



Future Technology Devices International Ltd.

Application Note

AN_152

How To Detect The Connection And Removal Of USB Devices On A System

Document Reference No.: FT_000328

Version 1.1

Issue Date: 2010-09-20

**This application note illustrates how to detect USB device insertion
and removal Using Windows Application Programming Interface**

Future Technology Devices International Limited (FTDI)

Unit 1,2 Seaward Place, Glasgow G41 1HH, United Kingdom
Tel.: +44 (0) 141 429 2777 Fax: + 44 (0) 141 429 2758
E-Mail (Support): support1@ftdichip.com Web: <http://www.ftdichip.com>

Copyright © 2010 Future Technology Devices International Limited

Table of Contents

1	Introduction.....	2
2	How to Detect a USB Device Insertion and Removal.....	3
2.1	RegisterDeviceNotification Function.....	3
2.2	Device Globally Unique Identifier (GUID).....	4
2.3	Processing WM_DEVICECHANGE message.....	5
3	D2XXNotify Application.....	8
3.1	Running the D2XXNotify Application From the Executable.....	8
3.2	Message Map.....	12
4	Contact Information.....	13
	Appendix A - References.....	15
	Appendix B - List of Figures and Tables.....	16
	Appendix C - Revision History.....	17
	Revision Record Sheet.....	18

1 Introduction

This application note illustrates how to detect the insertion and removal of USB devices on a system by using an application example called "D2XXNotify" which is available as a free download from [FTDI website](http://www.ftdichip.com/Support/SoftwareExamples/CodeExamples/VC++.htm) at <http://www.ftdichip.com/Support/SoftwareExamples/CodeExamples/VC++.htm> . The application watches for a [WM_DEVICECHANGE](#) message which is activated, processed and sent to the window whenever a user adds/removes something from the computer e.g. plugging in a USB thumb drive, FTDI devices.

Any software code examples given in this document are for information only and are not supported by FTDI.

2 How to Detect a USB Device Insertion and Removal

Windows Operating System broadcasts basic notifications message to any application with a top-level window by processing a [WM_DEVICECHANGE](#). A top-level window is a window that has an independent existence under the window manager. It can be moved and resized independently.

The WM_DEVICECHANGE message notifies the application of a change to the hardware configuration of a device or the computer. If the application is not a top-level window or it is required to be notified on all other types of device changes, the application can use a Win32 Application Programming interface (API) function called [RegisterDeviceNotification](#) which registers the application with the system to receive device notification events. This function will be given a "handle", which is a pointer to a [DEV_BROADCAST_DEVICEINTERFACE](#) structure (contains all information about a class of device) and a flag which is [DEVICE_NOTIFY_WINDOW_HANDLE](#) or [DEVICE_NOTIFY_SERVICE_HANDLE](#). A Win32 API function [UnRegisterDeviceNotification](#) closes the specified device notification handle returned by RegisterDeviceNotification function. For further information please refer to the [Microsoft website](#).

2.1 RegisterDeviceNotification Function

The following C code example shows how the [RegisterDeviceNotification](#) function registers an application with the system to receive device notifications.

```
DEV_BROADCAST_DEVICEINTERFACE dbch;  
dbch.dbcc_size = sizeof(dbch);  
dbch.dbcc_devicetype = DBT_DEVTYP_DEVICEINTERFACE;  
for (int i = 0; i < sizeof(GuidInterfaceList); i++)  
{  
    dbch.dbcc_classguid = GuidInterfaceList[i];  
    dbch.dbcc_name[0] = '\\0';  
    NotificationHandle = RegisterDeviceNotification( GetSafeHwnd(), &dbch,  
    DEVICE_NOTIFY_WINDOW_HANDLE);  
}
```

The function takes a handle `GetSafeHwnd()`, a flag parameter value `DEVICE_NOTIFY_WINDOW_HANDLE` and a `DEV_BROADCAST_DEVICEINTERFACE` data structure. The `DEVICE_NOTIFY_WINDOW_HANDLE` specifies that the caller is a window and the `DEV_BROADCAST_DEVICEINTERFACE` data structure defines the size of this structure, device type [DBT_DEVTYP_DEVICEINTERFACE](#), the name of the devices and the interface device Globally Unique Identifier (GUID), `GuidInterfaceList` which are explained in section 2.2.

NOTE: Values of the `DEV_BROADCAST_DEVICEINTERFACE` data structure need to be edited to fit the application requirements. To use the above code, ensure that the [Dbt.h](#) header is included to an application. For further details please refer to [Microsoft website](#).

2.2 Device Globally Unique Identifier (GUID)

Plug and Play (PnP) devices are typically associated with two different GUIDs, a device interface GUID, and a device class GUID. A device class GUID defines a broad category of devices. When you look in the Windows Device Manager, it is ordered by the type of devices. Each of those devices is a device class and each of those classes is identified by a device class GUID. A device interface GUID specifies a particular input/output interface contract. Every instance of the device interface GUID is expected to support the same basic set of inputs/outputs. The device interface GUID is what the device driver will register and enable or disabled based on the PnP state. Refer to section 2.1, a “for loop” code which shows how to register multiple device notifications for each GUID.

