Freescale Semiconductor
Application Note

Document Number: AN4314
Rev. 1, 07/2011

Using USB2SER DLL in C# Projects

by: Juan Cazares
IMM Software Engineer

1 Introduction

This application note explains how to use the USB2SER DLL in any C# (C sharp) project. This DLL is used to communicate with the USB2SER, which allows the reading and modification of USB descriptors allocated in nonvolatile memory.

First, the application note explains how to include and use the USB2SER DLL in a C# project. Second, it explains in detail the application programming interface (API) of the USB2SER DLL. Third, this document explains how the serial production GUI is implemented and how it uses the USB2SER DLL.

Contents

freescale™ semiconductor

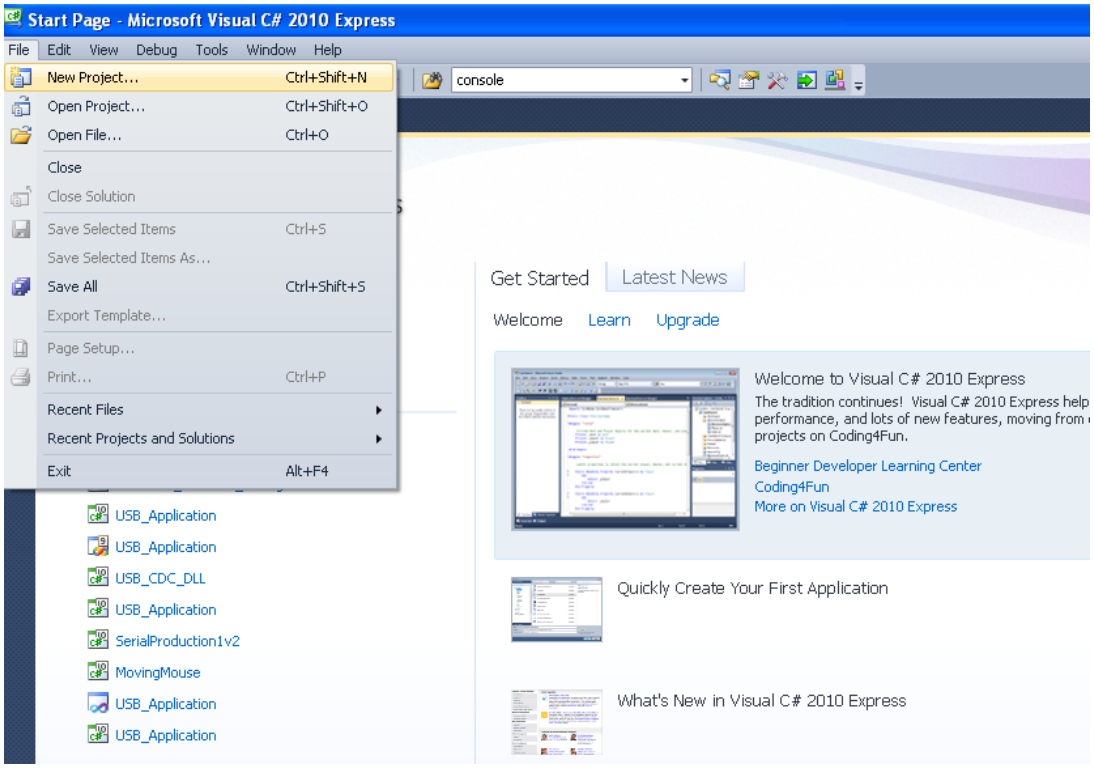
2 Using DLLs in a C# Project

2.1 Creating a new DLL folder

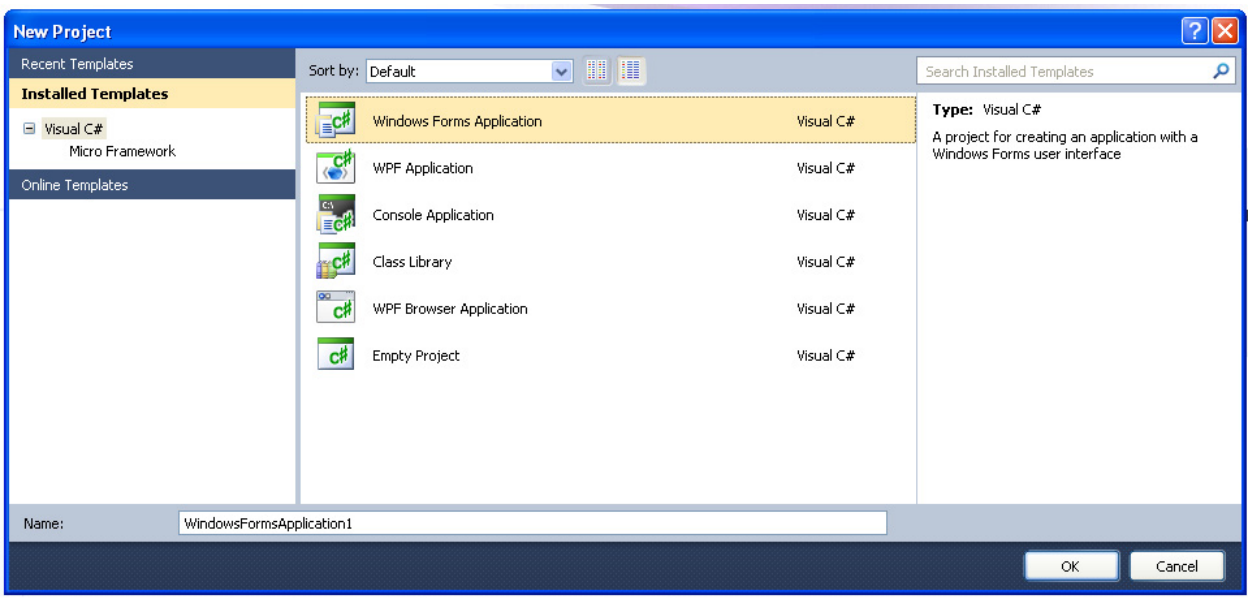
The USB2SER_DLL.dll can be used in an existent project. This document explains how to create a new project and then how to use the DLL.

Open C# studio and follow the next steps:

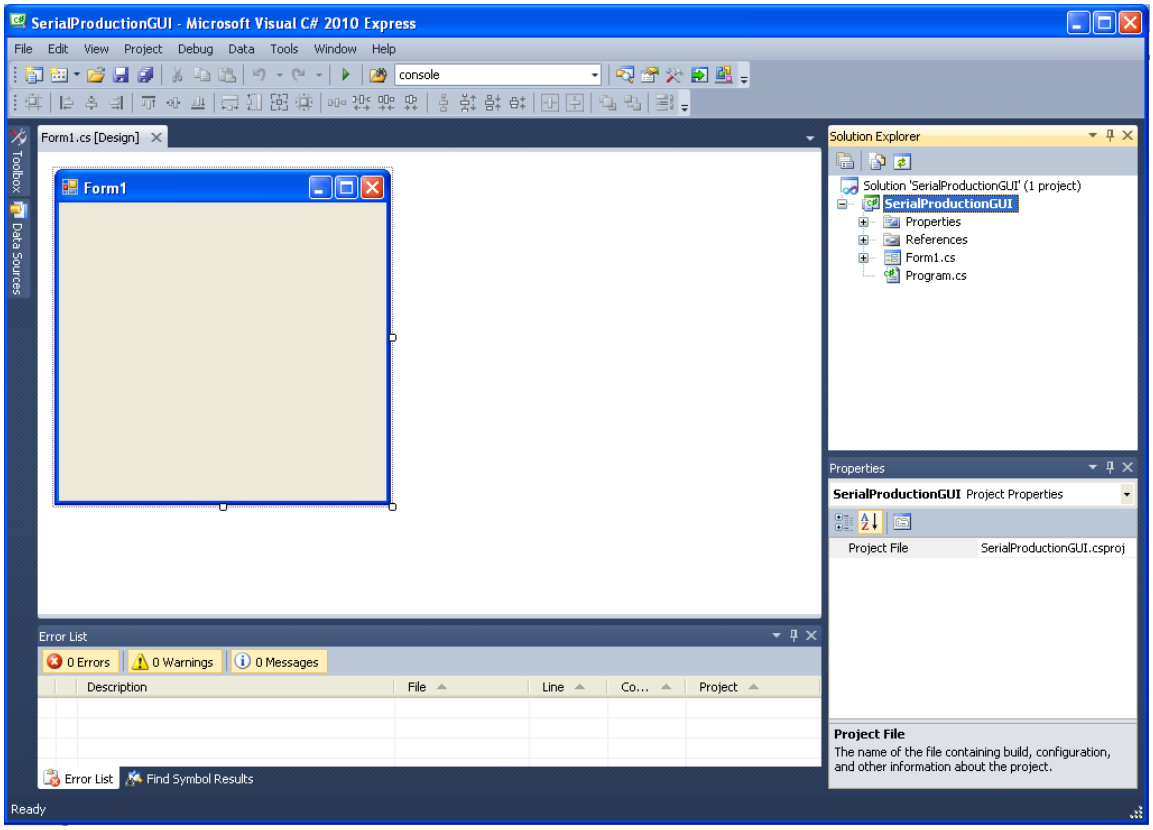
1. Choose the File pull-down menu.
2. Choose the New Project option.



