D2xx WinRT Guide

AN_271

Document Reference No.: FT_000928

Version 1.0

Issue Date: 2013-06-30

This document provides a getting started guide to the WinRT D2xx driver for Windows Store Apps.

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

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

Table of Contents

1 Introduction 2
1.1 System Requirements 2
2 Installing WinUSB Driver 3
2.1 Windows 8.1 3
2.1.1 Disable Windows Update 3
2.1.2 Install Driver 4
2.2 Windows 8.1 RT 6
3 Windows Store App 11
3.1 Add Reference 11
3.2 Manifest 12
3.3 Vendor ID, Product ID 13
3.4 List Devices 14
3.5 Open Device 14
3.6 EEPROM 15
4 Contact Information 17
5 Appendix A – References 18
Document References 18
Acronyms and Abbreviations 18
6 Appendix B – List of Tables & Figures 19
7 Appendix C - Code Listing 20
7.1 Package.appxmanifest 20
7.2 MainPage.xaml.cs 21
7.3 MainPage.xaml 24
8 Appendix D – Revision History 25

1 Introduction

D2xx WinRT is a device driver for FTDI's range of USB converter chips. It is designed to be used with the Windows 8.1 and Windows 8.1 RT platforms, and is based on the WinUSB generic device driver. The driver allows access to FTDI products from within a Windows Store App – this is not possible with the standard virtual COM port or D2xx driver model.

The driver is provided as a Windows Runtime Component (.winmd file) which is consumed by the host Windows Store App. FTDI devices can be accessed from any Store App irrespective of the programming language used to develop the app (C#, VB, JavaScript or C++).

Figure 1.1 illustrates the architecture of a typical Store App using the driver. The boxes in blue are the components that comprise the driver and are provided by FTDI. The boxes in green are the Microsoft layers, namely: the Windows Runtime API layer; and the WinUSB generic driver that the FTDI driver communicates with. Below this is the FTDI hardware layer. The top most layer would be the Store App written by the developer.

The API for the WinRT driver is a proprietary D2xx style interface. Microsoft does not allow access to a traditional serial port from within the Windows Runtime environment.

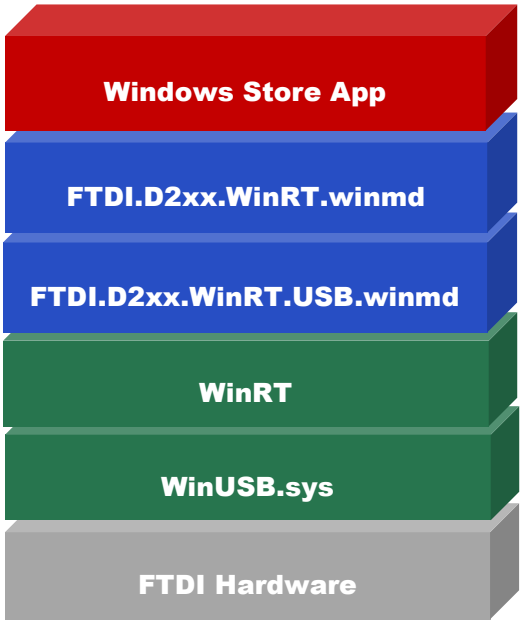

Figure 1 – Windows Store App Driver Architecture

This document provides a getting started guide for developers wishing to use the driver.

1.1 System Requirements

Minimum Supported Operating System: Windows 8.1, Windows 8.1 RT.

Minimum Supported Visual Studio Edition: Visual Studio 2013.

2 Installing WinUSB Driver

The D2xx WinRT driver requires the generic Windows WinUSB device driver (winusb.sys) to be installed for the FTDI device. The steps below discuss the installation process for the Windows 8.1 and the Windows 8.1 RT (ARM) platforms.

2.1 Windows 8.1

FTDI distribute an .inf file that can be used to install the WinUSB driver for FTDI's list of standard VID and PID combinations.

Note: *A .cat file which has been digitally signed with FTDI's certificate accompanies the .inf file; without a digital signature Windows may not allow the driver to be installed. If a custom VID and/or PID are required, it is the responsibility of the user to provide an appropriate, digitally signed .cat file and .inf file.*

If Windows Update is enabled on the target machine, and there is an internet connection, the OS will automatically install the Windows CDM driver for any new FTDI devices that are connected to the system. To use the WinRT driver this feature must be disabled and the WinUSB.sys driver installed instead.

2.1.1 Disable Windows Update

Open System Properties.

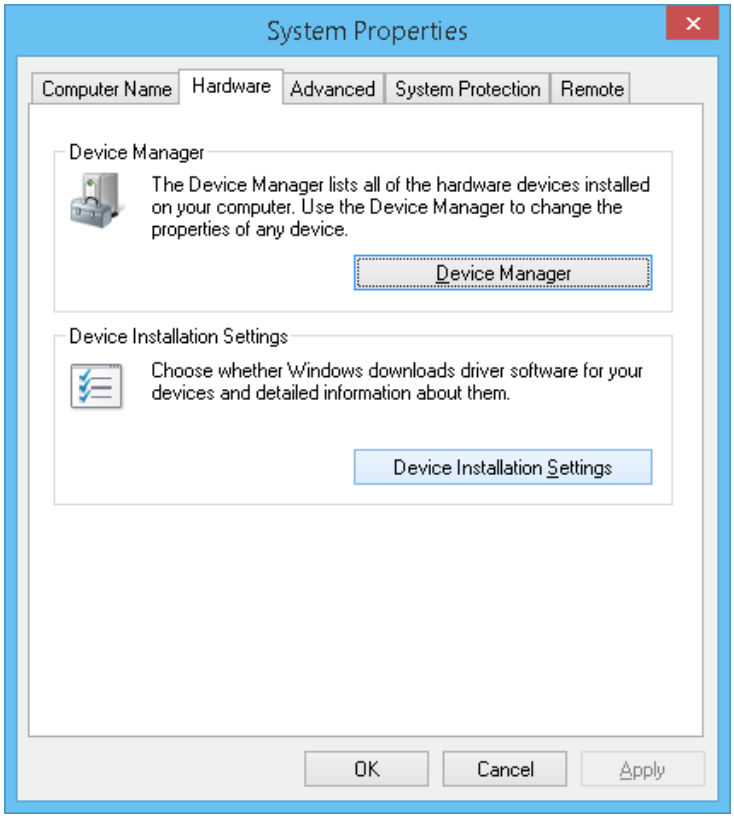

Figure 2 - System Properties

On the Hardware tab select 'Device Installation Settings'.

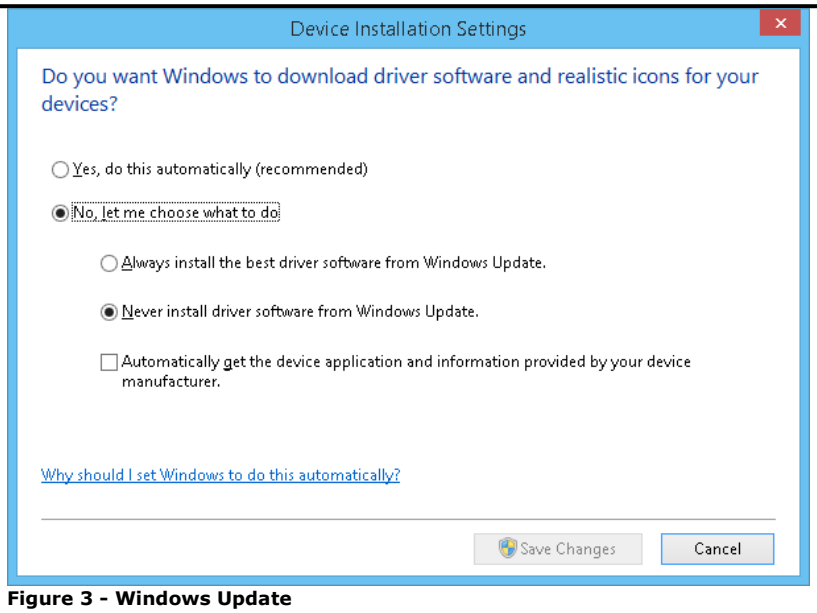

Figure 3 - Windows Update

Turn off Windows Update by selecting 'No, let me choose what to do' and 'Never install driver software from Windows Update'.

