Industrial IoT

Advantech GPIO

Windows KMDF Driver

User Manual

For Windows

Version <1.00>

Revision History

Date	Version	Description
2022/07/05	0.92	Change the format of revision history
2016/08/16	0.91	Update hyperlinks
2016/02/24	0.90	Initial draft
2023/3/16	1.00	1.Software Utility 2. Installation

Table of Contents

User Manual

1. Welcome to Advantech GPIO Windows KMDF Driver

1.1 About This Manual

This manual contains the information for getting started with the Advantech GPIO Windows KMDF Driver.

This manual supplies information about driver interfaces of Advantech GPIO device, including calling procedure of operating GPIO device and descriptions of each function, parameter, and data structure.

This manual contains step-by-step instructions for building applications with the GPIO Device Driver with Microsoft Visual C++ and Microsoft Visual C++ 2015 Smart Device. With the help of Advantech GPIO Driver, you can develop applications by tools like VC++ and VC++ Smart Device in different Windows operating systems (Windows XP/7/8/8.1/10/Embedded Standard).

This manual also provides examples for Advantech GPIO Windows KMDF Driver, explaining how to use the driver with series of real examples and offering a reference for you to develop your own applications.

This manual does not show you how to solve every possible programming problem. Before getting started, you should already be familiar with at least one of the supported programming environments and Windows XP/7/8/8.1/10/Embedded Standard.

1.2 Organization of This Manual

This user manual is divided into the following sections:

- Welcome to Advantech GPIO Windows KMDF Driver
- Advantech GPIO Windows KMDF Driver Overview

- Getting Started with Advantech GPIO Windows KMDF Driver
- Programming Guide
- Function Reference
- Software Utility & Programming Examples

Welcome to Advantech GPIO Windows KMDF Driver

This section gives you a basic concept of this manual.

Advantech GPIO Windows KMDF Driver Overview

This section gives you a basic concept of Advantech GPIO Windows KMDF Driver.

Getting Started with Advantech GPIO Windows KMDF Driver

This section gives the beginner a clear concept of the Advantech GPIO Windows KMDF Driver and a walk-through in creating a simple application. Step-by-step instructions are given for an application written in MFC and Visual C++ 2015 MFC Smart Device development environments.

Programming Guide

This section shows a basic code flow for the GPIO control and management.

Functions Reference

- ***Function Description***

  This section gives a brief introduction of each function (WINDOWS Native API) used in current development.

- ***CTL_CODE***

  This section describes all the control codes the Advantech GPIO Windows KMDF driver supports.

- ***Data Structure***

  This section describes the data structures that related to the functions we provide.

Programming Examples

This section gives an overview of the examples we provide.

2. Advantech GPIO Windows KMDF Driver Overview

The Advantech GPIO Windows KMDF Driver provides functions to maximize the hardware's performance. It is freely bundled with the Advantech GPIO Device.

The driver allows you to easily perform versatile GPIO operations in programs developed with tools like Microsoft Visual C++ 6.0, Visual C++ 2015 (Smart Device), and other programming languages in different Windows operating systems. By using this Driver, you don't have to use hardware-specific register commands.

The driver also provides a sample application. You can modify the sample application to meet your needs.

The usage of KMDF is in the following aspects:

- **Driver Installation**

  You can refer to Install KMDF Driver to install the driver.

- **Driver Uninstallation**

  You can refer to Uninstall KMDF Driver to uninstall the driver.

- **Development Kit Installation**

  None.

- **Interface**

  1. The device interface name is

     \\\\.\\AdvGPIODev

     Function involved: CreateFile

2.1 Environments

2.1.1 GPIO

2.1.1.1 Hardware

It supports only Advantech IAG x86 hardware platform products with GPIO design; please see the release notes to check the support list before using it.

2.1.1.2 Operating Systems

☐ Windows Embedded Standard 2009
☐ 32-bit/64-bit Microsoft Windows 7/8/8.1/10
☐ 32-bit/64-Bit Windows Embedded Standard 7
☐ 32-bit/64-Bit Windows Embedded 8 Standard
☐ 32-bit/64-Bit Windows Embedded 8.1 Industry Pro
☐ 32-bit/64-Bit Windows 10 Enterprise 2015 LTSB

2.1.1.3 Common Driver

The GPIO Driver is based on common driver (AdvCOMMON).

2.2 Product Features

The Advantech GPIO Windows KMDF driver mainly includes the following features:

- **GPIO Information:**
  - **GPIO Count:**
    Driver will enumerate GPIO and counts the total number of GPIO pins.
  - **GPIO Direction:**
    Gets the current direction setting of each GPIO pin. Input(1) or Output(0)
  - **GPIO Status:**
    Reads the current status of each GPIO pin. 1-On/High or 0-Off/Low.

- **GPIO Configuration**

- **GPIO Direction Control**
  GPIO can be configured as either Input (1) or Output (0).
- **GPIO Status Configuration**
  Programming the 1-On/High or 0-Off/Low status of the output type GPIO.

- **GPIO Tool Example Source Code**
  - A tool to control GPIO.
    You can use it to configure the GPIO.
  - Example programs
    The example programs can be used for the reference of software development.

2.3 Installation

2.3.1 Install KMDF Driver

Installation is required. If there is no existing installation of Advantech GPIO Windows KMDF driver on your computer, take the following steps to install Advantech GPIO Windows KMDF driver.

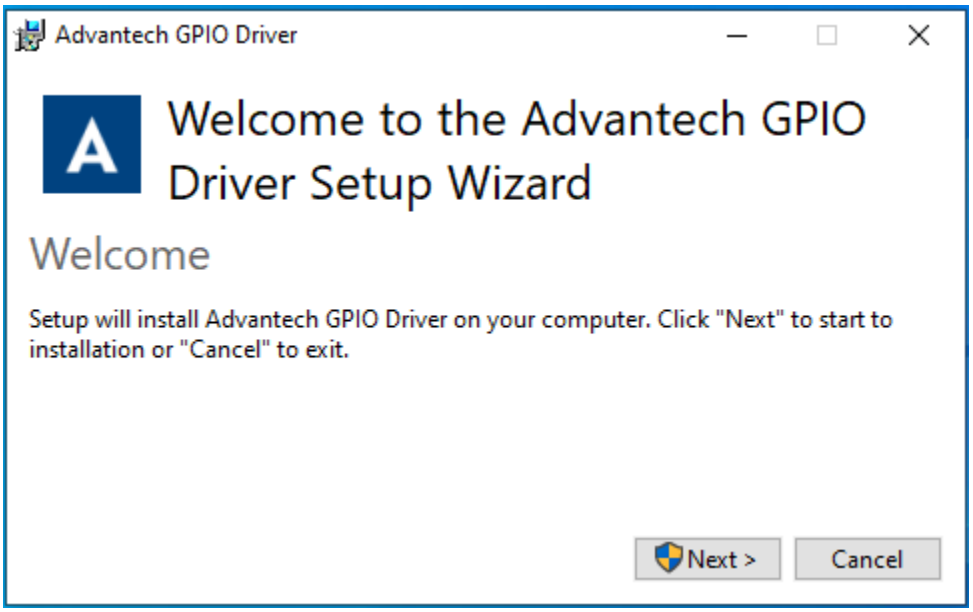
How to install Advantech GPIO Windows KMDF driver

1) Verify that your computer meets the hardware and software requirements to run Advantech GPIO Windows KMDF driver.
   For more information, see Environments.
2) If you do not already have a copy of the installer Advantech GPIO Windows KMDF driver, download the installer.
3) From Control Panel, remove any existing installation of Advantech AdvEC driver and GPIO driver from your computer.
4) With administrator-level privilege on your computer, run the installer for Advantech GPIO Windows KMDF driver.