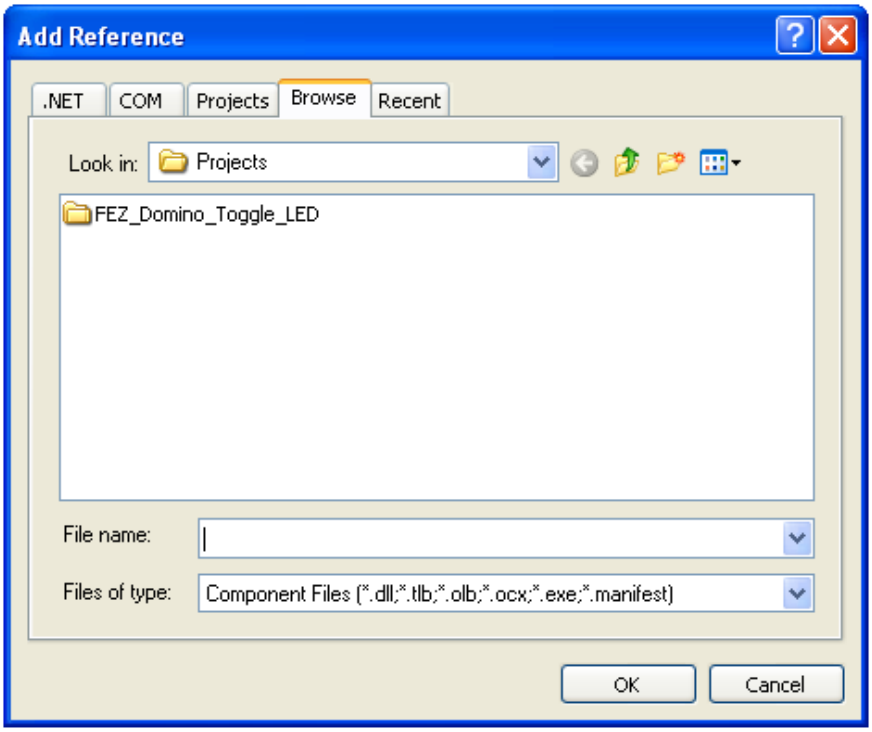
3. On the New Project screen, choose Windows Forms Applications.
4. Change the name for the new application (for example, “SerialProductionGUI”) and click OK.