2.1.2 Install Driver

Note: *If the device already has the CDM driver installed, this must be removed first before proceeding. To remove the driver, right click on the device within Device Manager and select Uninstall. Continue by pressing the 'Scan for hardware changes' button; the device will now appear under the other devices category as in* Figure 4*.*

To install the WinUSB driver right click the device within Device Manager and select 'Update Driver Software…'.

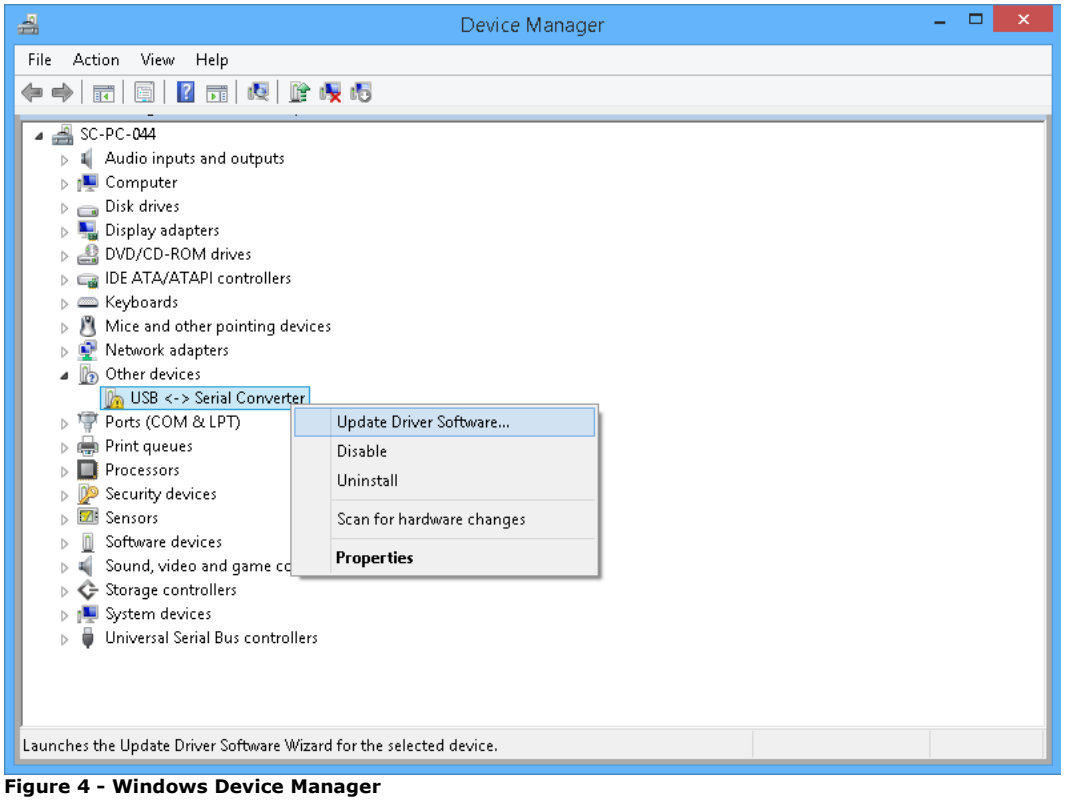

Figure 4 - Windows Device Manager

Select 'Browse my computer for driver software'.

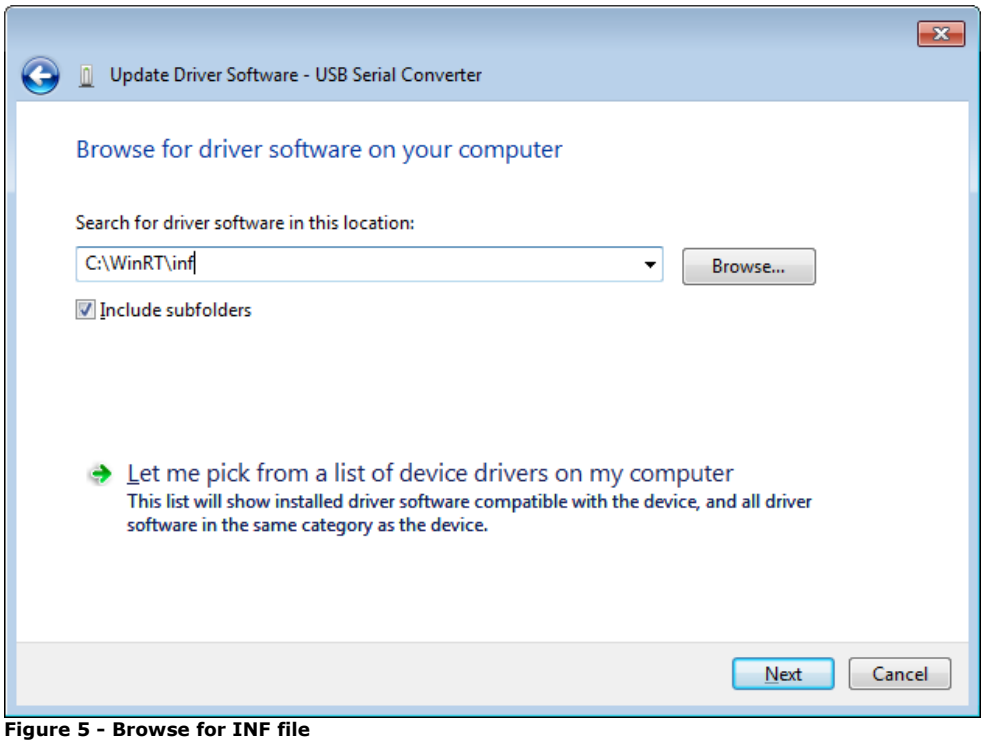

Figure 5 - Browse for INF file

In the textbox insert the path to the .inf file provided by FTDI. Hit Next to install the driver.

The WinUSB driver is now installed and the device will appear in the 'Universal Serial Bus devices' section as below.