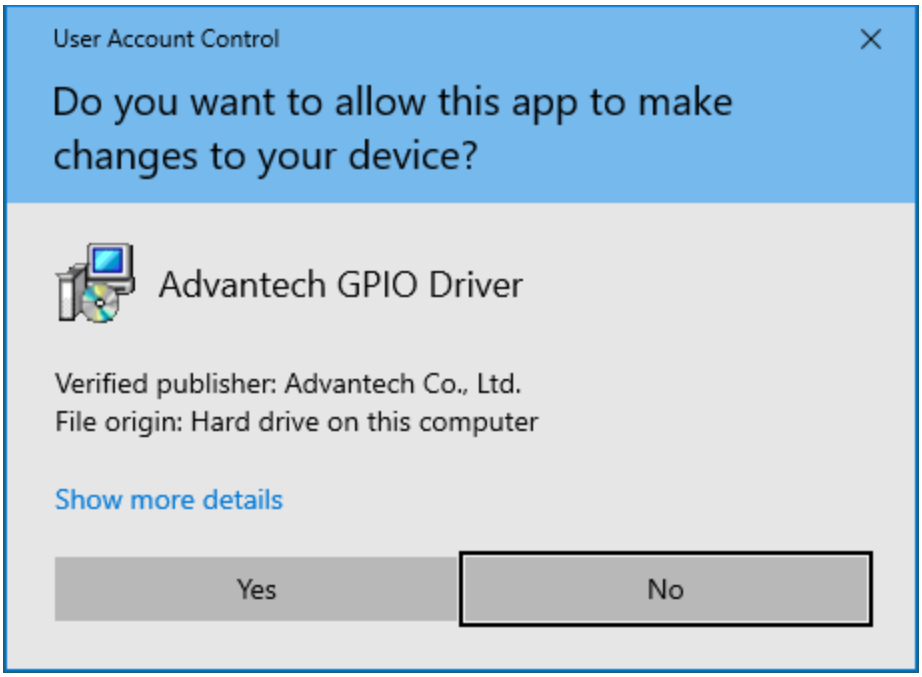
Below is an example of Advantech GPIO Windows KMDF driver Setup. If you want to stop the setup, press the "Cancel" button in the setup program. The Setup program will stop the procedure automatically.

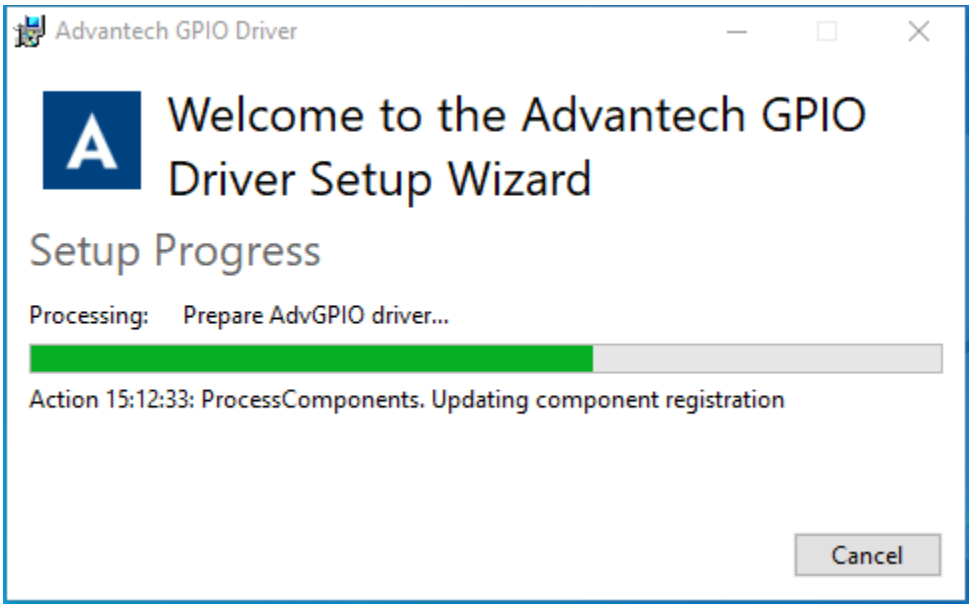
1. Run the Setup program.