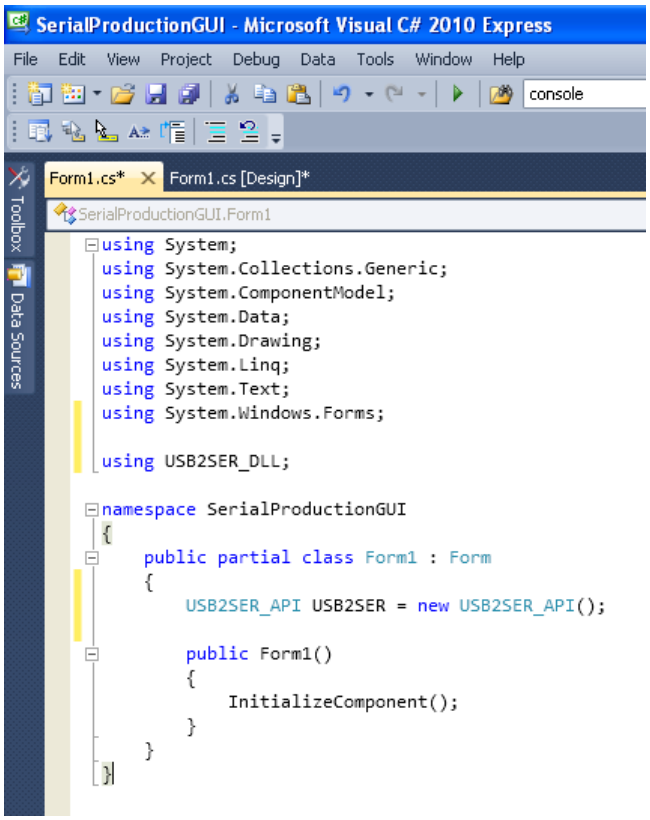
The project is created and shows an empty form.



5. On the solution explorer, right click on References, then choose Add Reference.



6. Select the Browse tab and look for the USB2SER_DLL.dll file. Select it and click OK.
7. Repeat steps 6 and 7 for any other desired DLL files.
8. Open the Form1 code and add the class USB2SER_DLL with the “using” word.



9. Now, by declaring a new USB2SER_API object the Form1 can use the methods provided by the referenced class.

Download and open the SerialProduction1v3 project from www.freescale.com for more information about the mentioned class.

2.2 Application programming interface

This section explains in detail all methods used by the USB2SER_DLL class.

2.2.1 USB2SER_DLL class

This class allows the configuration of the USB2SER's nonvolatile memory, which is intended to store the USB descriptors. This class provides methods to read and modify vendor ID, product ID, manufacturer string, product string, serial number string, and maximum current consumption. These parameters are protected by a password which should be validated first in order to allow access to the nonvolatile memory. This class also provides a method for changing the password.

The USB2SER_DLL class uses a Freescale proprietary USB CDC class to communicate with USB2SER devices. Please ensure that the USB2SER USB driver has been installed before attempting to use the USB2SER_DLL DLL.

2.2.2 USB2SER_DLL application programming interface

Public Variables: The USB2SER_DLL uses public variables as inputs and outputs for the methods. The variables listed below are read-only variables used by the *ReadUSBDescriptors* method:

```
public bool VendorID
public bool ProductID
public bool StrManufacturer
public bool StrProduct
public bool StrSerial
public bool NumConsumption
```

The variables listed below are read/write variables and are used by the application. The *BusPowered* variable configures whether the device is self- or bus-powered. The *SerialAutoIncrement* variable configures whether the serial number string is incremented each time the nonvolatile memory is written or whether it remains unchanged.

```
public bool BusPowered
public bool SerialAutoIncrement
public bool HexFormat
```