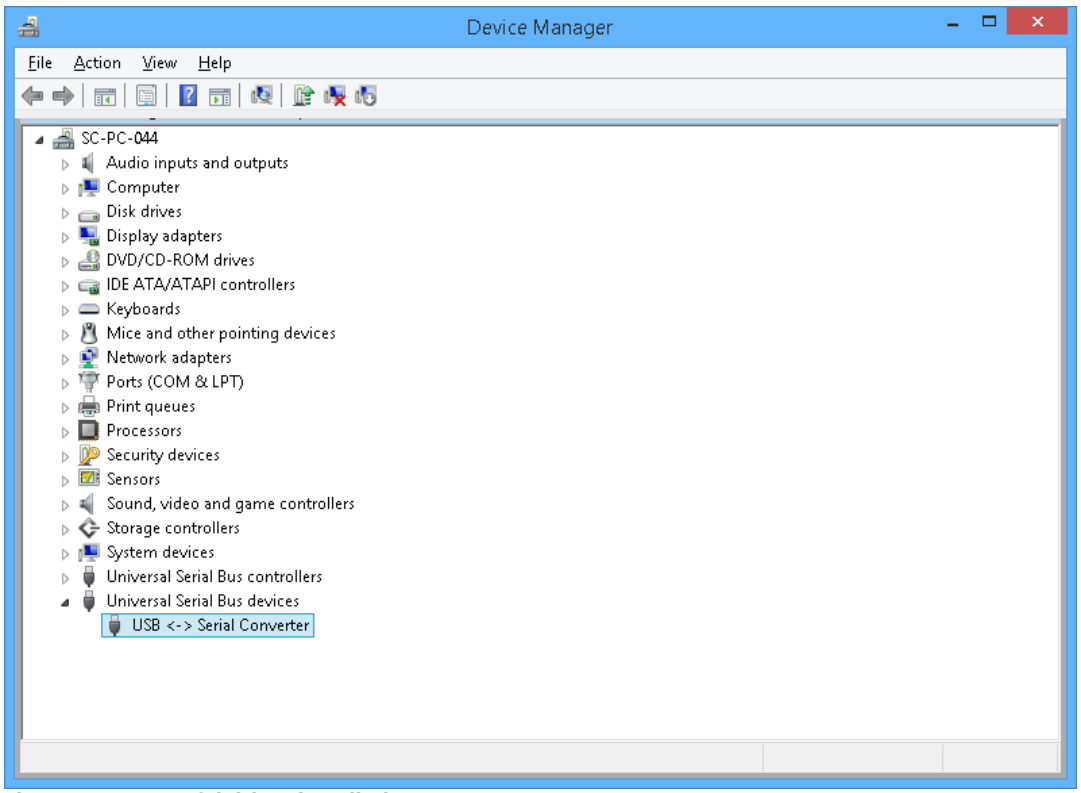

Figure 6 - Successful driver installation

2.2 Windows 8.1 RT

Insert the FTDI device into a USB port on the RT machine and open Device Manager. The device will have a small exclamation mark next to it to indicate that the OS has not installed a valid driver for the device.

Right click on the device and select 'Update Driver Software…'

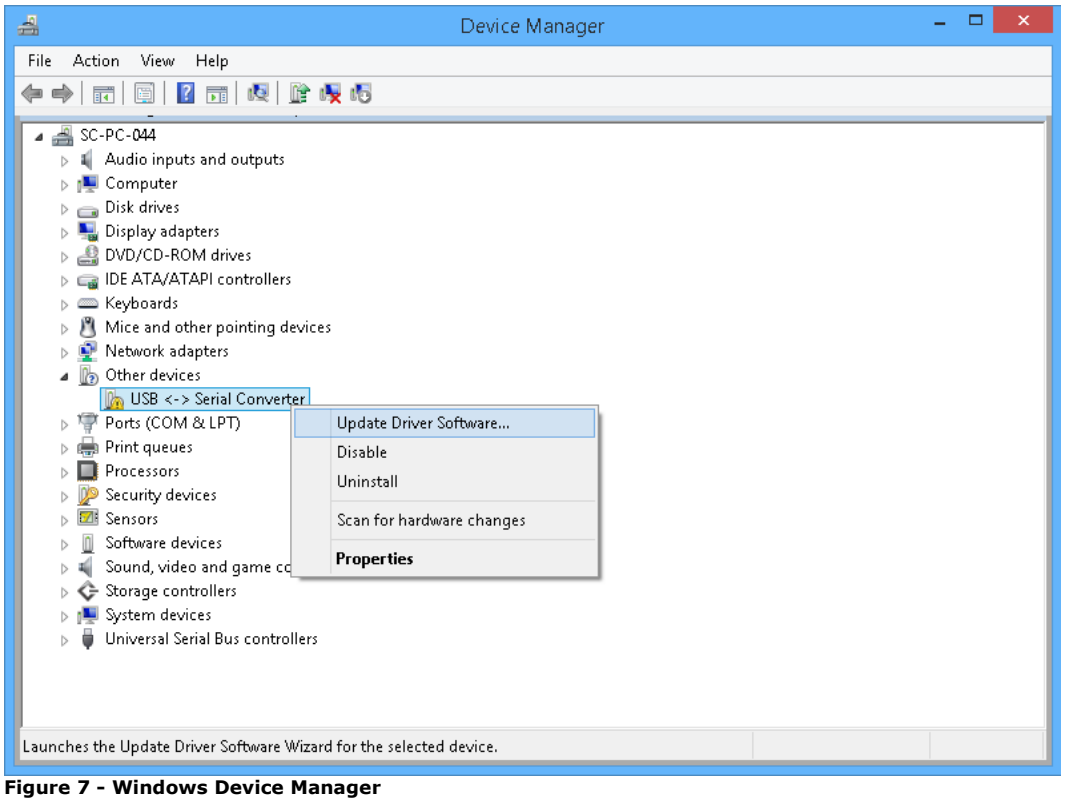

Figure 7 - Windows Device Manager

Select 'Browse my computer for driver software' in the next window.

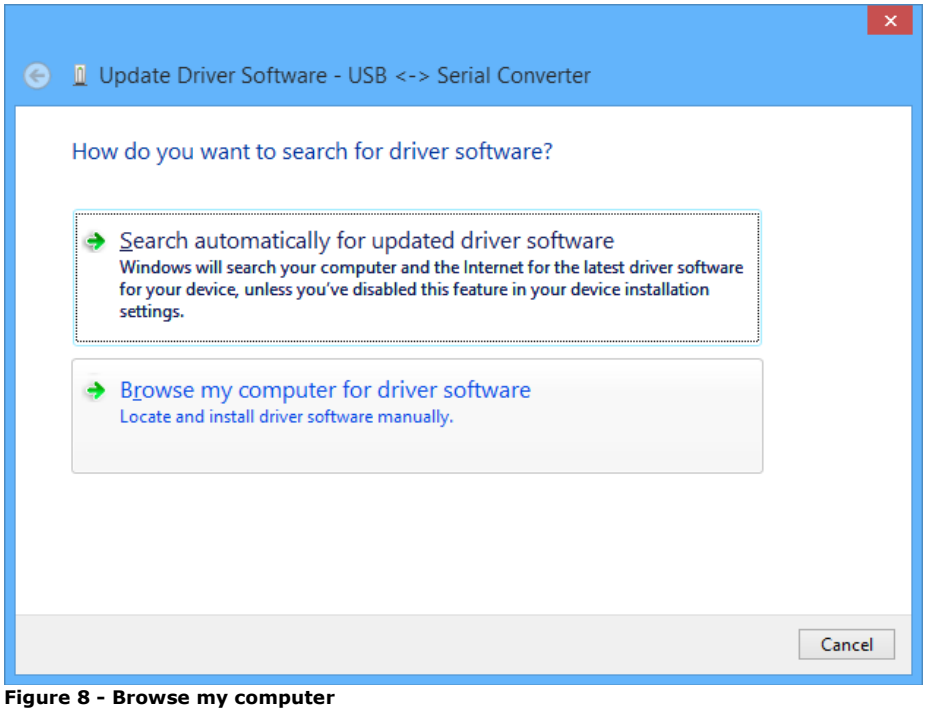

Figure 8 - Browse my computer

On the next screen click 'Let me pick from a list of device drivers on my computer'.