The following code lists GUIDs for device interface classes.

```
static const GUID GuidInterfaceList[] =
{
    // USB Raw Device Interface Class GUID
    { 0xa5dcbf10, 0x6530, 0x11d2, { 0x90, 0x1f, 0x00, 0xc0, 0x4f, 0xb9, 0x51,
    0xed } },

    // Disk Device Interface Class GUID
    { 0x53f56307, 0xb6bf, 0x11d0, { 0x94, 0xf2, 0x00, 0xa0, 0xc9, 0x1e, 0xfb,
    0x8b } },

    //Human Interface Device Class GUID
    { 0x4d1e55b2, 0xf16f, 0x11cf, { 0x88, 0xcb, 0x00, 0x11, 0x11, 0x00, 0x00,
    0x30 } },

    // FTDI_D2XX_Device Class GUID
    { 0x219d0508, 0x57a8, 0x4ff5, { 0x97, 0xa1, 0xbd, 0x86, 0x58, 0x7c, 0x6c,
    0x7e } },

    // FTDI_VCP_Device Class GUID
    { 0x86e0d1e0L, 0x8089, 0x11d0, { 0x9c, 0xe4, 0x08, 0x00, 0x3e, 0x30, 0x1f,
    0x73 } },
};
```

2.3 Processing WM_DEVICECHANGE message

Whenever a device change (i.e. device insertion and removal) is detected and a [WM_DEVICECHANGE](#) message is sent to the application, the message structure will transport the events value that has occurred in the [WParam](#) of the message. The wParam can have various values from the [Dbt.h](#) header file. For example [DBT_DEVICEARRIVAL](#) and [DBT_DEVICEREMOVALCOMPLETE](#) indicate arrival and removal of device respectively. For further details please refer to [Microsoft website](#).