2. When the setup program is running, click the “Next” button.

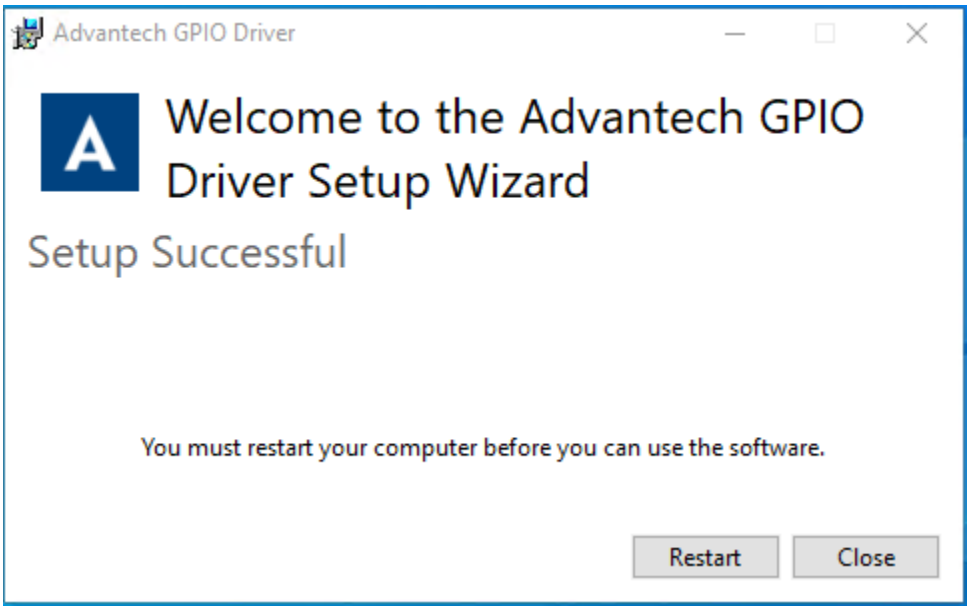


3. Allow this app to make changes, answer “YES”. And wait for completion.





4. Click the "Restart" button to finish the installation of Advantech GPIO Windows KMDF driver.

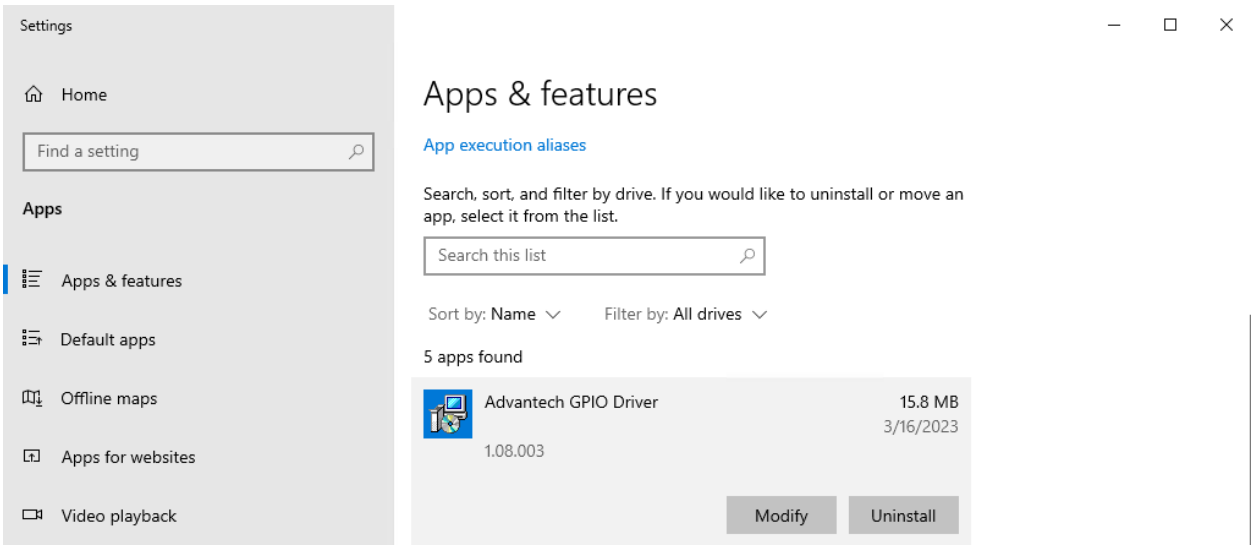


2.4 Uninstallation

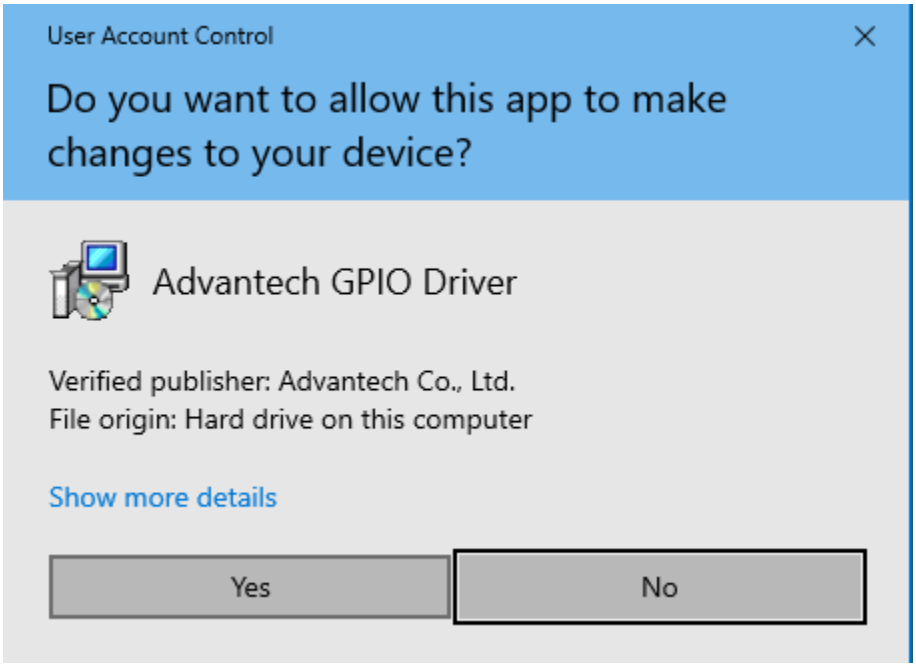
2.4.1 Uninstall KMDF Driver

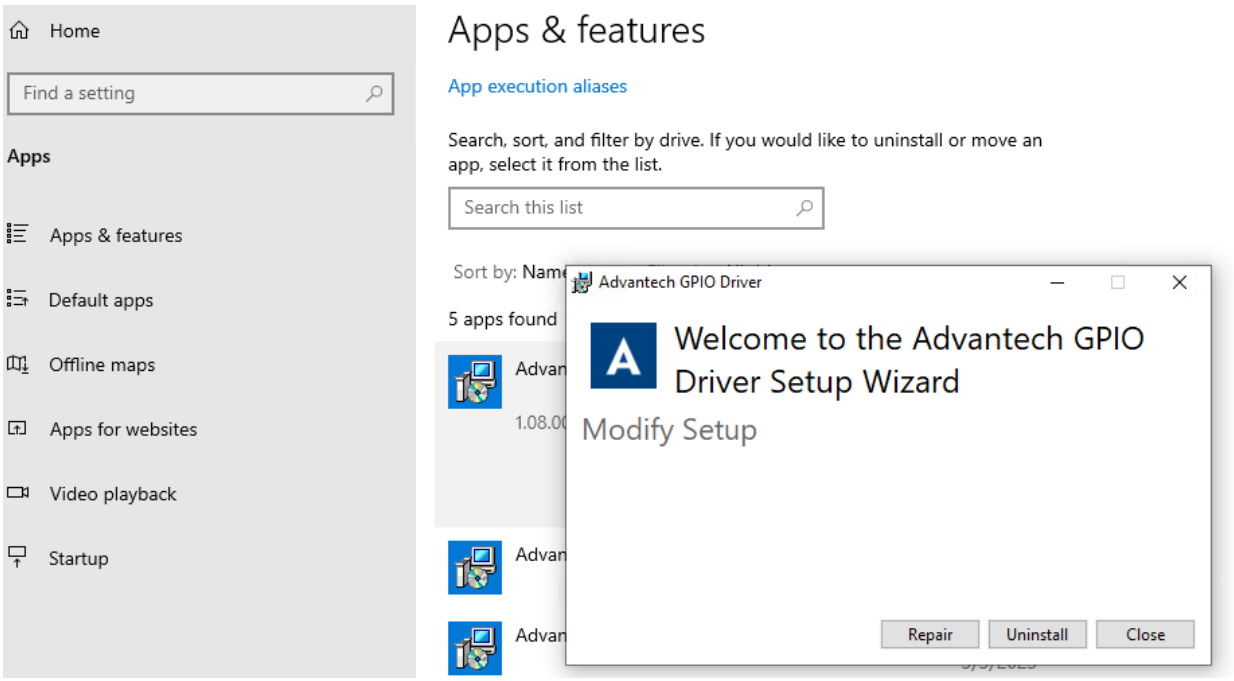
How to uninstall Advantech GPIO Windows KMDF driver