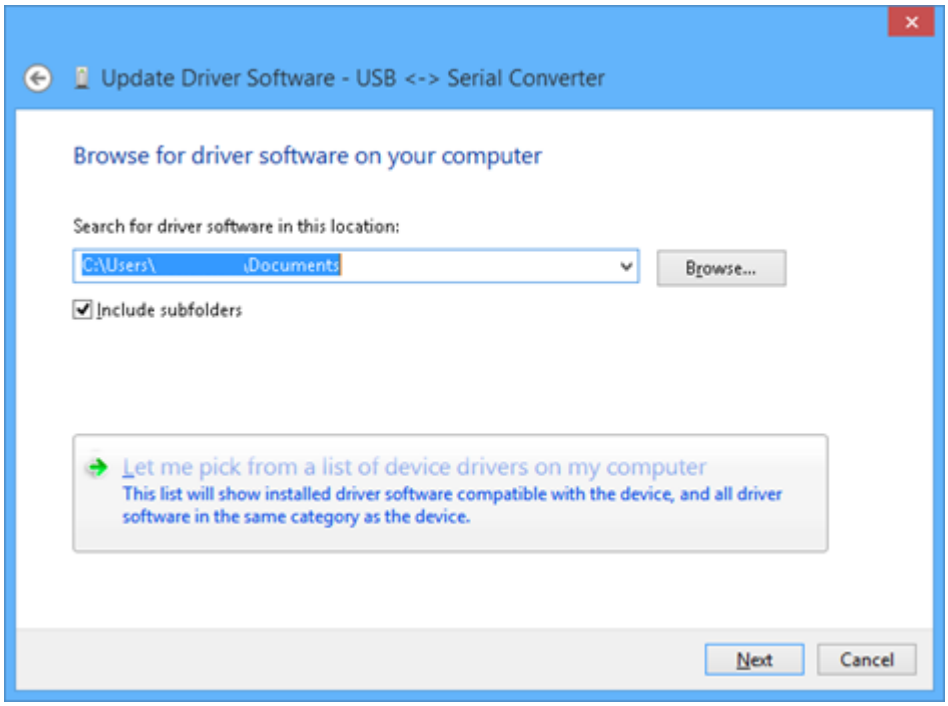

Figure 9 - Let me pick from a list of device drivers

In the next screen, scroll to the bottom of the list of device drivers. Select the 'Universal Serial Bus devices' option and click Next.

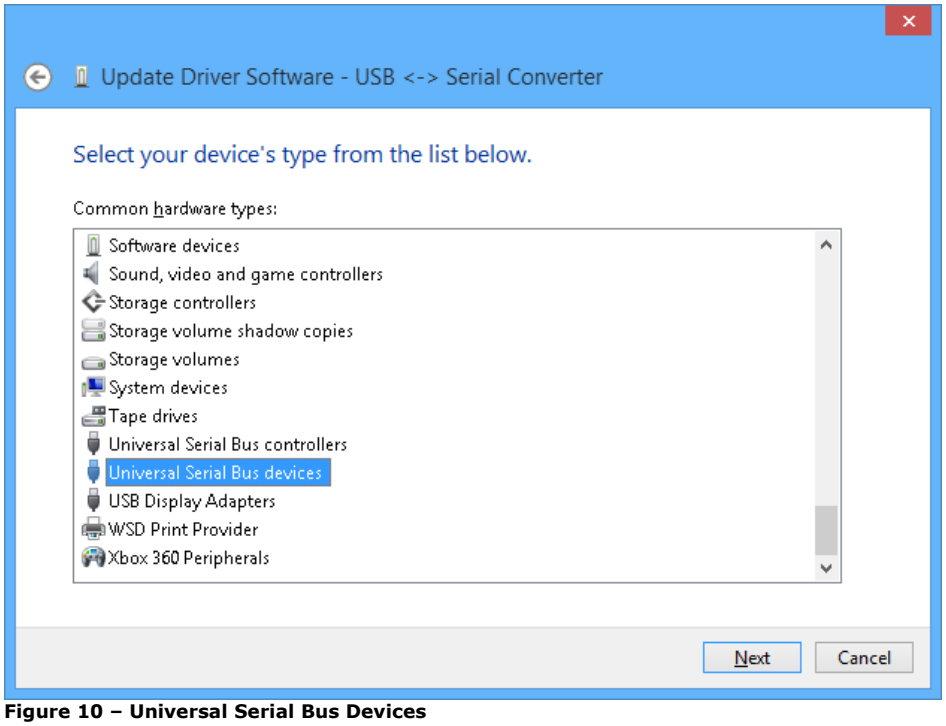

Figure 10 – Universal Serial Bus Devices

Select 'WinUSB Device' and click Next.

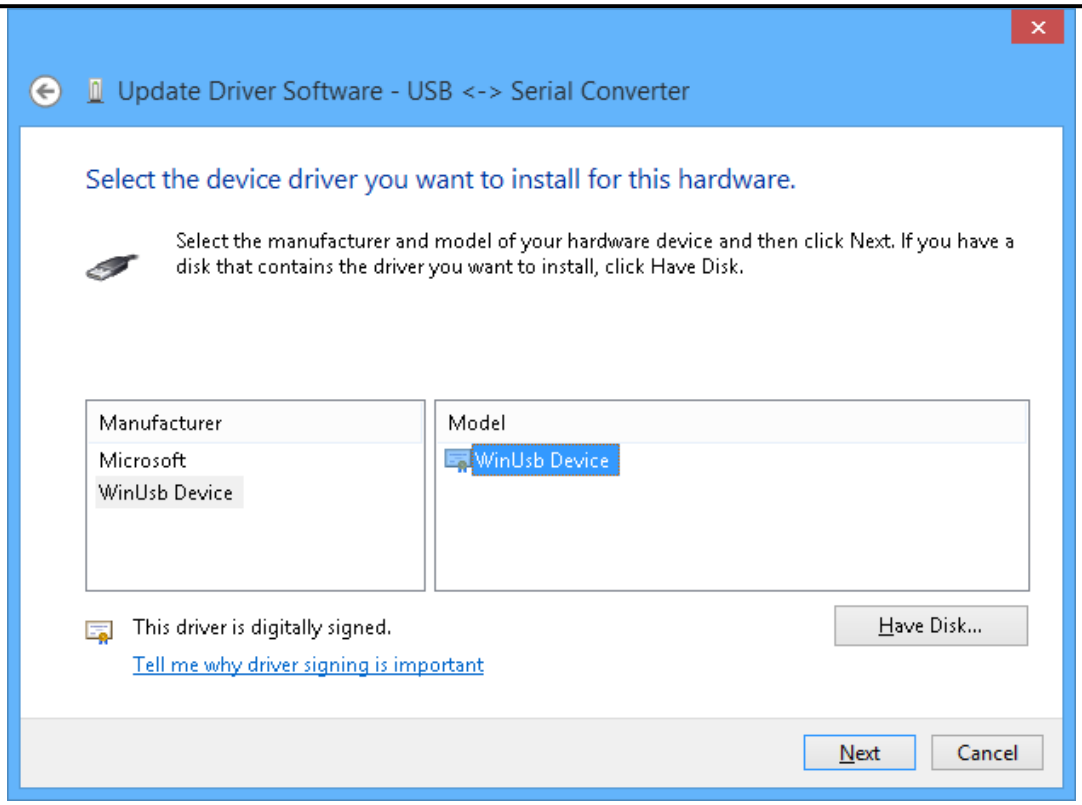

Figure 11 - WinUSB driver

Click 'Yes' to the warning window.

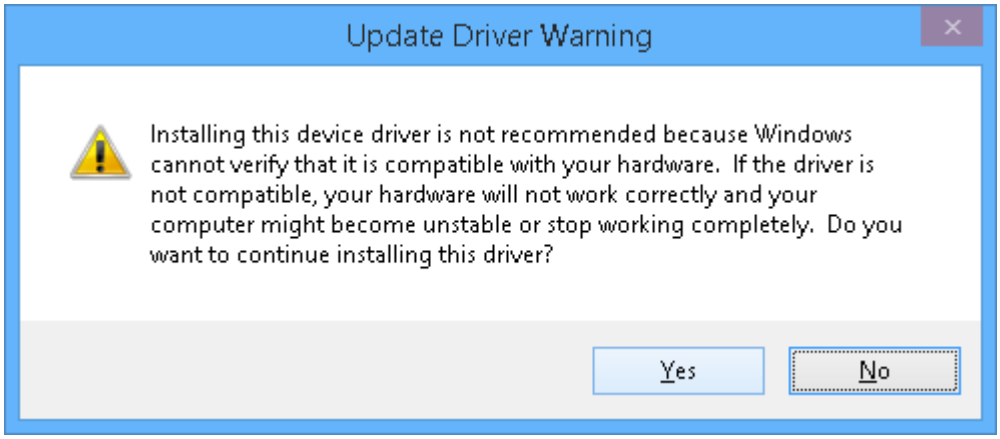

Figure 12 - Warning screen

The WinUSB driver is now installed and the device will appear in the 'Universal Serial Bus devices' section as below.