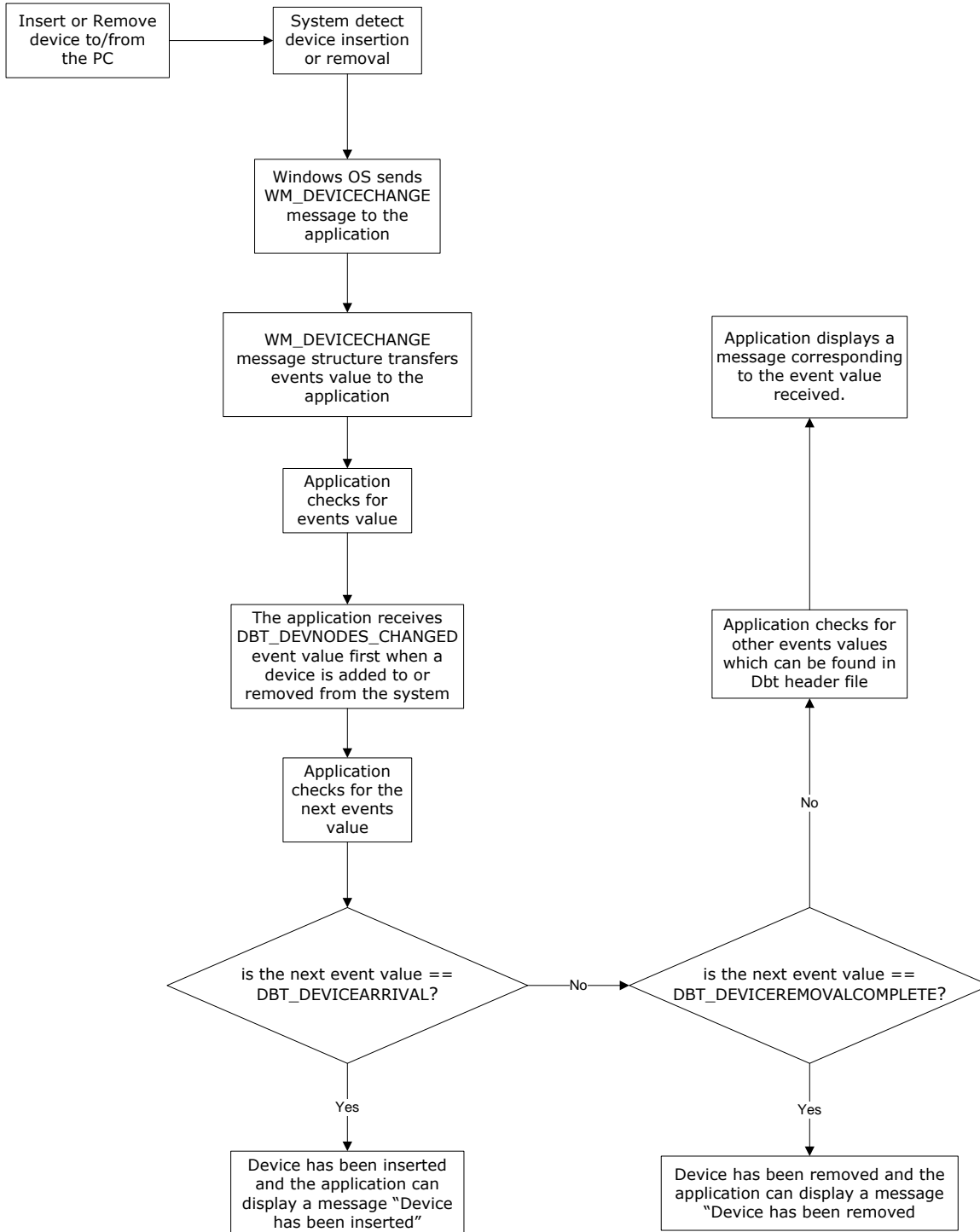


Figure 2.1 Processing WM_DeviceChange message

The sample code below shows how the WM_DEVICECHANGE message is processed when the application receives it:

```
BOOL CD2XXNotifyDlg::OnDeviceChange(UINT EventType, DWORD dwData)
{
    char Word [50];
    char Word1 [57];
    char Word2 [50];
    char Word3 [50];
    char ch = ' ';
    CString Msg = "duh";
    switch( EventType)
    {
    case DBT_CONFIGCHANGECANCELED:
        Msg.Format("DBT_CONFIGCHANGECANCELED");
        break;
    case DBT_CONFIGCHANGED:
        Msg.Format("DBT_CONFIGCHANGED");
        break;
    case DBT_CUSTOMEVENT:
        Msg.Format("DBT_CUSTOMEVENT");
        break;
    case DBT_DEVICEARRIVAL:
        Msg.Format("DBT_DEVICEARRIVAL");
        break;
    case DBT_DEVICEQUERYREMOVE:
        Msg.Format("DBT_DEVICEQUERYREMOVE");
        break;
    case DBT_DEVICEQUERYREMOVEFAILED:
        Msg.Format("DBT_DEVICEQUERYREMOVEFAILED");
        break;
    case DBT_DEVICEREMOVEPENDING:
        Msg.Format("DBT_DEVICEREMOVEPENDING");
        break;
    case DBT_DEVICEREMOVECOMPLETE:
        Msg.Format("DBT_DEVICEREMOVECOMPLETE");
        break;
    case DBT_DEVICETYPESPECIFIC:
        Msg.Format("DBT_DEVICETYPESPECIFIC");
        break;
    case DBT_QUERYCHANGECONFIG:
        Msg.Format("DBT_QUERYCHANGECONFIG");
        break;
    case DBT_DEVNODES_CHANGED:
        Msg.Format("DBT_DEVNODES_CHANGED");
        break;
    case DBT_USERDEFINED:
        Msg.Format("DBT_USERDEFINED");
        break;
    default:
        Msg.Format("Event type %d", EventType);
    }
}
```

```
PDEV_BROADCAST_DEVICEINTERFACE pdbch = (PDEV_BROADCAST_DEVICEINTERFACE)dwData;
if ( pdbch!=NULL && pdbch->dbcc_devicetype==DBT_DEVTYP_DEVICEINTERFACE)
{
    CString Msg2;
    Msg2.Format ("%s: %s",Msg,pdbch->dbcc_name);
    Msg = Msg2;
}
if (Msg == "DBT_DEVNODES_CHANGED")
{
    CListBox* EventList = (CListBox*)GetDlgItem(IDC_EVENT_LIST);
    EventList->AddString(Msg);
}
else
{
    strncpy(Word,Msg,17);
    Word[17] = '\0';
    if ( strcmp ( Word, "DBT_DEVICEARRIVAL" )== 0 )
    {
        strncpy(Word1,Msg,44);
        Word1[44] = '\0';
        strncpy(Word2, Word1, 17);
        Word2[17] = '\0';
        strncpy(Word3, &Word1[27], 17);
        Word3[17] = '\0';
    }
    else
    {
        strncpy(Word,Msg,24);
        Word[24] = '\0';
        if ( strcmp ( Word, "DBT_DEVICEREMOVECOMPLETE" )== 0 )
        {
            strncpy(Word1,Msg,51);
            Word1[51] = '\0';
            strncpy(Word2, Word1, 24);
            Word2[24] = '\0';
            strncpy(Word3, &Word1[34], 17);
            Word3[17] = '\0';
        }
        if (Word3[0]== 'V')
        {
            CListBox* EventList =
(CListBox*)GetDlgItem(IDC_EVENT_LIST);
            EventList->AddString(Word2);
            EventList->AddString(Word3);
        }
    }
}
return TRUE;
```

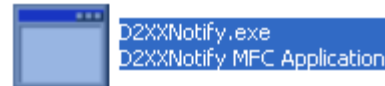

3 D2XXNotify Application

The "D2XXNotify" application provides a user interface to detect insertion or removal of USB devices on a computer. The application source code is available as a free download from [the FTDI website](#).