2.2.3 Methods

Table 1. USB2SER C# Methods

	Syntax	Description	Parameters	Return
OpenPort	`public bool OpenPort(string COM);`	Opens the serial port to communicate with the USB2SER device and enters the device in loader mode. Only one port can be opened at the time.	COM: string containing the serial port to open	TRUE: port closed successfully FALSE: port can´t be closed
ClosePort	`public bool ClosePort();`	Closes the serial port opened by the OpenPort method. Closes the port but the device remains in loader mode.	Void	TRUE: port closed successfully FALSE: port can´t be closed
ClosePort AndReset	`public bool ClosePortAnd Reset();`	Closes the serial port opened by the OpenPort method and forces the USB2SER device to come out of loader mode by performing a software reset. Should be used when an error occurs or just before closing the main application.	Void	TRUE: port closed successfully FALSE: port can´t be closed
Validate Password _CMD	`public bool ValidatePassword _CMD (string Password);`	Validates the password string, sends the password validation command, and waits for the answer.	Password: string containing the password to be validated by the USB2SER. Must always be 16 characters long. Only hexadecimal values are valid.	TRUE: port closed successfully FALSE: port can´t be closed
Change Password _CMD	`public bool ChangePassword _CMD(string Password);`	Validates the password string, sends the command to change the current password, and waits for the answer.	Password: string containing the new password. Must always be 16 characters long. Only hexadecimal values are valid.	TRUE: port closed successfully FALSE: port can´t be closed
Change VendorID _CMD	`public bool ChangeVendorID _CMD(string VendorID);`	Validates the vendor ID string, sends the command to change the current vendor ID, and waits for the answer.	VendorID: string containing the new vendor ID. Must always be 4 characters long. Only hexadecimal values are valid.	TRUE: port closed successfully FALSE: port can´t be closed

Table 1. USB2SER C# Methods (continued)

	Syntax	Description	Parameters	Return
Change ProductID _CMD	`public bool ChangeProductID _CMD(string ProductID);`	Validates the product ID string, sends the command to change the current product ID, and waits for the answer.	ProductID: string containing the new product ID. Must always be 4 characters long. Only hexadecimal values are valid.	TRUE: port closed successfully FALSE: port can´t be closed
Change Manufacturer String_CMD	`public bool ChangeManufacturer String_CMD (string strManufacturer);`	Validates the manufacturer string, sends the command to change the current manufacturer string, and waits for the answer.	strManufacturer: string containing the new manufacturer string. Must be from 1–63 characters long. Valid range is any printable character from 32–126.	TRUE: port closed successfully FALSE: port can´t be closed
Change Product String_CMD	`public bool ChangeProduct String_CMD(string strProduct);`	Validates the product string, sends the command to change the current product string, and waits for the answer.	strProduct: string containing the new product string. Must be from 1–63 characters long. Valid range is any printable character from 32–126.	TRUE: port closed successfully FALSE: port can´t be closed
Change Serial Number String_CMD	`public bool ChangeSerialNumber String_CMD(string strSerialNum);`	Closes the serial port opened by the OpenPort method. Closes the port, but the device remains in loader mode.	strSerialNum: string containing the new serial number string. Must be from 1–63 characters long. Valid range is any printable character from 32–126.	TRUE: port closed successfully FALSE: port can´t be closed
Change MaxCurrent Consumption _CMD	`public bool ChangeMaxCurrent Consumption_CMD (string Milliamperes);`	Validates the milliamperes string, sends the command to change the maximum current consumption, and waits for the answer.	Milliamperes: string containing the new current consumption value. The valid range is even values from 20–500 mA, if the bus-powered option is selected. For self-powered the valid value must be 0.	TRUE: port closed successfully FALSE: port can´t be closed

ReadUSB Descriptors _CMD	`public bool ReadUSBDescriptors _CMD();`	Sends the command to read the entire nonvolatile memory where the USB descriptors are located. Stores the USB descriptors in the following public variables:
		`Public string VendorID`
		`Public string ProductID`
		`Public string StrManufacturer`
		`Public string StrProduct`
		`Public string StrSerial`
		`Public string NumConsumption`
			Void	TRUE: port closed successfully
				FALSE: port can´t be closed

DiscoverUSB Device	`public string DiscoverUSBDevice (string VID, string PID);`	Validates the VID and PID parameters and polls all enumerated USB devices looking for those that have the desired VID and PID.	VID: string containing the vendor ID to match. PID: string containing the product ID to match. Both parameters must always be 4 characters long. Only hexadecimal values are valid	Returns NULL if the desired VID and PID was not found. Returns a string containing the friendly port name if the desired VID and PID were found.