1. Control panel ->”App & features”. Choose the Advantech GPIO Driver to Uninstall it.

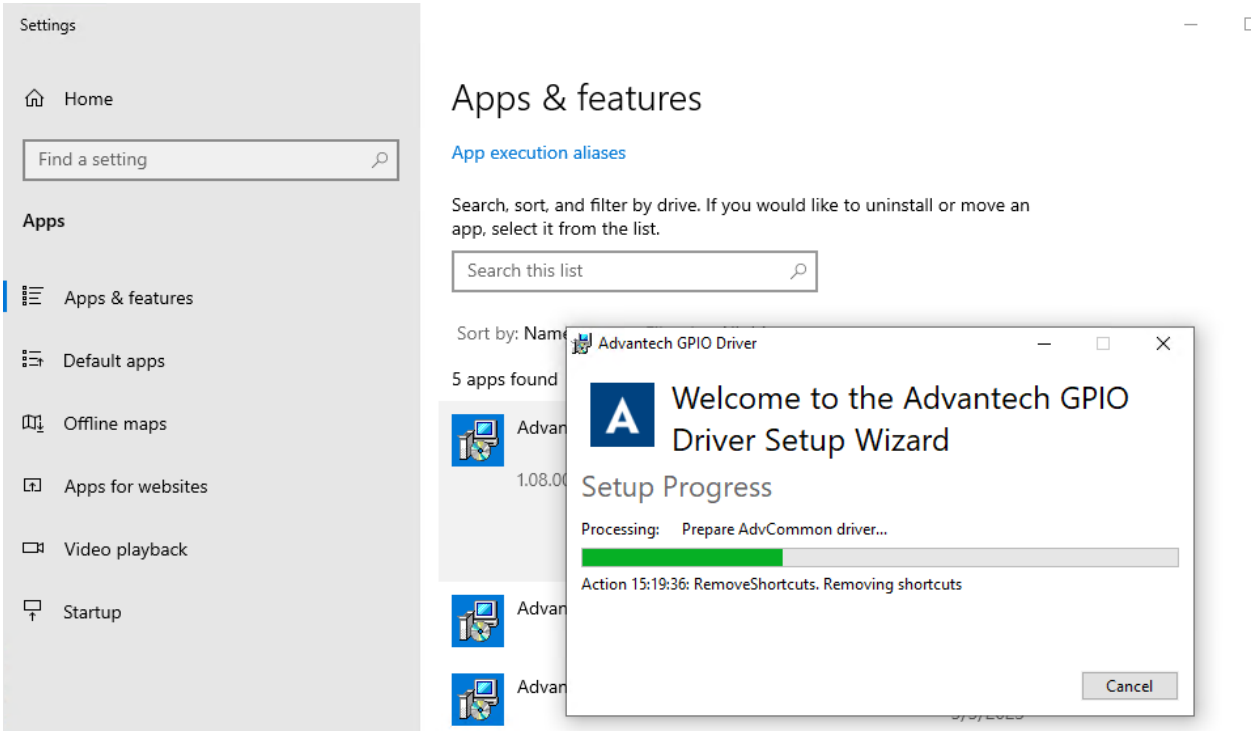


2. Allow this app to make changes ... , answer “Yes”, then click “Uninstall”

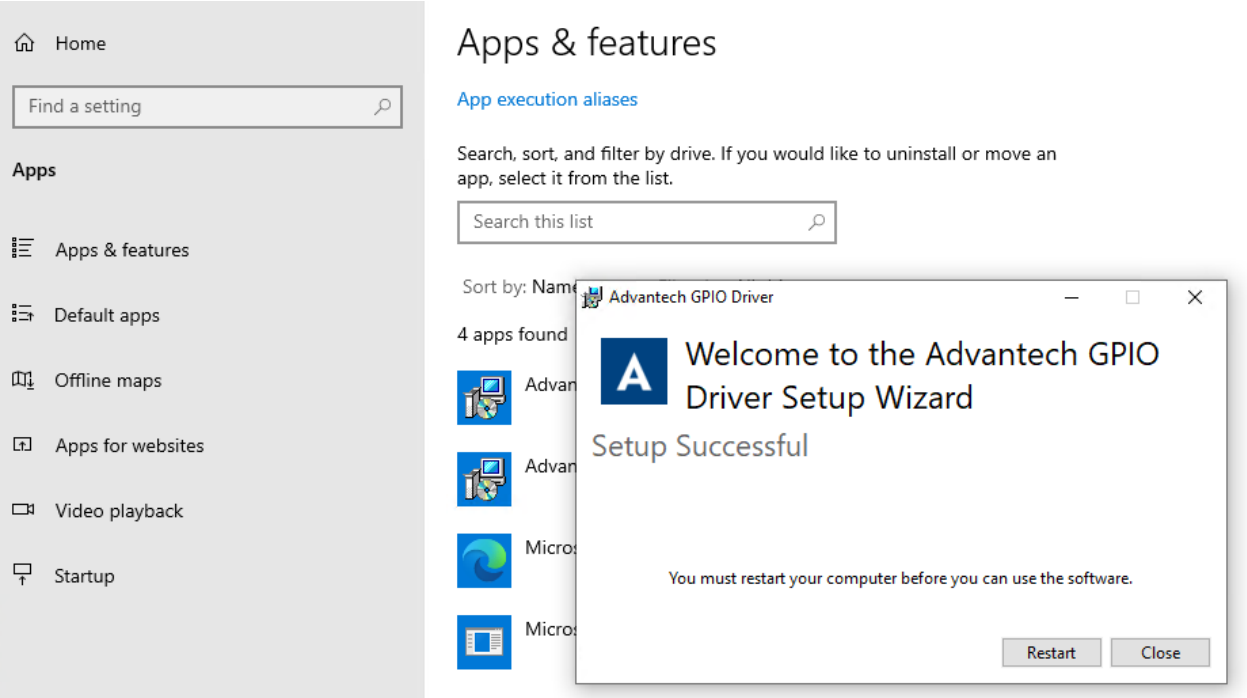




3. The uninstallation is running. Please wait for completion.



4. Click the "Restart" button to finish the uninstallation of Advantech GPIO Windows KMDF driver.



3. Getting Started with Advantech GPIO Windows KMDF Driver

This chapter provides a step-by-step example to demonstrate how to build an application using Advantech GPIO Windows KMDF Driver from scratch in Microsoft Visual C++ 6.0 and Microsoft Visual Studio 2015.

The following is the necessary file for programming:

- AdvGPIO_IOCTL.h: Function declaration, constant definition for Microsoft Visual C++ 6.0 or Microsoft Visual Studio 2015 Smart Device Project.

3.1 For Microsoft Visual C++

3.1.1 Create an Empty Visual C++ Project

To use the GPIO functions, follow this procedure:

1. Create your source files as you would for other Windows programs written in C++ by

calling DLL functions as typical function calls.

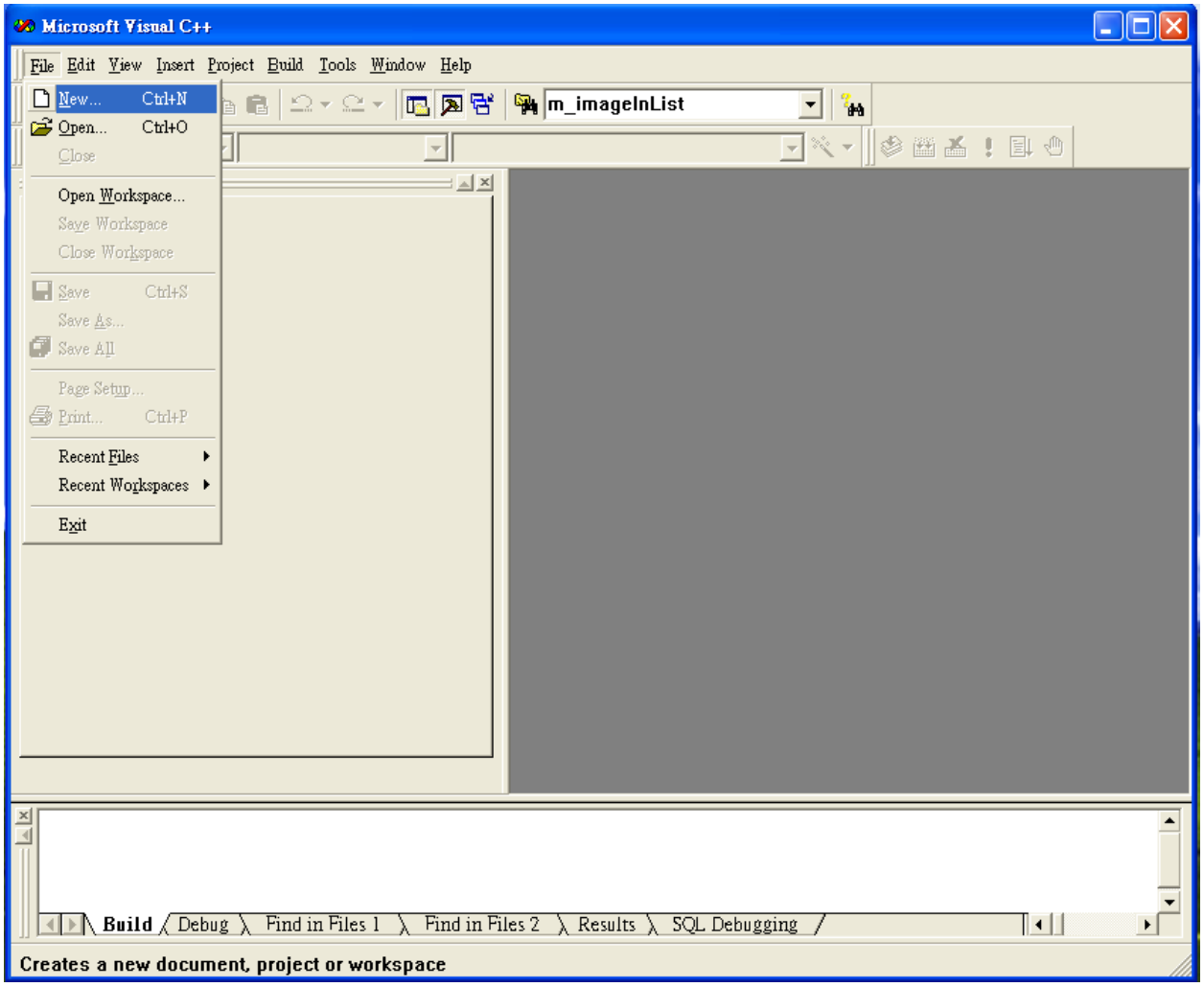
2. Include the header file, as shown in the following example:

#include "AdvGPIO_IOCTL.h"