3.1 Running the D2XXNotify Application From the Executable

The executable file "D2XXNotify.exe" can be downloaded from the [FTDI website](#) in a compressed zip file called "D2XXNotify.zip" (this also contains the application source code).

Uncompress the file using an extraction tool. There are many available on the web such as Winzip or WinRAR.



Run the executable file by double clicking on "D2XXNotify.exe" icon.

and the screen as shown in Figure 3.1 should appear

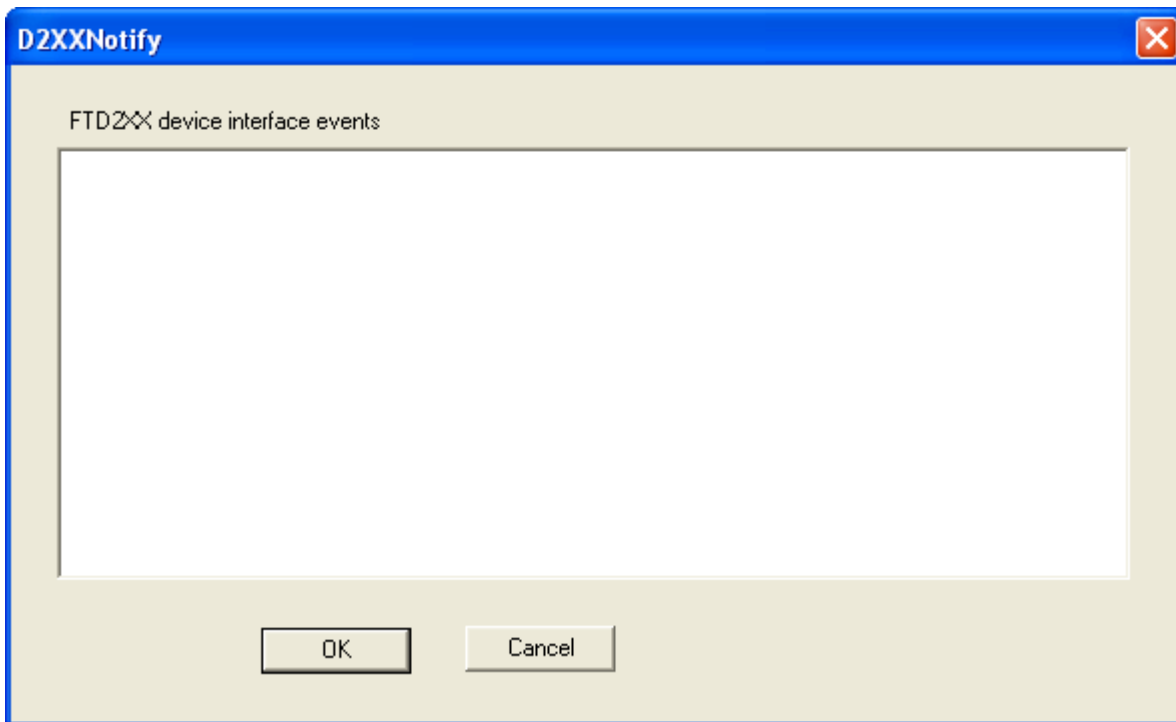


Figure 3.1 Run the Application from Executable File

When a USB device is inserted into a computer, a USB device interface event notification will be sent and the application receives the [WM_DEVICECHANGE](#) message. The application then calls the appropriate function specified in the Message Map (see section 3.2) and the screen appears as shown in Figure 3.2