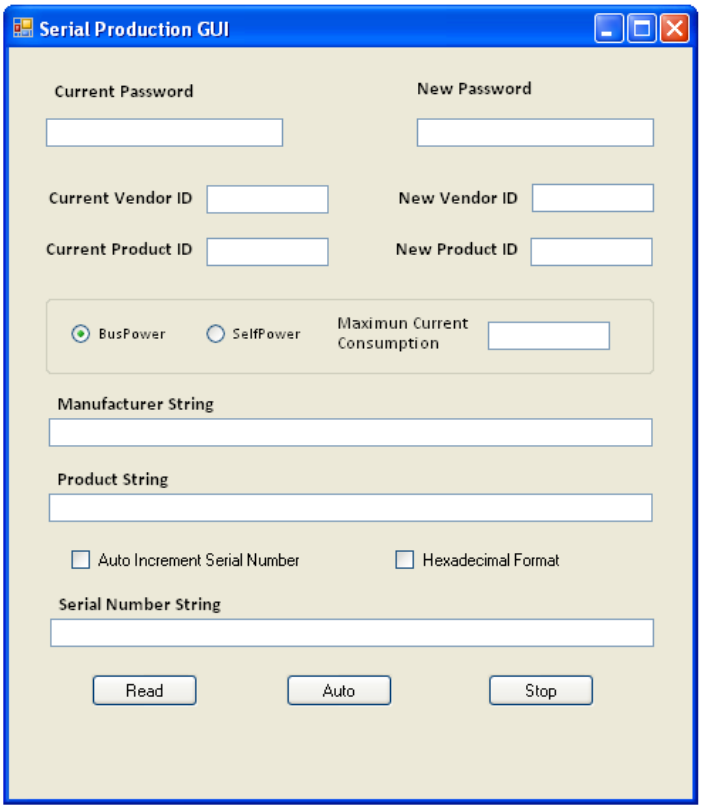
Table 1. USB2SER C# Methods (continued)

	Syntax	Description	Parameters	Return
Validate Password	`public bool ValidatePassword (string strField);`	Validates if the password is in hexadecimal format and 16 characters long.	strField: string to be validated	TRUE: string is valid FALSE: string is not valid
Validate VendorID	`public bool ValidateVendorID (string strField);`	Validates if the product ID is in hexadecimal format and 4 characters long.	strField: string to be validated	TRUE: port closed successfully FALSE: port can´t be closed
Validate ProductID	`public bool Validate ProductID(string strField);`	Validates if the product ID is in hexadecimal format and 4 characters long.	strField: string to be validated	TRUE: port closed successfully FALSE: port can´t be closed
Validate Manufacturer String	`public bool Validate ManufacturerString (string strField);`	Validates if the manufacturer string is a printable ASCII symbol not longer than 63 characters.	strField: string to be validated	TRUE: port closed successfully FALSE: port can´t be closed
Validate Product String	`public bool ValidateProduct String(string strField);`	Validates if the product string is a printable ASCII symbol not longer than 63 characters.	strField: string to be validated	TRUE: port closed successfully FALSE: port can´t be closed
Validate SerialNum String	`public bool ValidateSerial NumString(string strField);`	If the SerialAutoIncrement variable is false, this method validates if the product string is a printable ASCII symbol not longer than 63 characters.If the SerialAutoIncrement variable is true and the HexFormat variable is false, this method validates that the product string is in decimal format and no longer than 8 characters. If the SerialAutoIncrement variable is true and the HexFormat variable is true, this method validates that the product string is in hexadecimal format and not longer than 8 characters.	strField: string to be validated	TRUE: port closed successfully FALSE: port can´t be closed
Validate MaxCurrent Consumption	`public bool ValidateMaxCur-rentConsumtion (string strField);`	If the *BusPowered* variable is true, this method validates that the maximum current consumption is between 20–500 mA. If the *BusPowered* variable is false, this method validates that the maximum current consumption is 0 mA.	strField: string to be validated	TRUE: port closed successfully FALSE: port can´t be closed