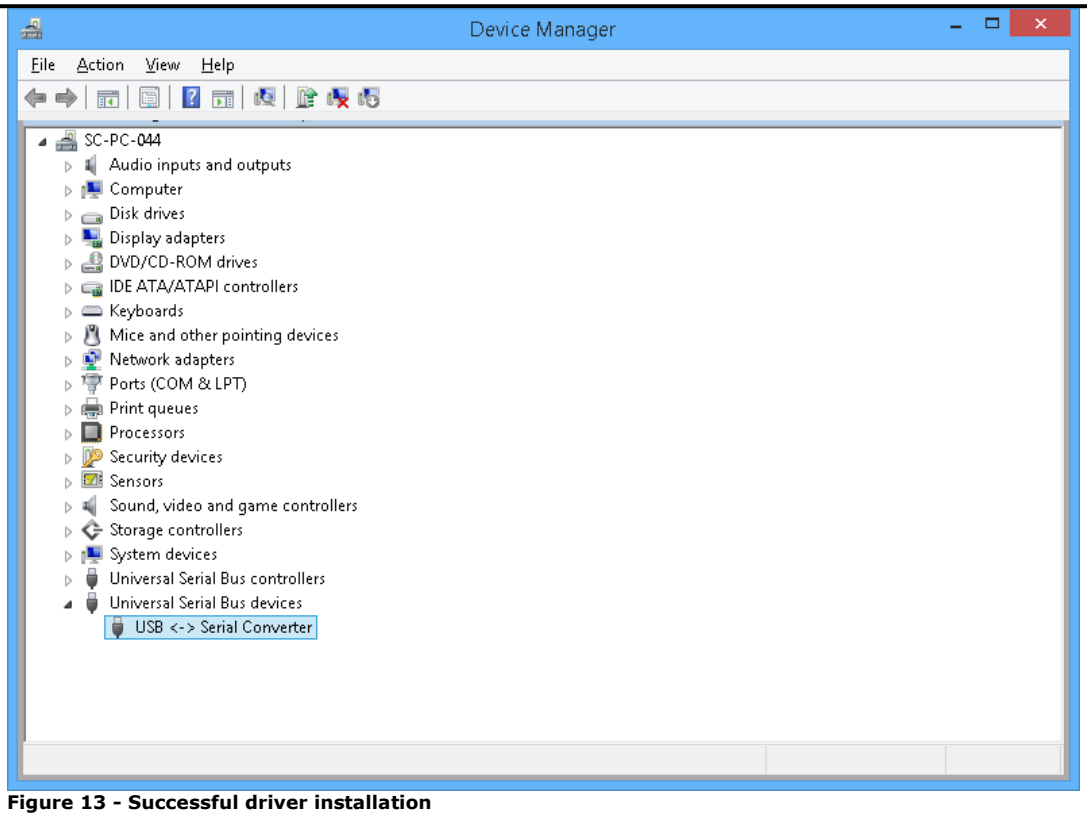

Figure 13 - Successful driver installation

Finally, the device must have a DeviceInterfaceGUID key within the registry before access to the device can be granted.

To add the key, open the Registry Editor and find the key that corresponds to the device in question:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\USB\<VID_vvvv&PID_pppp>

Where vvvv is the device vendor ID and pppp is the device product ID.

Under the **Device Parameters** key, add a string entry named **DeviceInterfaceGUID.** Set the data portion of the key to something of the following:

{7febe9fc-6c36-4244-a665-12b01d69750d}

Note: A valid GUID can be generated from a tool such as GuidGen.exe.

Unplug and replug the device into the same port - the driver is now installed successfully for the device.

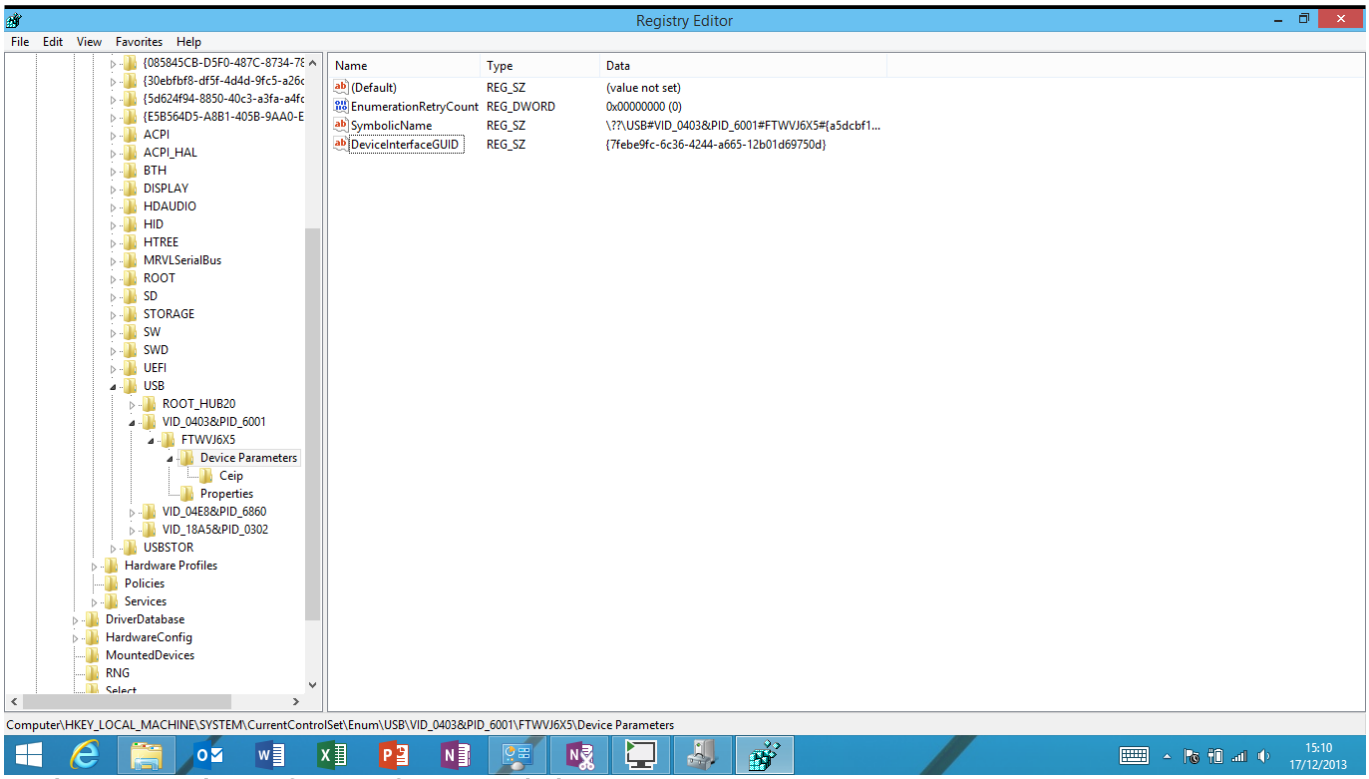

Figure 14 - DeviceInterfaceGUID for an FTDI device

3 Windows Store App

With the WinUSB driver installed for the FTDI device it is now possible to access it using the D2xx WinRT component. The host Store App must consume the component as a reference to the project. The instructions below discuss how to add the component to a Store App.

At the time of writing, Visual Studio 2013 is the minimum version required to create a Store App that uses the D2xx WinRT component.

3.1 Add Reference

Inside Visual Studio a blank Windows Store App has been created. As shown in Figure 15 a new project named TestApp has been created.

Right click on the project and select 'Add Reference…' from the context menu.

From the resulting screen (Figure 16) select 'Browse…' at the bottom right to select the component.