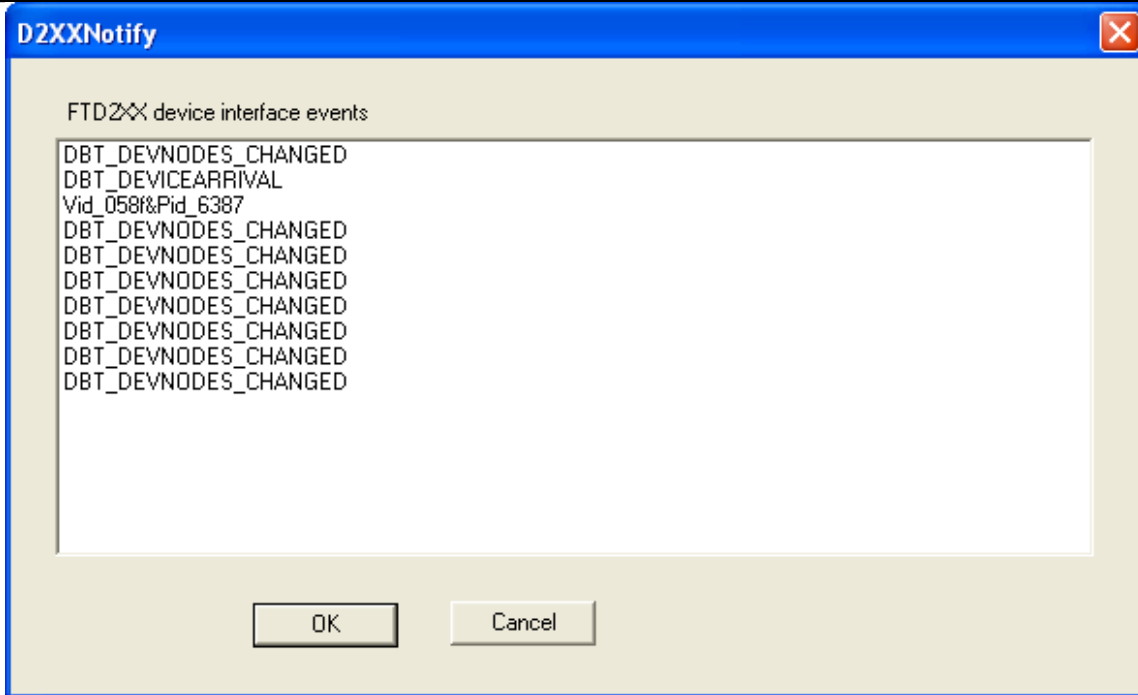


Figure 3.2 Application Receive DBT_DEVICEARRIVAL message

With reference to Figure 3.2, the "D2XXNotify" application displays the "DBT_DEVNODES_CHANGED", "DBT_DEVICEARRIVAL" messages and the PID and VID of the device when a device has been added to the system (the system sent the DBT_DEVNODES_CHANGED and DBT_DEVICEARRIVAL device events). When a USB device is removed from the computer, a USB device interface event notification will be sent and the application receives the [WM_DEVICECHANGE](#) message and the screen appears as shown in Figure 3.3

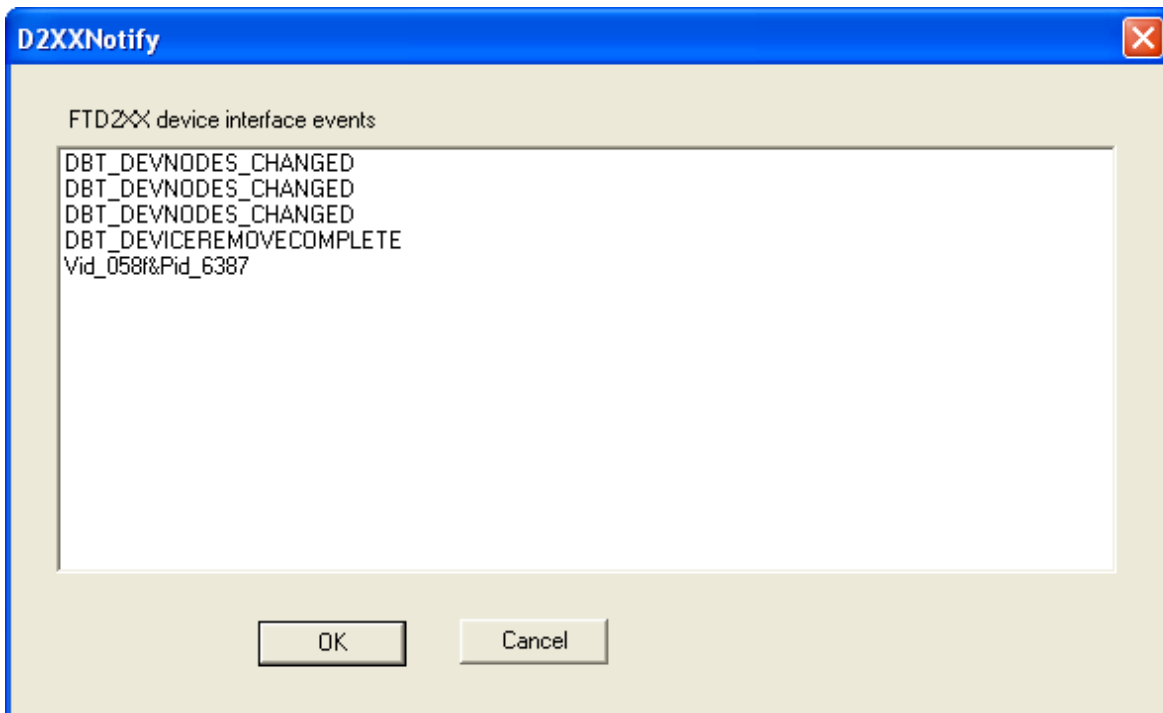



Figure 3.3 Application Receive DBT_DEVICECOMPLETE message

With reference to Figure 3.3, the "D2XXNotify" application displays the "DBT_DEVNODES_CHANGED", "DBT_DEVICEREMOVECOMPLETE" messages and the PID and VID of the device when a device has been removed from the system (the system sent the DBT_DEVNODES_CHANGED and DBT_DEVICEREMOVECOMPLETE device events).