3 Serial Production GUI User Guide

This section explains how the serial production GUI example works. The serial production GUI allows the reading and writing of the USB2SER USB descriptors by using the USB2SER_DLL DLL.

The serial production GUI has only one form named *MainForm*.



3.1 Valid formats and values for USB descriptors' fields

The methods ValidatePassword, ValidateVendorID, ValidateProductID, ValidateManufacturerString, ValidateProductString, ValidateSerialNumString, and ValidateMaxCurrentConsumption can be called by the application at any time. These methods are useful for validating any field on the C# form before executing any read or write process.

The serial production GUI uses these methods to ensure all fields on the MainForm are within a valid range. A pop-up error notification is displayed if an error is detected.

3.2 Discover USB devices

To establish communication with the USB2SER device, the USB host must first enumerate the device. The enumeration process assigns a COM port to the USB2SER device. The port number assigned by the USB host can be any number. Any application trying to communicate with the USB2SER needs to know the port number to open it and to be able to send and receive data.

The USB2SER_DLL has a utility, named DiscoverUSBDevice, for the discovery of USB devices. This method can detect and receive information about any USB device enumerated on the USB host. The scope of this method was reduced to detect only USB CDC class devices.

Since the USB host can enumerate USB CDC class devices other than the USB2SER device, the DiscoverUSBDevice method receives a vendor ID and a product ID as parameters to discover only those devices that its vendor ID and product ID match with the received parameters.

The serial production GUI is intended to communicate with one USB2SER device at the time. Ensure that no more than one device with the same vendor ID and product ID is connected.

3.3 Read USB descriptors

To read the USB descriptors, the field's current password, current vendor ID, and current product ID must be filled before pressing the READ button. The reading process uses the DiscoverUSBDevice method to discover the port number of the enumerated USB2SER device. If the READ button is pressed and no USB2SER device is enumerated, the process stays in a loop until the device is enumerated or the STOP button is pressed. When the device is connected to the USB host, the reading process starts automatically.

3.4 Customize USB descriptors

To customize the USB descriptors, all fields on the form must be filled before pressing the AUTO button.

Once all fields are completed, the AUTO button can be pressed to start the customization process. This process uses the DiscoverUSBDevice method to discover the port number of the enumerated USB2SER device. If the AUTO button is pressed and no USB2SER device is enumerated, the process stays in a loop until the device is enumerated or the STOP button is pressed. When the device is connected to the USB host, the customization process starts automatically. This process allows the customization of several devices in an automated manner much like a serial production line. So when the customization process ends, the next device can be connected and the application will start to customize the device automatically using the same parameters.

The serial number string can be modified using the same value if the "Auto incremental serial number" box is not checked.

The serial number string can be modified using an incremental value if the "Auto incremental serial number" box is checked. In this mode, the initial string will be increased each time a USB2SER device is connected on the USB host. The number is increased in hexadecimal format if the "Hexadecimal format" box is checked. Otherwise the number is increased in decimal format.

4 Conclusion

The DLLs described in this document provide a secure way to use previously-validated methods and objects. They also allow code reuse and speed up the development of new designs. DLLs also protect intellectual property since the DLL does not provide source code.

5 Revision History

Table 2. Revision History

Version	Changes
Rev. 0	First public version of this document.
Rev. 1	Changed “USB2SER SDA” to “USB2SER” in first paragraph on page 1.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN4314
Rev. 1
07/2011

freescale™
semiconductor