The driver is distributed as two components: FTDI.D2xx.WinRT.winmd; and FTDI.D2xx.WinRT.USB.winmd. The former is the API layer that is exposed to the application, with the latter being an internal component responsible for USB communication - this layer is not exposed to the user. Both of these components must be explicitly referenced by the host application for the driver to function.

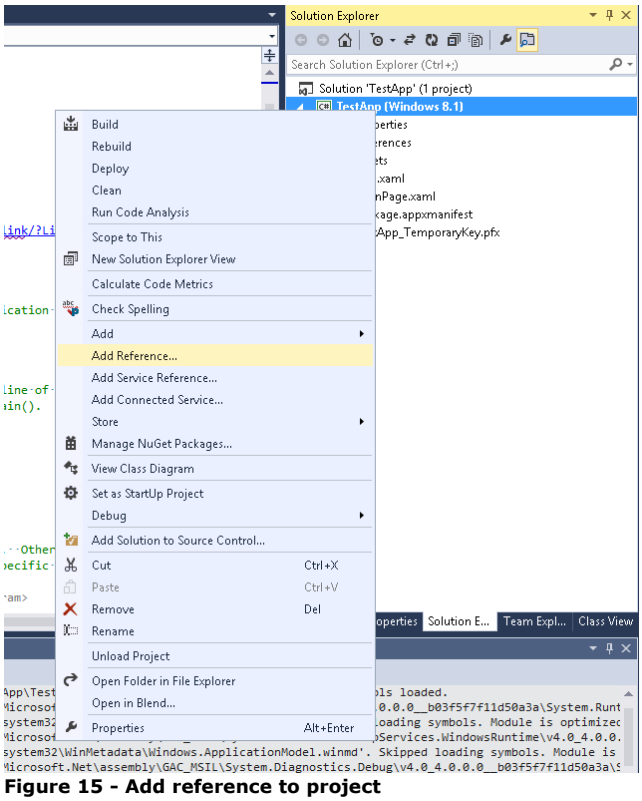

Figure 15 - Add reference to project

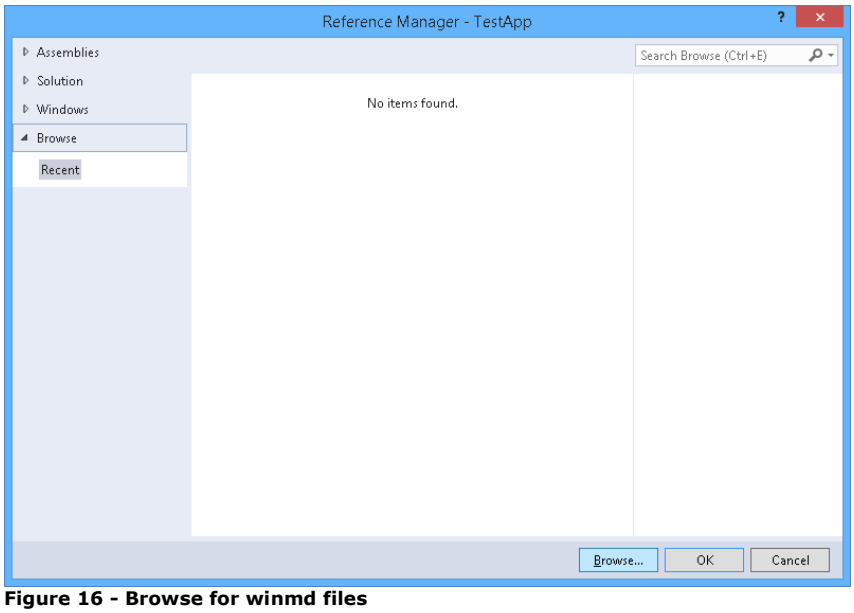

Figure 16 - Browse for winmd files

3.2 Manifest

As well as including the driver components the Store App must also explicitly reference the USB devices that it will communicate with. This is achieved through the Package.appxmanifest file that accompanies all Store Apps. The excerpt below shows a typical manifest file; here we are allowing the application access to all FTDI devices.

The vendor IDs and product IDs listed below are the FTDI defaults, but these can be added/removed to match any custom VID/PID combination required. The important thing to remember is that any device that the application wants access to must be listed in the manifest.
Note: *The VID and PID numbers are in hexadecimal format.*

```
...
<Capabilities>
    <m2:DeviceCapability Name="usb">
      <!--FT232AM, FT232BM, FT232R and FT245R Devices-->
      <m2:Device Id="vidpid:0403 6001">
        <m2:Function Type="name:vendorSpecific" />
      </m2:Device>
      <!--FT2232D and FT2232H Devices-->
      <m2:Device Id="vidpid:0403 6010">
        <m2:Function Type="name:vendorSpecific" />
      </m2:Device>
      <!--FT4232H Device-->
      <m2:Device Id="vidpid:0403 6011">
        <m2:Function Type="name:vendorSpecific" />
      </m2:Device>
      <!--FT232H Device-->
      <m2:Device Id="vidpid:0403 6014">
        <m2:Function Type="name:vendorSpecific" />
      </m2:Device>
      <!--FT-X-Series Devices-->
      <m2:Device Id="vidpid:0403 6015">
        <m2:Function Type="name:vendorSpecific" />
      </m2:Device>
    </m2:DeviceCapability>
</Capabilities>
...
```


3.3 Vendor ID, Product ID

The driver compromises three main types used to access and communicate with FTDI devices (Table 1). The first of these is the FTManager class defined within the namespace FTDI.D2xx.WinRT. This class is used to enumerate and list FTDI devices connected to the system.

Class/Interface	Description
FTManager	Class responsible for enumerating and listing all FTDI devices connected to the system. Used in combination with the IFTDeviceInfoNode interface to return an IFTDevice interface.
IFTDeviceInfoNode	Interface that contains information about a specific FTDI device connected to the system.
IFTDevice	Interface representing a generic FTDI device. Used to communicate with the physical device hardware. Returned from the FTManager class on an open.

Table 1 - Three main types within the WinRT driver.

On instantiation of the FTManager class the driver will automatically attempt to access any FTDI devices connected to the system. The devices it will attempt to communicate with correspond to the default FTDI vendor ID and product IDs as listed below.

Vendor ID	Product ID	Device(s)
0x0403	0x6001	FT232AM, FT232BM, FT232R and FT245R
0x0403	0x6010	FT2232D and FT2232H
0x0403	0x6011	FT4232H
0x0403	0x6014	FT232H
0x0403	0x6015	FT X-Series

Table 2 - List of default FTDI VIDs and PIDs

It is envisaged that developers may have their own vendor ID and/or product ID allocated and may not want to use the standard values. The FTManager class provides two functions to allow this:

	Name	Description
	AddVIDPID	Add a custom vendor ID/product ID combination to the list of allowed devices.
	RemoveVIDPID	Remove a custom vendor ID/product ID combination from the list of allowed devices.

These functions are static and must be called before an instance of the FTManager class is created as shown in the snippet below.

```
private FTManager InitializeDriver()
{
   FTManager.AddVIDPID(0x1234, 0x4321);
   FTManager.RemoveVIDPID(0x0403, 0x6001);
   FTManager.RemoveVIDPID(0x0403, 0x6010);
   FTManager.RemoveVIDPID(0x0403, 0x6011);
   FTManager.RemoveVIDPID(0x0403, 0x6014);
   FTManager.RemoveVIDPID(0x0403, 0x6015);
   return new FTManager();
}
```

This application will now only have access to devices corresponding to the custom vendor ID and product ID of 0x1234, 0x4321. Again, the vendor ID and product ID must be listed in the manifest to allow access to this device.

3.4 List Devices

With the driver initialized it is now possible to return a list of all FTDI devices matching the requested VID/PID combination that are connected to the system.

	Name	Description
	GetDeviceList	Returns a list of FTDI devices connected to the system.