Alternative Approach : Building and Running the D2XXNotify Application From Visual Studio
 Alternatively, open the extracted file "D2XXNotify" and open the project solution on Microsoft Visual

Studio by double clicking "d2xxnotify.sln" file  and the window will appear as shown in Figure 3.4

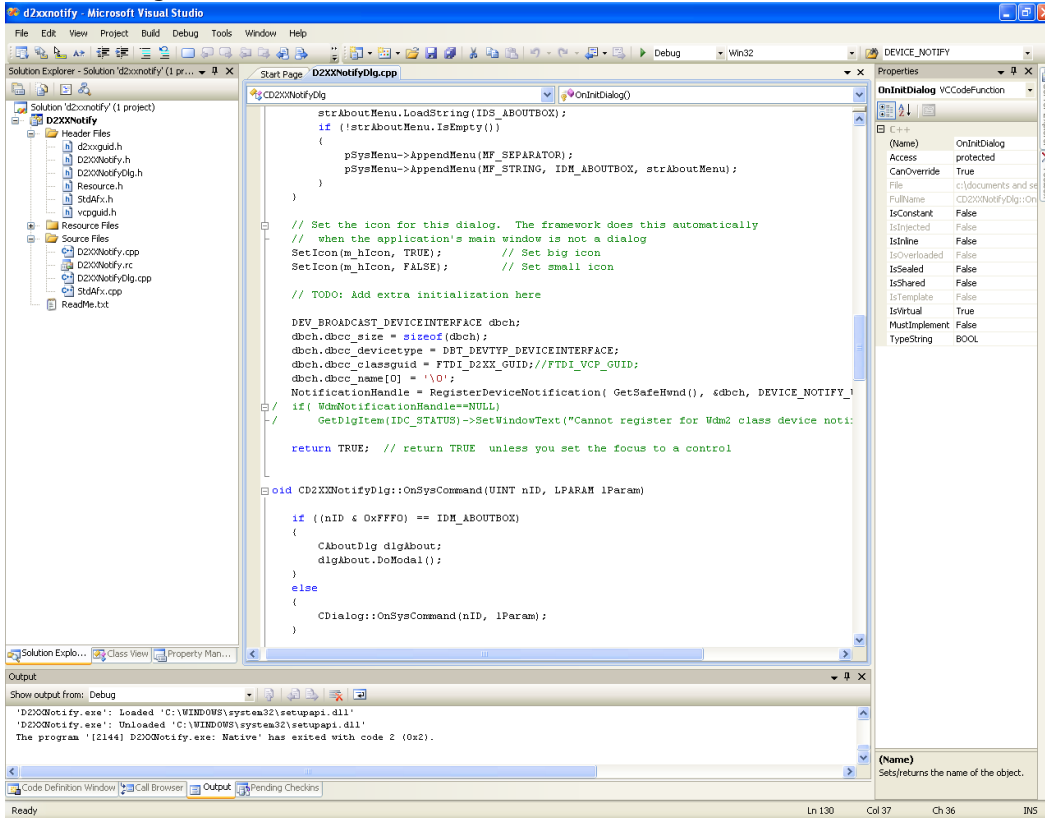


Figure 3.4 Open the Application in Visual Studio

Once an application is opened, it is necessary to build the application to delete files from a previous build. This build also creates a "D2XXNotify" executable. To build and clean the "D2XXNotify" application using Visual Studio, go to "Build" tab and click "Clean Solution" and "Rebuild Solution"