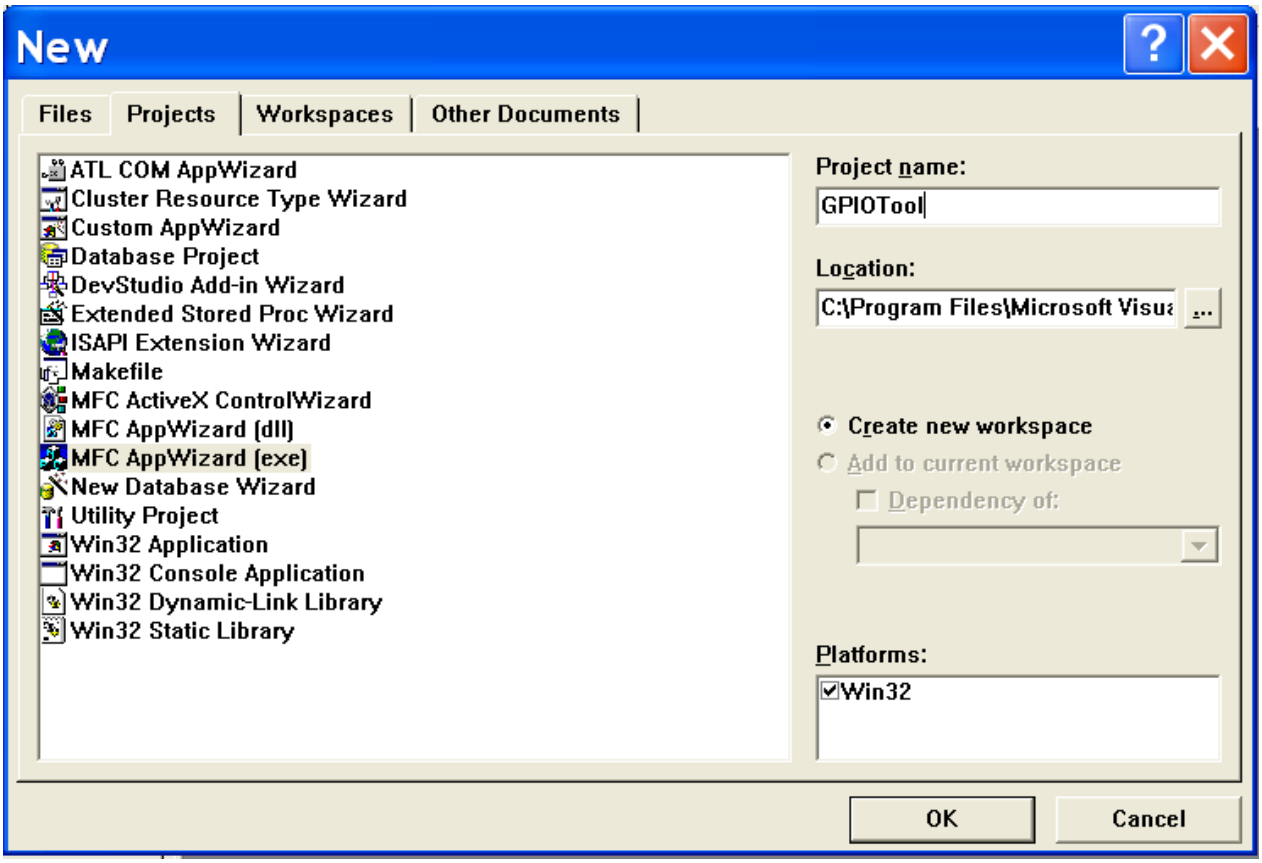
(Installation C:\Program Files\Advantech\AdvGPIO\Examples\VC++\AdvGPIOTool \AdvGPIO_IOCTL.h)

For a general outline of creating a Visual C++ Windows programs, complete the following procedure:

1. Click File/New from the main menu to create your application project and source code as you work on any other Visual C++ program.



2. Define the type of new project as "MFC AppWizard (exe)", and assign a project file directory

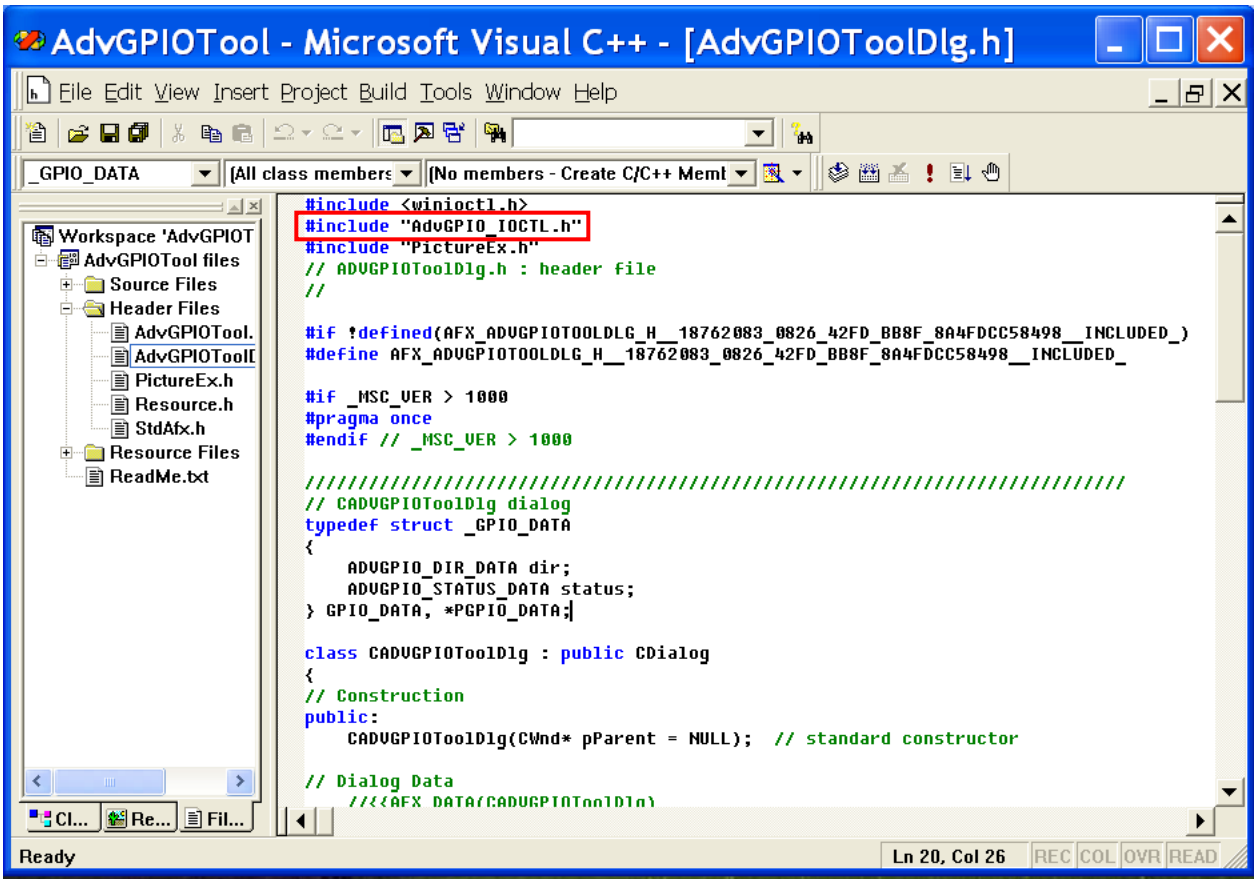


Run through the wizard to create the new project from Empty.

3.1.2 Adding Necessary File

In order to develop GPIO applications for Advantech GPIO Windows KMDF Driver, you have to firstly add necessary file.

1. Include the Advantech GPIO Windows KMDF Driver for Visual C++ header files (AdvGPIO_IOCTL.h). The header file is located in where your KMDF example installed, like the following example:
C:\Program Files\Advantech\GPIO\Examples\VC++\AdvGPIOTool.



After adding the header file, you can view the GPIO constant definition, parameter declaration, and IO control codes that are defined in this header file. These definitions can all be used in your application programs.

3.1.3 Writing Codes

Write your application source code. For more detailed program development information, please refer to the Visual C++ User's Manual.

