WinDriver PLX 9656 Sample

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

Files

- **p9656_diag.c**
  The main file which demonstrates access to the PLX 9656, using plx_lib.c
- **../lib/plx_lib.c**
  A library for accessing PLX devices using the WinDriver High Level APIs
- **../diag_lib/plx_diag_lib.c**
  A library for accessing PLX devices using the WinDriver High Level APIs
- **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.

Creating your own project

- Create a new project using your IDE.
- Choose console mode project.
- Include the following files in the project: `p9656_diag.c`
  `../lib/plx_lib.c`
  `../diag_lib/plx_diag_lib.c`

- Include the WinDriver Diagnostics samples shared files: `(WD_BASEDIR)/samples/c/shared/wdc_diag_lib.c` `(WD_BASEDIR)/samples/c/diag_lib.c` `$(WD_BASEDIR)` is the directory where WinDriver is installed at.

- Link your project with `$(WD_BASEDIR)/lib/wdapi<version>.lib` (Windows) or `$(WD_BASEDIR)/lib/libwdapi<version>.so` (Linux) or `$(WD_BASEDIR)/lib/libwdapi<version>.dylib` (MacOS) In order to access WinDriver's High-Level API. `$(WD_BASEDIR)` is the directory where WinDriver is installed at.
- Make sure to add the relevant flags to your system: `-DKERNEL_64BIT` if using a 64-bit operating system. `-DWD_DRIVER_NAME_CHANGE` if using a renamed driver.

Converting to a GUI application:

This sample was written as a console mode application (rather than a GUI application) that uses standard input and output. This was done in order to simplify the source code. You may change it into a GUI application by removing all calls to `printf()` and `scanf()` functions, and calling `MessageBox()` instead (on Windows). On other operating systems, you can use the relevant libraries such as GTK or Qt.