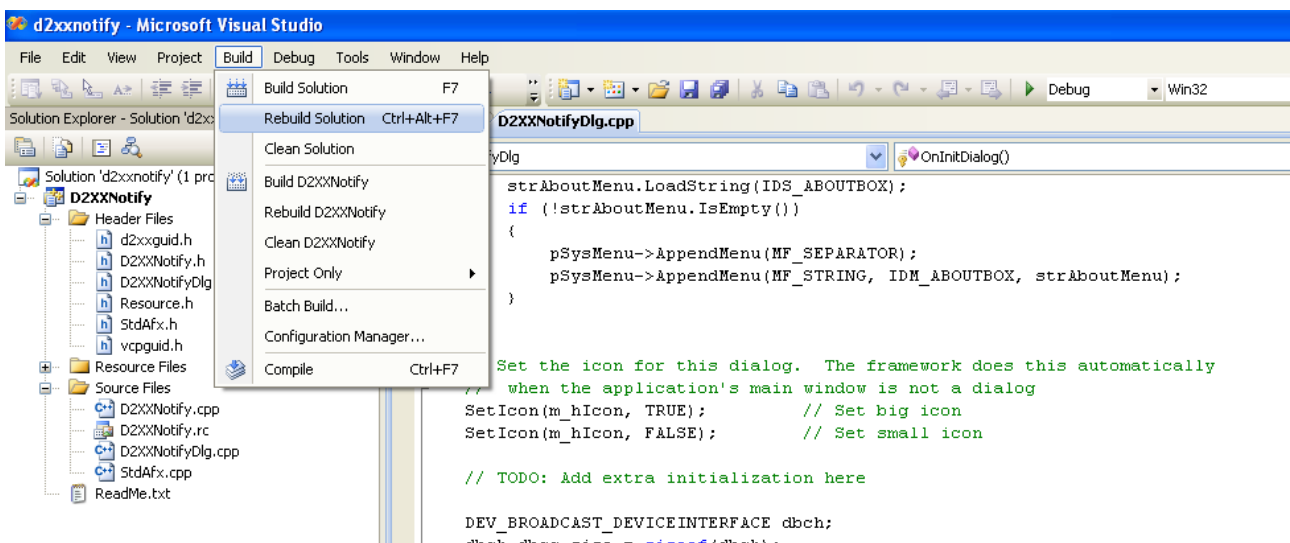


Figure 3.5 Build the Application

After clicking "Rebuild Solution" the IDE will compile the source code with no compilation errors if the source code hasn't been altered.

To run the application, go to "Debug" tab and click "Start Without Debugging"

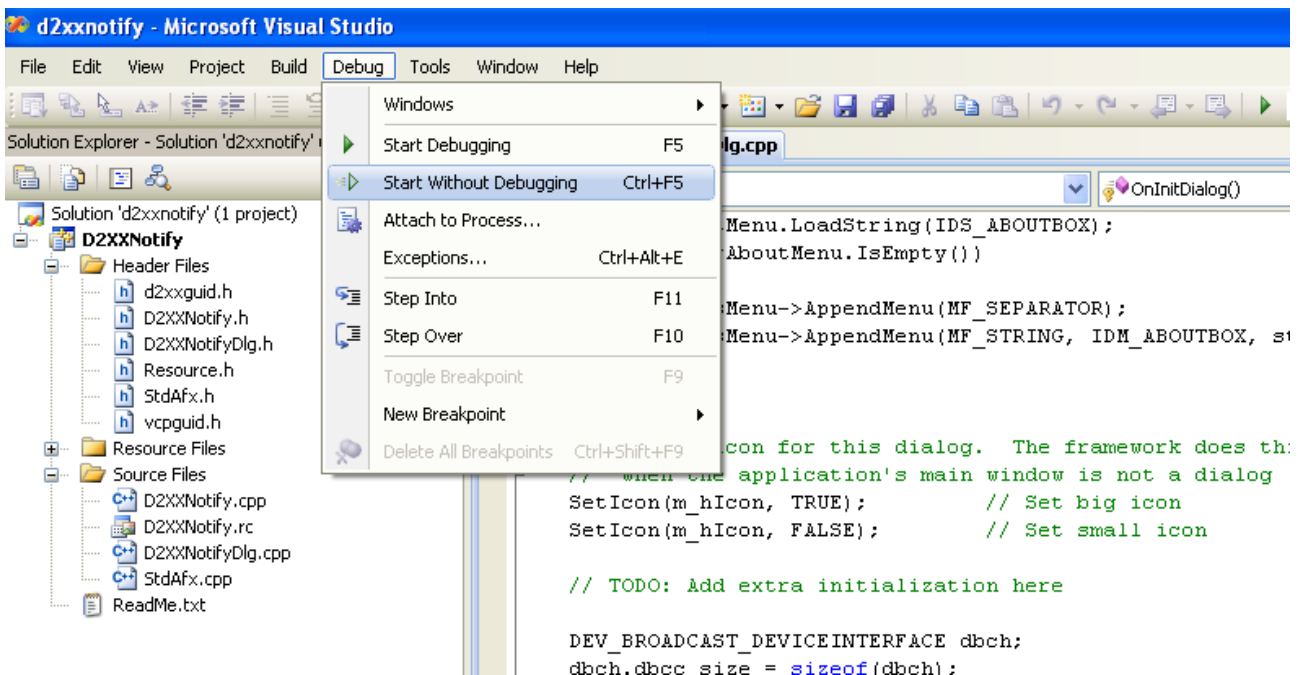


Figure 3.6 Run the Application from the Debug tab

3.2 Message Map

[Message Map](#) is the Microsoft Foundation Class (MFC) way of handling application messages. It is a table that correlates messages and member functions. When a window receives a message, MFC scans the window's message map to determine if the handler exists for that message and specifies which message needs to be called. The following C code example illustrates how the message map can be used within the application.

```
class CAboutDlg : public CDialog
{
    void CD2XXNotifyDlg::OnPaint();
    BOOL CD2XXNotifyDlg::OnDeviceChange(UINT EventType, DWORD dwData);

    protected:
    DECLARE_MESSAGE_MAP()
};

void CAboutDlg::DoDataExchange(CDataExchange* pDX){}

BEGIN_MESSAGE_MAP(CD2XXNotifyDlg, CDialog)
ON_WM_PAINT()
ON_WM_DEVICECHANGE()
END_MESSAGE_MAP()

void CD2XXNotifyDlg::OnPaint()
{}
BOOL CD2XXNotifyDlg::OnDeviceChange(UINT EventType, DWORD dwData)
{}
```

When an application receives a message, the windows message map will be scanned to see if the handler for WM_PAINT() and WM_DEVICECHANGE() exist and call the OnPaint and OnDeviceChange functions.