3.1.4 Test Your Program

1. Click on Compile under the Build menu to compile your code.

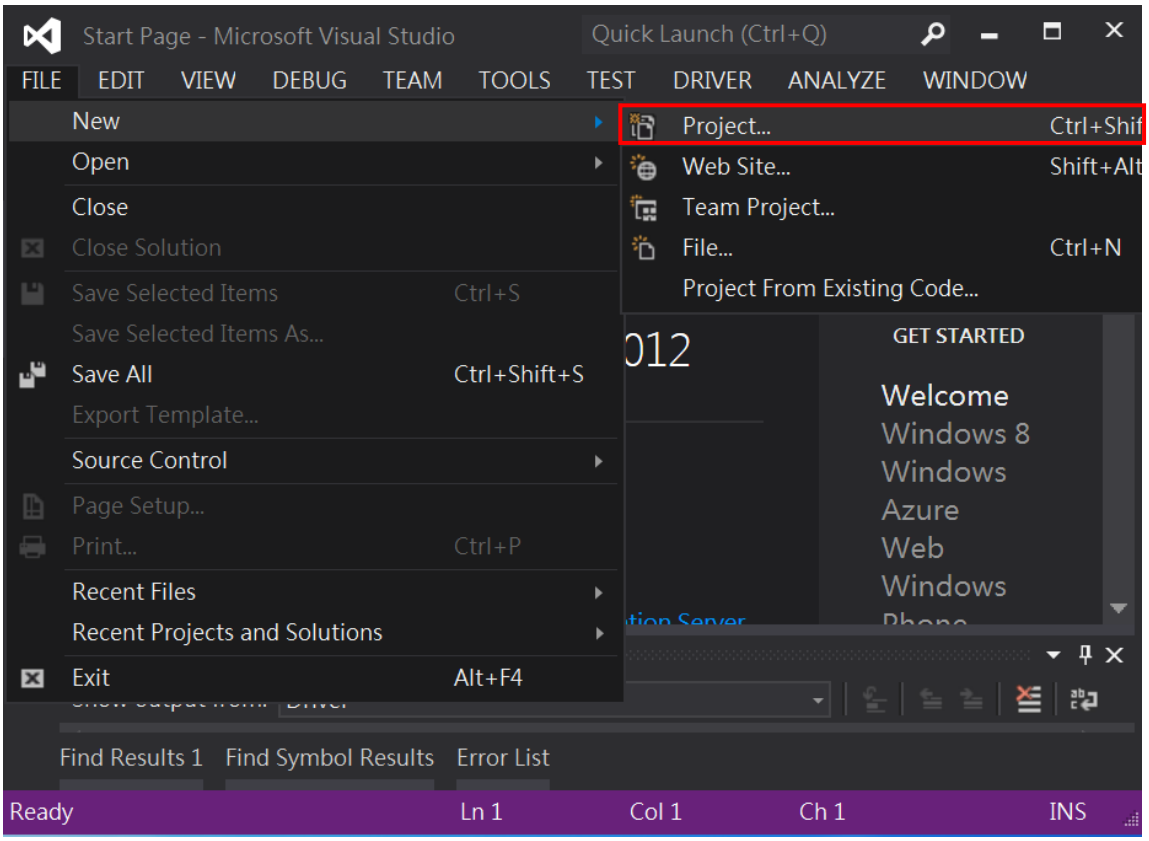
2. Run your saved ***.exe on you target platform.

3.2 For Microsoft Visual Studio 2015 Smart Device

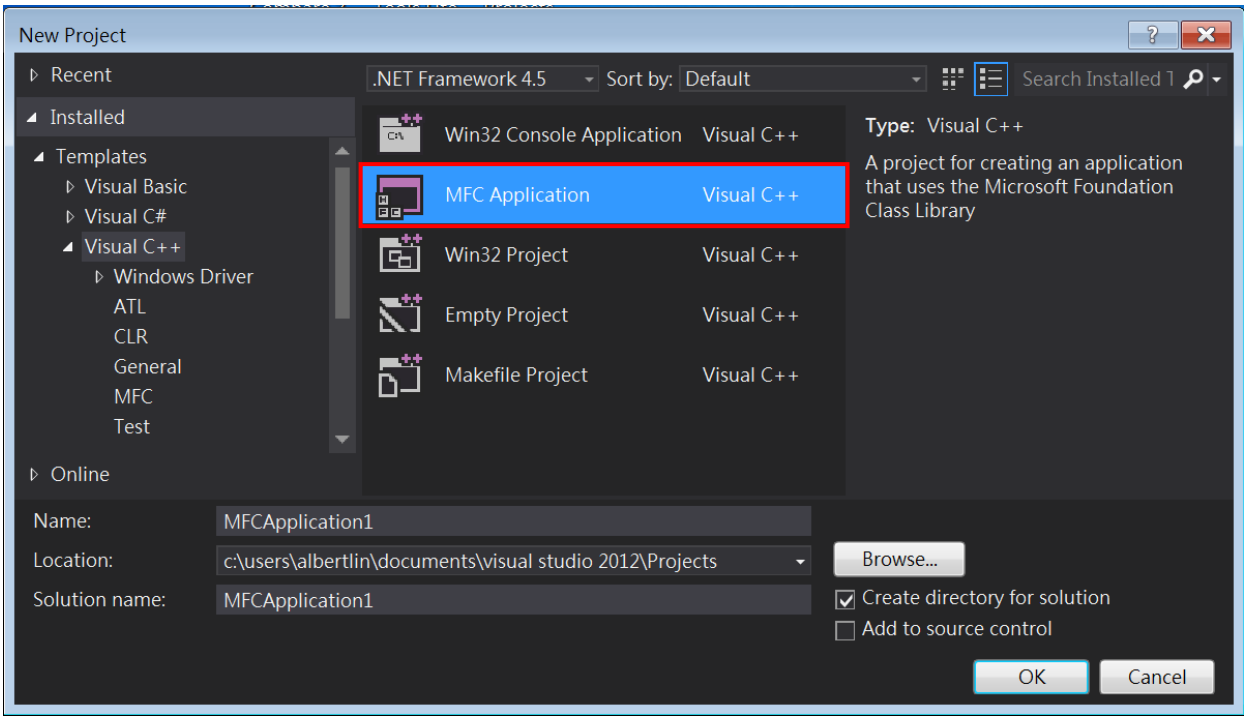
3.2.1 Create an Empty Virtual C++ Smart Device Project

For a general outline of creating a Virtual C++ Smart Device programs, complete the following procedure:

1. Click File/New from the main menu to create your application project and source code as you would for any other Visual C++ Smart Device program.



