* if they are being lost or misrecognized.
  - When pronouncing the word *a*, pronounce it as in *say*, rather than as in *uh*.
  - Dictate punctuation.
  - Speak in cadence of 6-8 word phrases followed by a brief pause.
  - Stop dictating when engaging in side conversations or when there are excessive background noises. You must be the primary speaker.
  - Dictate in a quiet area to minimize background noise as much as possible. Stay away from machines, radios, fans, and crowds.
  - Avoid clearing your throat and yawning while you are dictating. Do not talk through a yawn or when you clear your throat. Please stop dictating.
  - Eliminate utterances (*ums*, *ahs*, coughing) and similar sounds.
  - Do not allow the tone or volume of your voice to trail off or fluctuate at the end of sentences or around small words.
  - Do not over-enunciate or elongate spoken words.
  - Do not speak too loudly or softly.
  - Do not pause in the middle of a word.
  - Do not chew gum or eat while dictating.
  - Do not repeat large segments of text and verbal instructions to the editor.
  - Do not use the microphone to point at films or other objects.
-



# *Configuration for Microphone Sharing*

This appendix contains information about the configuration file that is needed for applications to share the microphone.

## Configuration for Microphone Sharing

In order for applications to share the Microphone and have access to the generated events, you must create a configuration file: **usbmgr.conf**. The configuration file stores the list of applications which can receive the *PowerMic II* device events. This configuration file must reside in the same directory as **hiddev.dll**, **usbmgr.exe**, and **usbmgr.dll**. The configuration file has the following form:

```
<settings version="1.0">
  <applications default="appdefault.exe">
    <item path="application1"/>
    <item path="application2"/>
    <item path="applicationN"/>
  </applications>
</settings>
```

All applications should be included as the name of the executable file. The application name is not case-sensitive, must not be full qualified, and it must have an extension.