[DECLARE_MESSAGE_MAP](#) tells the application that the class in which this is called is going to have a message map and handle messages. A class can only have one message map.

[BEGIN_MESSAGE_MAP](#) begins the definition of the message map by specifying the name of the class whose message map this is and the name of the base class.

[END_MESSAGE_MAP](#) ends the definition of the message map.

To use the above example code ensure that the MFC Library is included into the application. For more information on message maps see [Message Handling and Mapping Topics](#).

4 Contact Information

Head Office – Glasgow, UK

Future Technology Devices International Limited
Unit 1, 2 Seaward Place, Centurion Business Park
Glasgow G41 1HH
United Kingdom
Tel: +44 (0) 141 429 2777
Fax: +44 (0) 141 429 2758

E-mail (Sales) sales1@ftdichip.com
E-mail (Support) support1@ftdichip.com
E-mail (General Enquiries) admin1@ftdichip.com
Web Site URL <http://www.ftdichip.com>
Web Shop URL <http://www.ftdichip.com>

Branch Office – Taipei, Taiwan

Future Technology Devices International Limited (Taiwan)
2F, No. 516, Sec. 1, NeiHu Road
Taipei 114
Taiwan, R.O.C.
Tel: +886 (0) 2 8791 3570
Fax: +886 (0) 2 8791 3576

E-mail (Sales) tw.sales1@ftdichip.com
E-mail (Support) tw.support1@ftdichip.com
E-mail (General Enquiries) tw.admin1@ftdichip.com
Web Site URL <http://www.ftdichip.com>

Branch Office – Hillsboro, Oregon, USA

Future Technology Devices International Limited (USA)
7235 NW Evergreen Parkway, Suite 600
Hillsboro, OR 97123-5803
USA
Tel: +1 (503) 547 0988
Fax: +1 (503) 547 0987

E-Mail (Sales) us.sales@ftdichip.com
E-Mail (Support) us.support@ftdichip.com
E-Mail (General Enquiries) us.admin@ftdichip.com
Web Site URL <http://www.ftdichip.com>

Branch Office – Shanghai, China

Future Technology Devices International Limited (China)
Room 408, 317 Xianxia Road,
Shanghai, 200051
China
Tel: +86 21 62351596
Fax: +86 21 62351595

E-mail (Sales) cn.sales@ftdichip.com
E-mail (Support) cn.support@ftdichip.com
E-mail (General Enquiries) cn.admin@ftdichip.com
Web Site URL <http://www.ftdichip.com>

Distributor and Sales Representatives

Please visit the Sales Network page of the FTDI Web site for the contact details of our distributor(s) and sales representative(s) in your country.



Vinculum is part of Future Technology Devices International Ltd. Neither the whole nor any part of the information contained in, or the product described in this manual, may be adapted or reproduced in any material or electronic form without the prior written consent of the copyright holder. This product and its documentation are supplied on an as-is basis and no warranty as to their suitability for any particular purpose is either made or implied. Future Technology Devices International Ltd will not accept any claim for damages howsoever arising as a result of use or failure of this product. Your statutory rights are not affected. This product or any variant of it is not intended for use in any medical appliance, device or system in which the failure of the product might reasonably be expected to result in personal injury. This document provides preliminary information that may be subject to change without notice. No freedom to use patents or other intellectual property rights is implied by the publication of this document. Future Technology Devices International Ltd, Unit 1, 2 Seaward Place, Centurion Business Park, Glasgow G41 1HH United Kingdom. Scotland Registered Number: SC136640

Appendix A - References

<http://msdn.microsoft.com/en-us/library/aa363480.aspx>

<http://blogs.msdn.com/b/doronh/archive/2006/02/15/532679.aspx>

<http://www.codemiles.com/download/file.php?id=719>

<http://aclacl.brinkster.net/MFC/ch01d.htm>

<http://infohost.nmt.edu/tcc/help/pubs/tkinter/toplevel.html>

<http://www.codersource.net/mfc/mfc-tutorials/mfc-handling-messagemaps.aspx>

Appendix B - List of Figures and Tables

List of Figures

Figure 2.1 Processing WM_DeviceChange message	5
Figure 3.1 Run the Application from Executable File	8
Figure 3.2 Application Receive DBT_DEVICEARRIVAL message	9
Figure 3.3 Application Receive DBT_DEVICECOMPLETE message	9
Figure 3.5 Build the Application	10
Figure 3.6 Run the Application from the Debug tab	11

Appendix C - Revision History

Version 1.0	Initial release	31 st August 2010
Version 1.1	Added link to the application example on the FTDI website	20 th September 2010