2. Define the type of new project as "MFC Application", assign a project file directory

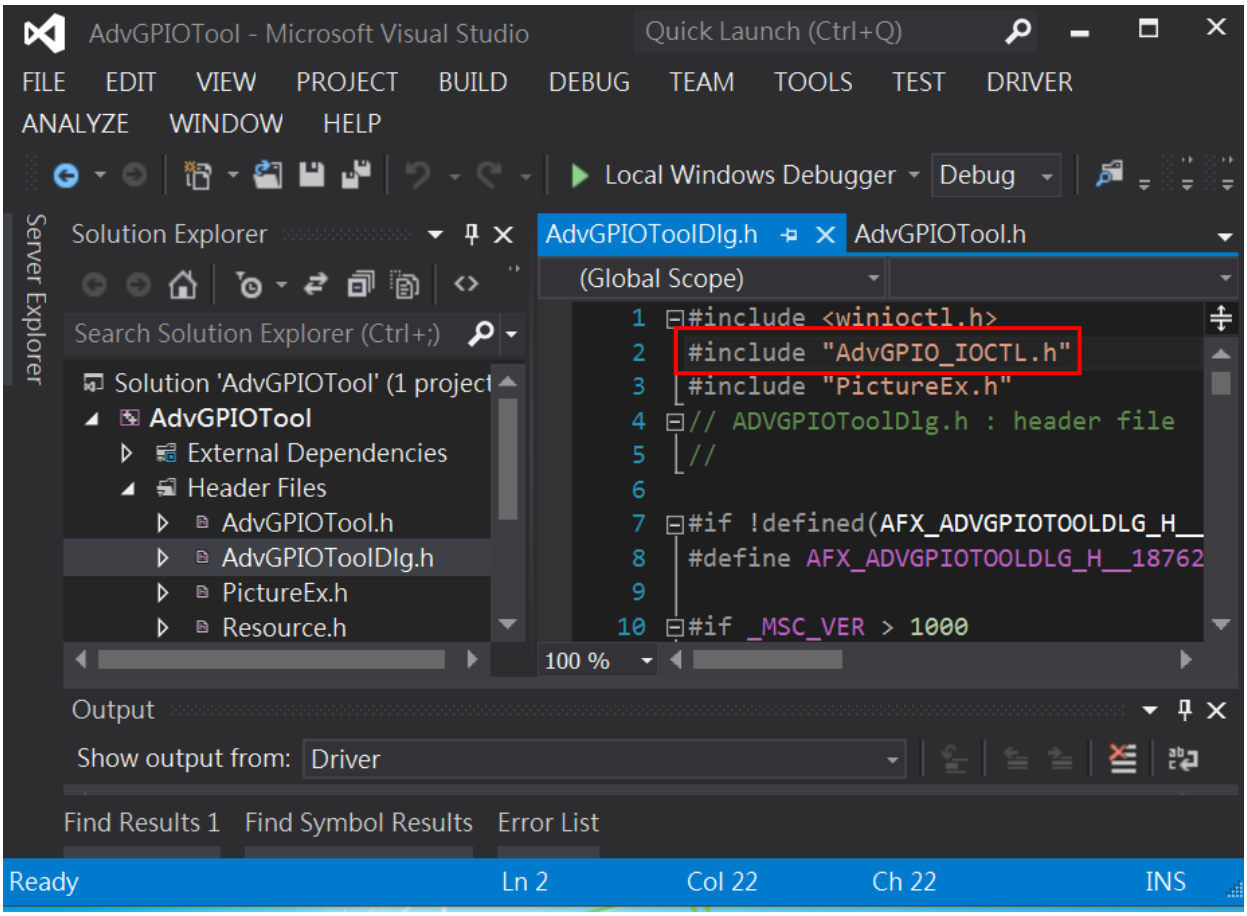


Run through the wizard to create the new project from Empty.

3.2.2 Include Necessary File

In order to develop GPIO applications with Advantech GPIO Windows KMDF Drivers, you have to firstly add necessary file.

1. Include the Advantech GPIO Windows KMDF Driver for Visual C++ Smart Device header files AdvGPIO_IOCTL.h. The header file is located in where your SDK installed, like following example:
C:\Program Files\Advantech\GPIO\Examples\VC++\AdvGPIOTool.



After adding the header file, you can view the GPIO constant definition, parameter declaration, and IO control codes that are defined in this header file. These definitions can all be used in your application programs.

3.2.3 Writing Codes

Write your application source code. For more detailed program development information, please refer to the Microsoft Visual Studio 2015 User's Manual.

3.2.4 Test Your Program

1. Click on Compile under the Build menu to compile your code.

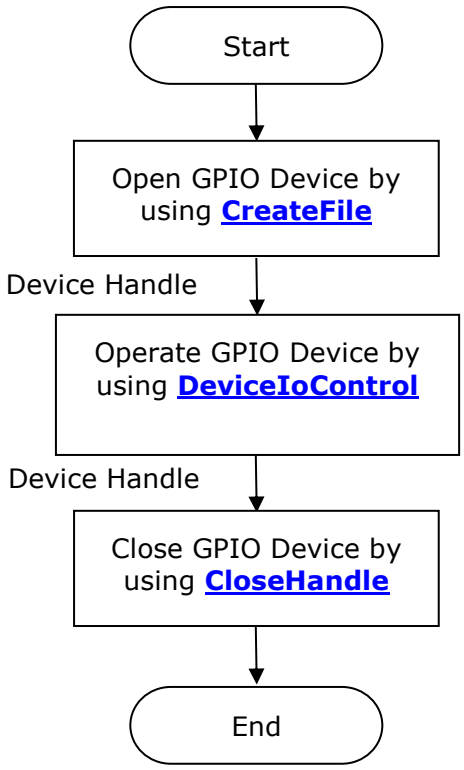
2. Run your saved ***.exe on you target platform.

4. Programming Guide

User can directly access drivers with WINDOWS Native API. In the following, we will provide an example by opening GPIO device and reading its current status to explain how to write basic applications in VC environment. Necessary files for developing applications are listed below. Suppose installation paths of all header files in the example are C:\Program Files\Advantech\GPIO\Examples\VC++\AdvGPIOTool.

Device Function Group

The following figure describes the common call flow of the GPIO which is necessary for all GPIO operation:



5. Function Reference

Advantech's GPIO Windows KMDF Driver contains a set of control codes and associated structures that can be used in various applications. The control codes support many development environments and programming languages, including Microsoft Visual C++ Program and Microsoft Visual C++ Program.

5.1 Function Description

You can manipulate GPIO through the WINDOWS Native APIs, thus make you use the GPIO device through their existing application and examples without any change.
In your application, use the **CreateFile** function to open GPIO device; call the **DeviceIoControl** function to send a control code directly to the Advantech GPIO Windows KMDF driver, causing the GPIO device to perform the corresponding operation; call the **CloseHandle** when operation is completed to close the opened GPIO device.

The following tables describe the main WINDOWS Native APIs are used in current development.

Item	Name	Note
1)	CreateFile	Open GPIO device.
2)	CloseHandle	Close the opened GPIO device when operation is completed.
3)	DeviceIoControl	Send a control code directly to the GPIO device driver.

Only brief introduction is given in this manual regarding detailed usage of each function. Notes are made to notify you important operation. For more detailed information about the usage, please see MSDN.

5.1.1 CreateFile

You can use the **CreateFile** function to open GPIO device. The function returns a handle that can be used to access the GPIO device.

Syntax

HANDLE CreateFile(

LPCTSTR *lpFileName*,

DWORD *dwDesiredAccess*,

DWORD *dwShareMode*,

LPSECURITY_ATTRIBUTES *lpSecurityAttributes*,

DWORD *dwCreationDisposition*,

DWORD *dwFlagsAndAttributes*,

HANDLE *hTemplateFile*

);

Parameters

Name	Direction	Description
lpFileName	Input	[in] A pointer to a null-terminated string that specifies the name of the GPIO device to open. ***Note** Use \\\\.\\AdvGPIODev.
dwDesiredAccess	Input	[in] The access to the GPIO device, Ways of opening the GPIO device, which is usually GENERIC_READ | GENERIC_WRITE.
dwShareMode	Input	[in] The sharing modes of the GPIO device, which can be read, write, both, or none. Which is usually FILE_SHARE_READ | FILE_SHARE_WRITE.
lpSecurityAttributes	Input	[in] A pointer to a **SECURITY_ATTRIBUTES** structure that determines whether or not the returned handle can be inherited by child processes. ***Note** The handle cannot be inherited. It must be set to **NULL**.
dwCreationDisposition	Input	[in] An action to take on files that exist and do not exist, which is usually **OPEN_EXISTING**.
dwFlagsAndAttributes	Input	[in] The file attributes and flags. ***Note** The GPIO device is not being opened or created for asynchronous I/O. It must be set to **0**.
hTemplateFile	Input	NULL

Return Value

If the function succeeds, the return value is an open handle to the GPIO device. If the function fails, the return value is INVALID_HANDLE_VALUE. To get extended error information, call **GetLastError** function.

Remarks

Use the **CloseHandle** function to close the opened GPIO device handle that **CreateFile** returns when operation is completed.

Example Code

```
#include "Tchar.h"
#include "wtypes.h"
#include "winioctl.h"
#include "AdvGPIO_IOCTL.h"

// -----------------------------------------------------------------------
// DESCRIPTION: Open the GPIO Device
// -----------------------------------------------------------------------
//-------------------------------------------------------------
// Function    : GPIO_DeviceOpen
//
// PURPOSE     : Open the GPIO Device
//
// Parameters  : DriverHandle (OUT)
//                              Handle of device
//
// Return      : NULL and DriverHandle (success)
//
//-------------------------------------------------------------
HANDLE GPIO_DeviceOpen()
{
        HANDLE DriverHandle = NULL;
        TCHAR GPIOName[20] = TEXT("\\\\.\\AdvGPIODev");

        DriverHandle = CreateFile(
```

```
                GPIOName,
                GENERIC_READ | GENERIC_WRITE,
                FILE_SHARE_READ | FILE_SHARE_WRITE,
                NULL,
                OPEN_EXISTING,
                0,
                NULL );

        return DriverHandle;
}
```

See Also

CloseHandle

5.1.2 CloseHandle

Close the GPIO device by calling this function when operation is completed.

Syntax

```
BOOL CloseHandle(
  HANDLE hObject
);
```

Parameters

Name	Direction	Description
hObject	Input	[in] Handle to the GPIO device which was opened.

Return Value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError** function.