The GetDeviceList function returns an IList of FTDI devices. The List contains elements of type IFTDeviceInfoNode from the namespace FTDI.D2xx.WinRT.Device; each element contains information about a specific FTDI device. The information within the IFTDeviceInfoNode is used to identify the device: the chip type; the serial number; the product description etc.

```
public void StartDevice()
{
    IList<IFTDeviceInfoNode> deviceList = ftManager.GetDeviceList();

    foreach (IFTDeviceInfoNode deviceInfo in deviceList)
    {
        Debug.WriteLine("Device Type: {0}\r\nSerial Number: {1}\r\nDescription: {2}\r\n\r\n ",
           deviceInfo.DeviceType.ToString(), deviceInfo.SerialNumber, deviceInfo.Description);
        if (deviceInfo.Description == "My USB Product")
        {

       ...
```

3.5 Open Device

With the device identified within the system it can now be opened using one of the 3 open functions provided by the FTManger class.

	Name	Description
	OpenByDescription	Open a handle to the specified FTDI device.
	OpenByDeviceID	Open a handle to the specified FTDI device.
	OpenBySerialNumber	Open a handle to the specified FTDI device.

The function parameters correspond to information contained within the IFTDeviceInfoNode interface. Each of the functions will return an IFTDevice interface that can subsequently be used to communicate with the physical device.

In the example below we have opened the device based on the Description from the IFTDeviceInfoNode. A simple loopback task has then been created to write and read data from the open device.

...

```
myDevice = ftManager.OpenByDescription(deviceInfo.Description);

await myDevice.SetBaudRateAsync(9600);
await myDevice.SetFlowControlAsync(FLOW_CONTROL.RTS_CTS, 0x00, 0x00);
await myDevice.SetDataCharacteristicsAsync(WORD_LENGTH.BITS_8, STOP_BITS.BITS_1,
                                           PARITY.NONE);
await myDevice.SetLatencyTimerAsync(16);

var action = ThreadPool.RunAsync(async (source) =>
{
    byte[] dataTx = new byte[10];
    for (int i = 0; i < dataTx.Length; i++)
        dataTx[i] = (byte)i;

    while (!cancellationTokenSource.Token.IsCancellationRequested)
    {
        byte[] dataRx = new byte[10];
        await myDevice.WriteAsync(dataTx, 10);
        myDevice.ReadAsync(dataRx, 10);
    }
}, WorkItemPriority.Normal);
...
```

3.6 EEPROM

The interface IFTDevice provides two functions for reading and writing the entire contents of a device EEPROM.

	Name	Description
	EepromReadAsync	Read the contents of the device EEPROM.
	EepromProgramAsync	Program the contents of the device EEPROM.

Reading the EEPROM data from the device requires the resultant interface to be cast to the correct underlying type. For example, below we have read the data from an FTDI device. We know that this device is a FT232R device and therefore we can safely cast this to the FT232R_EEPROM type.

```
private async void ReadEEPROM()
{
    if (myDevice != null)
    {
        IFT_EEPROM ee = await myDevice.EepromReadAsync();

        if (myDevice.DeviceInfoNode.DeviceType == DEVICE_TYPE.FT232R)
        {
            // Cast to the type that corresponds to the device type.
            FT232R_EEPROM eeData = ee as FT232R_EEPROM;

            if (eeData == null)
                return;

            Debug.WriteLine(@"Manufacturer:{0}\r\n
                            Serial Number: {1}\r\n
                            Product Description:
                            {2}\r\n\r\n ",
                            eeData.Manufacturer,
                            eeData.SerialNumber,
                            eeData.Product);
        }
```

```
    }
}
```

Conversely with programming, we pass the specific EEPROM type for that device into the EepromProgram function as shown below.

```
private async void ProgramEEPROM()
{
    if (myDevice == null)
        return;

    if (myDevice.DeviceInfoNode.DeviceType != DEVICE_TYPE.FT232R)
        return;

    FT232R_EEPROM ee = new FT232R_EEPROM();

    ee.VendorID = 0x0403;
    ee.ProductID = 0x6001;
    ee.LoadVCP = true;
    ee.Manufacturer = "FTDI";
    ee.Product = "FT232R";
    ee.SerialNumber = "FT7654321";
    ee.SerialNumberEnable = true;
    ee.UsbVersion = USB_VERSION.USB_20;
    ee.SelfPowered = false;
    ee.RemoteWakeupEnable = false;
    ee.PullDownEnable = false;
    ee.MaxPower = 500;
    ee.InvertTXD = false;
    ee.InvertRXD = false;
    ee.InvertRTS = false;
    ee.InvertRI = false;
    ee.InvertDTR = false;
    ee.InvertDSR = false;
    ee.InvertDCD = false;
    ee.InvertCTS = false;
    ee.HighIO = false;
    ee.ExternalOscillatorEnable = false;
    ee.CBus4 = FTDI.D2xx.WinRT.Device.EEPROM.FT232R.CBUS_SIGNALS.TXDEN;
    ee.CBus3 = FTDI.D2xx.WinRT.Device.EEPROM.FT232R.CBUS_SIGNALS.TXDEN;
    ee.CBus2 = FTDI.D2xx.WinRT.Device.EEPROM.FT232R.CBUS_SIGNALS.TXDEN;
    ee.CBus1 = FTDI.D2xx.WinRT.Device.EEPROM.FT232R.CBUS_SIGNALS.TXDEN;
    ee.CBus0 = FTDI.D2xx.WinRT.Device.EEPROM.FT232R.CBUS_SIGNALS.TXDEN;

    await myDevice.EepromProgramAsync(ee);
}
```

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

Branch Office – Tigard, Oregon, USA

Future Technology Devices International Limited
(USA)
7130 SW Fir Loop
Tigard, OR 97223
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

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

Branch Office – Shanghai, China

Future Technology Devices International Limited
(China)
Room 1103, No. 666 West Huaihai Road,
Shanghai, 200052
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

http://ftdichip.com

5 Appendix A – References

Document References

Acronyms and Abbreviations

Terms	Description
API	Application Programming Interface.
OS	Operating System.
PID	USB Product ID – part of the USB device descriptor.
VID	USB Vendor ID – part of the USB device descriptor.
WinRT	Windows RunTime – A set of API calls that sits on top of the Windows 8 OS - used by Windows Store Apps to communicate with the OS.
Windows RT	An ARM based version of the Windows 8 OS.
CDM	Combined Driver Model – the name given to the virtual COM port and D2xx combined driver for Windows.

6 Appendix B – List of Tables & Figures

List of Tables

Table 1 - Three main types within the WinRT driver. 13
Table 2 - List of default FTDI VIDs and PIDs 13

List of Figures

Figure 1 – Windows Store App Driver Architecture 2
Figure 2 - System Properties 3
Figure 3 - Windows Update 4
Figure 4 - Windows Device Manager 4
Figure 5 - Browse for INF file 5
Figure 6 - Successful driver installation 5
Figure 7 - Windows Device Manager 6
Figure 8 - Browse my computer 6
Figure 9 - Let me pick from a list of device drivers 7
Figure 10 – Universal Serial Bus Devices 7
Figure 11 - WinUSB driver 8
Figure 12 - Warning screen 8
Figure 13 - Successful driver installation 9
Figure 14 - DeviceInterfaceGUID for an FTDI device 10
Figure 15 - Add reference to project 11
Figure 16 - Browse for winmd files 12

