WinDriver USB diagnostic GUI C# sample

The source code for this project is provided with Jungo WinDriver. To compile this application, you will need a compiler and CMake installed.

Overview

An INF must be installed in order for Windows to access and control the USB device. You can generate and install INF files via the DriverWizard, or install an existing INF file with 'wdreg.exe'. This sample allows reading and writing to pipes, changing configurations and settings, reading information about the device,opening several devices simultaneusly and performing reading speed tests.

Files

- **UsbSample.cs**
  The main file, which demonstrates accessing and controlling USB devices using the usb_lib_netcore.dll library.

- **../../lib/Release/usb_lib_netcore.dll**
  A library file which uses WinDriver High-Level APIs to access and control USB devices. These APIs are defined in wdapi1610.dll and are wrapped for .NET Core in wdapi_netcore1610.dll.

- **../AssemblyInfo.cs**
  Contains general Information about an assembly. Edit this file to modify the information associated with an assembly.

- **../DeviceTabPage.cs, ../FormChangeSettings.cs, ../FormTransfers.cs**
  GUI components which use 'Windows Forms'.

- **CMakeLists.txt**
  An input file for the CMake build system.

- **readme.pdf**
  Describes the sample files.

We provide several methods for compiling this code:

Compiling this project using Microsoft Visual Studio/Visual Studio Code

- If you are using Microsoft Visual Studio 2017 or higher, or Visual Studio Code, make sure to have CMake support installed for it.
- When you open the sample folder (with File->Open->Folder...) or open the CMakeLists.txt file (with File->Open->CMake...), Visual Studio will automatically invoke the `cmake` command to generate a CMake cache for the project. To generate cache manually, press the 'Switch between solutions and available views' button, right click on the CMake project and select 'Generate Cache'.
- Expand the CMake project - all available targets for the project will be listed.
- Right clicking on the target will allow you to build it.

Compiling using a different IDE/Compiler:

Run the following command in the terminal from the working directory of the project:

```
$ cmake . -B build
```

This will create a Unix Makefile for the project in a new sub-directory named `build`. To build it, go to that sub-directory and run:

```
$ make
```

To add verbosity to a build you can run:

```
$ VERBOSE=1 make
```

or if you prefer the build to always be verbose you can generate the CMake cache in the following way:

```
$ cmake . -B build -DCMAKE_VERBOSE_MAKEFILE=ON
```

You can use CMake to generate projects for various other platforms and IDEs. Consult CMake's documentation for more information.

Compiling this project using `dotnet`:

Build

Run the following command from the directory containing the .csproj file:

```
$ dotnet build [project file] -p:Configuration=Release
```

This command builds the project and its dependencies and creates an executable that can be used to run the application (the hosting system must have the .NET shared runtime installed on it in order to run the executable).

Publish

Run the following command from the directory containing the .csproj file:

```
$ dotnet publish [project file] --use-current-runtime

$ dotnet publish [project file] --r <runtime id>
```

This command builds the project and its dependencies and creates a platform-specific executable ready for deployment on a host system. Please note that it is not necessary for for the .NET shared runtime to be installed on the host system for the executable to work.