User's Manual

.NET Interop for BusTools-1553

abaco SYSTEMS

Copyrights

.NET Interop for BusTools-1553
BusTools/1553-API Software Revision: 8.22
Document Date: 01 Feb 2018

Abaco Systems, Inc.
26 Castilian Drive, Suite B
Goleta, CA 93117
Main +1 805-965-8000 or +1 805-883-6101
Support +1 805-965-6097

support@abaco.com (email)

https://www.abaco.com/products/avionics

Additional Resources

For more information, please visit the Abaco Systems website at:
www.abaco.com

Contents

CHAPTER 1

Overview

Revision

This revision of the .NET Interop Solution is paired with BusTools-1553 API v8.22. It should not be used with other versions of the BusTools-1553 API. This revision requires Microsoft Visual Studio 2008 or later.

Introduction

The .NET Interop Reference Solution can give you a quick start to harnessing the power of the .NET framework for your 1553 application.

The BusTools-1553 API library is an "unmanaged" DLL, and does not use the .NET framework. By using managed wrapper classes, you can use the BusTools-1553 library in your managed application.

The reference solution consists of:

- A managed wrapper class written in C# that encapsulates the BusTools-1553 API and data types.
- A sample managed GUI written in C# that uses the wrapper class to operate an Abaco 1553 board.
- A sample C# project and C++ unmanaged DLL project that can be used as a workbench to explore aspects of .NET interop.

The managed wrapper class can be used with C#, VB.NET, or any of the managed languages .NET supports.

This documentation assumes familiarity with Visual Studio 2008 or later, and creating and running .NET applications.

CHAPTER 2

The Reference Solution

Description

The reference solution is a Microsoft Visual Studio 2008 solution in the root folder, named **BustoolsInterop.sln**. When you open this solution, you will see four projects in the solution explorer window.

- **BustoolsCsApp**: A C# GUI that opens and operates a 1553 board.
- **BustoolsCsWrapper**: A C# class library that wraps the unmanaged BusTools-1553 API calls, constants, and data types.
- **BustoolsInterop**: A C# project that works in conjunction with the CDll project to demonstrate .NET interop concepts.
- **CDll**: An unmanaged C++ DLL that represents an unmanaged API. Used by the BustoolsInterop project.

Familiarity with .NET and C# concepts including interop is required to fully understand the Reference Solution.

Note: The Reference Solution demonstrates one way to use BusTools-1553 with a .NET application. Other ways and other user-defined wrapper definitions are possible.

CHAPTER 3

The API, Data, and Constants Classes

The Bustools1553 Namespace

The **Bustools1553** namespace encapsulates all the API calls, Data Types, and Constant definitions. It is recommended you do not change this namespace name, as it identifies the wrapper and provides name separation when loaded into other projects.

The API Class

The API static class contains managed entry points for each API call in the unmanaged BusTools-1553 library. .NET interop requires that managed entry points be contained in a static class. This class is found in file **API.cs**.

The DataTypes Namespace

The DataTypes namespace contains managed equivalents of the structures required by the unmanaged BusTools-1553 library. The managed equivalents are implemented using C# structures, classes, and unions. This namespace is found in file **DataTypes.cs.**

The Constants Class

The Constants static class contains managed definitions of the constants required by the unmanaged BusTools-1553 library. This class is found in file **Constants.cs**.

CHAPTER 4

Building and Running the Application

You can build and run the Reference Application "out of the box", when the following requirements are met:

- You have Microsoft Visual Studio 2008 or later.
- BusTools-1553 hardware and API v8.18 or later have been properly installed on your host.

In Visual Studio, set BustoolsCsApp as the startup project, and then do "Rebuild All". You can then run BustoolsCsApp.

CHAPTER 5

Adding the Managed Wrapper to an Existing .NET Application

First, it is suggested (but not required) that you add the C# project **BustoolsCsWrapper** to your existing .NET solution.