7 Appendix C - Code Listing

7.1 Package.appxmanifest

```
<?xml version="1.0" encoding="utf-8"?>
<Package xmlns="http://schemas.microsoft.com/appx/2010/manifest"
xmlns:m2="http://schemas.microsoft.com/appx/2013/manifest">
  <Identity Name="03f55a05-25cb-4310-9d66-97535ca1fed9" Publisher="CN=FTDI"
Version="0.9.0.2" />
  <Properties>
    <DisplayName>TestApp</DisplayName>
    <PublisherDisplayName>FTDI</PublisherDisplayName>
    <Logo>Assets\StoreLogo.png</Logo>
  </Properties>
  <Prerequisites>
    <OSMinVersion>6.3.0</OSMinVersion>
    <OSMaxVersionTested>6.3.0</OSMaxVersionTested>
  </Prerequisites>
  <Resources>
    <Resource Language="x-generate" />
  </Resources>
  <Applications>
    <Application Id="App" Executable="$targetnametoken$.exe" EntryPoint="TestApp.App">
      <m2:VisualElements DisplayName="TestApp" Square150x150Logo="Assets\Logo.png"
Square30x30Logo="Assets\SmallLogo.png" Description="TestApp" ForegroundText="light"
BackgroundColor="#464646">
        <m2:SplashScreen Image="Assets\SplashScreen.png" />
      </m2:VisualElements>
    </Application>
  </Applications>
  <Capabilities>
    <m2:DeviceCapability Name="usb">
      <!—My Custom Device-->
      <m2:Device Id="vidpid:1234 4321">
        <m2:Function Type="name:vendorSpecific" />
      </m2:Device>
      <!—FT232BM, FT232R-->
      <m2:Device Id="vidpid:0403 6001">
        <m2:Function Type="name:vendorSpecific" />
      </m2:Device>
    </m2:DeviceCapability>
  </Capabilities>
</Package>
```

7.2 MainPage.xaml.cs

```
using FTDI.D2xx.WinRT;
using FTDI.D2xx.WinRT.Device;
using FTDI.D2xx.WinRT.Device.EEPROM;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Threading;
using Windows.System.Threading;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;

namespace TestApp
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame.
    /// </summary>
    public sealed partial class MainPage : Page
    {
        private FTManager ftManager;
        private IFTDevice myDevice = null;
        private CancellationTokenSource cancellationTokenSource = new
                                            CancellationTokenSource();

        public MainPage()
        {
            this.InitializeComponent();
            ftManager = InitializeDriver();
        }

        private FTManager InitializeDriver()
        {
#if CUSTOM_VID_PID
            bool result = FTManager.AddVIDPID(0x1234, 0x4321);
            result = FTManager.RemoveVIDPID(0x0403, 0x6001);
            result = FTManager.RemoveVIDPID(0x0403, 0x6010);
            result = FTManager.RemoveVIDPID(0x0403, 0x6011);
            result = FTManager.RemoveVIDPID(0x0403, 0x6014);
            result = FTManager.RemoveVIDPID(0x0403, 0x6015);
#endif
            return new FTManager();
        }

        public async void StartDevice()
        {
            IList<IFTDeviceInfoNode> deviceList = ftManager.GetDeviceList();

            foreach (IFTDeviceInfoNode deviceInfo in deviceList)
            {
                Debug.WriteLine("Device Type: {0}\r\nSerial Number: {1}\r\nDescription:
                                {2}\r\n\r\n ",
                                deviceInfo.DeviceType.ToString(),
                                deviceInfo.SerialNumber,
                                deviceInfo.Description);
                if (deviceInfo.DeviceType == DEVICE_TYPE.FT232R)
                {
                    myDevice = ftManager.OpenByDescription(deviceInfo.Description);

                    await myDevice.SetBaudRateAsync(9600);
                    await myDevice.SetFlowControlAsync(FLOW_CONTROL.RTS_CTS, 0x00, 0x00);
                    await myDevice.SetDataCharacteristicsAsync(WORD_LENGTH.BITS_8,
                              STOP_BITS.BITS_1,
                             PARITY.NONE);
```

```
                await myDevice.SetLatencyTimerAsync(16);

                var action = ThreadPool.RunAsync(async (source) =>
                {
                    byte[] dataTx = new byte[10];
                    for (int i = 0; i < dataTx.Length; i++)
                        dataTx[i] = (byte)i;

                    while (!cancellationTokenSource.Token.IsCancellationRequested)
                    {
                        byte[] dataRx = new byte[10];
                        await myDevice.WriteAsync(dataTx, 10);
                        myDevice.ReadAsync(dataRx, 10);
                    }
                }, WorkItemPriority.Normal);
            }
        }
    }

    private async void ReadEEPROM()
    {
        if (myDevice != null)
        {
            IFT_EEPROM ee = await myDevice.EepromReadAsync();

            if (myDevice.DeviceInfoNode.DeviceType == DEVICE_TYPE.FT232R)
            {
                // Cast to the type that corresponds to the device type.
                FT232R_EEPROM eeData = ee as FT232R_EEPROM;

                if (eeData == null)
                    return;

                Debug.WriteLine(@"Manufacturer: {0}\r\nSerial Number: {1}\r\nProduct
                                Description: {2}\r\n\r\n ",
                                eeData.Manufacturer,
                                eeData.SerialNumber,
                                eeData.Product);
            }
        }
    }

    private async void ProgramEEPROM()
    {
        if (myDevice == null)
            return;

        if (myDevice.DeviceInfoNode.DeviceType != DEVICE_TYPE.FT232R)
            return;

        FT232R_EEPROM ee = new FT232R_EEPROM();

        ee.VendorID = 0x0403;
        ee.ProductID = 0x6001;
        ee.LoadVCP = true;
        ee.Manufacturer = "FTDI";
        ee.Product = "FT232R";
        ee.SerialNumber = "FT7654321";
        ee.SerialNumberEnable = true;
        ee.UsbVersion = USB_VERSION.USB_20;
        ee.SelfPowered = false;
        ee.RemoteWakeupEnable = false;
        ee.PullDownEnable = false;
        ee.MaxPower = 500;
```

```
            ee.InvertTXD = false;
            ee.InvertRXD = false;
            ee.InvertRTS = false;
            ee.InvertRI = false;
            ee.InvertDTR = false;
            ee.InvertDSR = false;
            ee.InvertDCD = false;
            ee.InvertCTS = false;
            ee.HighIO = false;
            ee.ExternalOscillatorEnable = false;
            ee.CBus4 = FTDI.D2xx.WinRT.Device.EEPROM.FT232R.CBUS_SIGNALS.TXDEN;
            ee.CBus3 = FTDI.D2xx.WinRT.Device.EEPROM.FT232R.CBUS_SIGNALS.TXDEN;
            ee.CBus2 = FTDI.D2xx.WinRT.Device.EEPROM.FT232R.CBUS_SIGNALS.TXDEN;
            ee.CBus1 = FTDI.D2xx.WinRT.Device.EEPROM.FT232R.CBUS_SIGNALS.TXDEN;
            ee.CBus0 = FTDI.D2xx.WinRT.Device.EEPROM.FT232R.CBUS_SIGNALS.TXDEN;

            await myDevice.EepromProgramAsync(ee);
        }

        private void btnListDevices_Click(object sender, RoutedEventArgs e)
        {
            StartDevice();
        }

        private void btnReadEEPROM_Click(object sender, RoutedEventArgs e)
        {
            ReadEEPROM();
        }

        private void btnProgramEEPROM_Click(object sender, RoutedEventArgs e)
        {
            ProgramEEPROM();
        }
    }
}
```

7.3 MainPage.xaml

```
<Page
    x:Class="TestApp.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:TestApp"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <StackPanel Background="#FF49C4F1">
        <Button x:Name="btnStartDevice" Content="Start Device" Click="btnListDevices_Click" />
        <Button x:Name="btnReadEEPROM" Content="Read EEPROM" Click="btnReadEEPROM_Click" />
        <Button x:Name="btnProgramEEPROM" Content="Program EEPROM"
                Click="btnProgramEEPROM_Click" />
    </StackPanel>
</Page>
```

8 Appendix D – Revision History

Document Title:	D2xx WinRT Guide AN_271
Document Reference No:	FT_000928
Clearance No:	FTDI# 395
Product Page:	http://www.ftdichip.com/FTProducts.htm
Document Feedback:	Send Feedback

Version 1.0	Initial Release	01/07/2014