Then in the Solution Explorer, right-click your project and select "Add Reference". If you added BustoolsCsWrapper to your solution, click the Projects tab and select BustoolsCsWrapper. Otherwise, select the Browse tab and locate BustoolsCsWrapper.dll on your disk.

At the top of each of your code pages, add the following lines:

```
using Bustools1553;
using Bustools1553.DataTypes;
using System.Runtime.InteropServices;
```

You may need to edit the file **API.cs** in the BustoolsCsWrapper project. At the top of this file is a statement that defines exactly where Busapi32.dll should be found. By default, no path is specified, so the standard Windows search order will be used.

You can now use the managed wrapper classes in your project.

CHAPTER 6

Important Coding Differences when using .NET Wrappers

Managed .NET Applications can't directly access memory via pointers, so there is no possibility of sharing memory between the Managed Application and an Unmanaged DLL. This is where "Interop Marshaling" with "Platform Invoke" a.k.a. "P/Invoke" comes in.

When a Managed Application calls functions in an Unmanaged DLL, the parameters to be passed are "Marshaled" across the managed / unmanaged boundary by P/Invoke. In general, Marshaling means copying data across the boundary in one or both directions.

When a structure or class is Marshaled to the Unmanaged DLL, it is copied to the DLL's address space where the DLL can access it. When the function returns, the structure may be Marshaled back to the Managed Application if specified. Because of this, certain BusTools operations that require shared pointers to **IntFifo structures** need to be modified.

Additionally, Marshalling presents some challenges especially with multi-dimensional arrays which are embedded in structures. Because of this, it is required to flatten the array and use accessor functions to get at it with multiple indices.

Lastly, most BusTools functions require pointers to structures, so a **.NET class** (a reference type) is required. However, some BusTools functions require arrays of structures. These must be Marshaled as a **.NET struct** (a value type). This means that the **.NET wrapper** contains both struct and class representations of certain data types.

IntFifo Creation and Updating

When using the BusTools API in C, the user creates the IntFifo and passes the API a pointer to it. When using .NET, the user requests that the API create the IntFifo because it must stay in unmanaged memory.

Additionally, when the managed application has completed its IntFifo processing and modified the tail pointer, It must make a special call to have the API update its version of the tail pointer.

For an example of this, see the function **SetupBcIntFifo** in **RtTests.cs**, in the **BustoolsCsApp** project.

Multi-Dimensional Array Data

The **API_INT_FIFO** Data Type contains the embedded multi-dimensional arrays **eventMask** and **filterMask**. The wrapper flattens these and defines these members as Private. To access them, use the accessor functions **GetEventMask**, **SetEventMask**, **GetFilterMask**, and **SetFilterMask**.

The **API_BC_MBUF** and **API_BC_MBUF_STRUCT** Data Type contains the embedded multi-dimensional array data. The wrapper flattens it and defines this member as Private. To access it, use the accessor functions **GetData** and **SetData**.

Class and Struct versions of Data Types

The BusTools API functions **BC_ReadLastMessageBlock**, **BM_ReadLastMessageBlock**, and **RT_ReadLastMessageBlock** require arrays of **API_BC_MBUF**, **API_BM_MBUF**, and **API_RT_MBUF_READ** respectively. Therefore, the wrapper provides both a class and struct version of each. The struct versions have "_STRUCT" at the end of their names.

If you don't use the struct MBUF version when calling the above functions, you will get an Execution Exception.

For an example of this, see the function **BcUserTimerCallback** in **RtTests.cs**, in the **BustoolsCsApp** project.

CHAPTER 7

Application Notes

BustoolsCsApp

The primary code for this project can be found in the Form1 and RtTests classes. Form1 implements a basic GUI, and RtTests contains code that sets up and runs a 1553 card.

BustoolsInterop

The primary code for this project can be found in the Form1 class. Although this is a Windows Forms project, the interop tests all take place in the Form1 constructor, and do not actually implement any GUI operations. Thus, the result is a blank form.

However, you can use the Visual Studio debugger to step through the constructor code to see how interop works.