Example Code

```
#include "Tchar.h"
```

```
#include "wtypes.h"
#include "winioctl.h"
#include "AdvGPIO_IOCTL.h"

// -------------------------------------------------------------------------
// DESCRIPTION: Close the GPIO Device
// -------------------------------------------------------------------------
//-------------------------------------------------------------------
// Function    : GPIO_DeviceClose
//
// PURPOSE     : Close GPIO Device by handle.
//
// Parameters  : DriverHandle (IN)
//             Handle of device
//
// Return      : TRUE (success)
//
//-------------------------------------------------------------------
BOOL GPIO_DeviceClose ( HANDLE DriverHandle )
{
        if (DriverHandle != INVALID_HANDLE_VALUE)
        {
                CloseHandle(DriverHandle);

                // reset DeviceHandle
                DriverHandle = NULL;
        }

        return TRUE;
}
```

See Also

CreateFile

5.1.3 DeviceIoControl

User can use the **DeviceIoControl** function to send a control code directly to the GPIO device driver, causing the GPIO device to perform the corresponding operation. Such as configure GPIO direction, get GPIO current direction, configure GPIO status, get current GPIO status, etc.

Syntax

```
BOOL DeviceIoControl(
  HANDLE hDevice,
  DWORD dwIoControlCode,
  LPVOID lpInBuffer,
  DWORD nInBufferSize,
  LPVOID lpOutBuffer,
  DWORD nOutBufferSize,
  LPDWORD lpBytesReturned,
  LPOVERLAPPED lpOverlapped
);
```

Parameters

Name	Direction	Description
hDevice	Input	[in] Handle to the GPIO device on which the operation is to be performed. To retrieve a GPIO device handle, use the **CreateFile** function
dwIoControlCode	Input	[in] Control code for the specific operation. This value identifies the specific operation to be performed. For a list of the supported control codes, see **CTL_CODE**.
lpInBuffer	Input	[in] Pointer to the input buffer that contains the data required to perform the operation. This parameter can be NULL if *dwIoControlCode* specifies an operation that does not require input data.
nInBufferSize	Input	[in] Size of the input buffer, in bytes.
lpOutBuffer	Output	[out] Pointer to the output buffer that is to receive the

		data returned by the operation. This parameter can be NULL if *dwIoControlCode* specifies an operation that does not return data.
nOutBufferSize	Input	[in] Size of the output buffer, in bytes.
lpBytesReturned	Output	[out] Pointer to a variable that receives the size of the data stored in the output buffer, in bytes.
lpOverlapped	Input	[in] Pointer to an OVERLAPPED structure. ***Note** *lpOverlapped* must be set to **NULL**.

Return Value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError** function.

5.2 CTL_CODE

The following tables describe all the control code the GPIO device driver support.

Item	Name	Note
1)	IOCTL_ADVGPIO_GET_COUNT	Gets the GPIO count.
2)	IOCTL_ADVGPIO_GET_DIR	Gets the current direction configuration of the specified GPIO.
3)	IOCTL_ADVGPIO_SET_DIR	Configure the specified GPIO direction.
4)	IOCTL_ADVGPIO_GET_STATUS	Gets the current state of the specified GPIO.
5)	IOCTL_ADVGPIO_SET_STATUS	Sets the state of the specified GPIO.

5.2.1 IOCTL_ADVGPIO_GET_COUNT

The **IOCTL_ADVGPIO_GET_COUNT** control code gets the GPIO count.

To perform this operation, call the **DeviceIoControl** function with the following parameters.

```
BOOL DeviceIoControl(
  (HANDLE) hDevice,                 // handle to device
  IOCTL_ADVGPIO_GET_COUNT,          // dwIoControlCode
  NULL,                             // lpInBuffer
  0,                                // nInBufferSize
  (LPVOID) lpOutBuffer,             // output buffer
  (DWORD) nOutBufferSize,           // size of output buffer
  (LPDWORD) lpBytesReturned,        // number of bytes returned
  NULL,                             // OVERLAPPED structure
);
```

Parameters

hDevice

[in] Handle to the GPIO device. To obtain an GPIO device handle, call the **CreateFile** function.

dwIoControlCode

[in] Control code for the operation. Use **IOCTL_ADVGPIO_GET_COUNT** for this operation.

lpInBuffer

Not used with this operation; set to NULL.

nInBufferSize

Not used with this operation; set to zero.

lpOutBuffer

[out] Pointer to an integer (int) buffer.

nOutBufferSize

[in] Size of the output buffer, in bytes.

lpBytesReturned

[out] Pointer to a variable that receives the size of the data stored in the output buffer, in bytes.

lpOverlapped

NULL. *lpOverlapped* must be set to **NULL**.

Return Values

If the operation succeeds, **DeviceIoControl** returns a nonzero value.

If the operation fails, **DeviceIoControl** returns zero. To get extended error information, call **GetLastError** function.

Example Code

```
#include "Tchar.h"
#include "wtypes.h"
#include "winioctl.h"
#include "AdvGPIO_IOCTL.h"
// -------------------------------------------------------------------------
// DESCRIPTION: Gets GPIO count.
// -------------------------------------------------------------------------
int GPIO_GetCount ( HANDLE DriverHandle )
{
        int   mGpioCount = 0;

        DWORD dwReturn = 0;
        BOOL  bRet = DeviceIoControl(
                DriverHandle,
                IOCTL_ADVGPIO_GET_COUNT,
                NULL,
                0,
                &mGpioCount,
                sizeof( mGpioCount ),
                &dwReturn,
                NULL );

        return bRet ? mGpioCount : 0;
}
```

Requirements

Header: Declared in AdvGPIO_IOCTL.h.

See Also

DeviceIoControl

5.2.2 IOCTL_ADVGPIO_GET_DIR

The **IOCTL_ADVGPIO_GET_DIR** control code to get the current GPIO direction.

The input parameter structure, **ADVGPIO_DIR_DATA**, indicates GPIO direction information and specified the index of GPIO.

The output parameter structure, **ADVGPIO_DIR_DATA**, report the specified GPIO direction information.

To perform this operation, call the **DeviceIoControl** function with the following parameters.

```
BOOL DeviceIoControl(
  (HANDLE) hDevice,              // handle to device
  IOCTL_ADVGPIO_GET_DIR,         // dwIoControlCode
  (LPVOID) lpInBuffer,           // lpInBuffer
  (DWORD) nInBufferSize,         // nInBufferSize
  (LPVOID) lpOutBuffer,          // output buffer
  (DWORD) nOutBufferSize,        // size of output buffer
  (LPDWORD) lpBytesReturned,     // number of bytes returned
  NULL,                          // OVERLAPPED structure
);
```

Parameters

hDevice

[in] Handle to the GPIO device. To obtain a GPIO device handle, call the **CreateFile** function.

dwIoControlCode

[in] Control code for the operation. Use **IOCTL_ADVGPIO_GET_DIR** for this operation.

lpInBuffer

[in] Pointer to an **ADVGPIO_DIR_DATA** structure.

nInBufferSize

[in] Size of the input buffer, in bytes.

lpOutBuffer

[out] Pointer to an **ADVGPIO_DIR_DATA** structure.

nOutBufferSize

[in] Size of the output buffer, in bytes.

lpBytesReturned

[out] Pointer to a variable that receives the size of the data stored in the output buffer, in bytes.

lpOverlapped

NULL. *lpOverlapped* must be set to **NULL**.

Return Values

If the operation succeeds, **DeviceIoControl** returns a nonzero value.

If the operation fails, **DeviceIoControl** returns zero. To get extended error information, call **GetLastError** function.

Example Code

```
#include "Tchar.h"
#include "wtypes.h"
#include "winioctl.h"
#include "AdvGPIO_IOCTL.h"
// -------------------------------------------------------------------------
// DESCRIPTION: Gets the specified GPIO direction.
// -------------------------------------------------------------------------
BOOL GPIO_GetDirection ( HANDLE DriverHandle, UINT Index, BOOL *pbDir )
{
        ADVGPIO_DIR_DATA dirData = {0};



        dirData.uPinNumber = Index;

        DWORD dwReturn = 0;
        BOOL   bRet = DeviceIoControl(
                DriverHandle,
                IOCTL_ADVGPIO_GET_DIR,
                &dirData,
                sizeof( dirData ),
                &dirData,
```

```
                sizeof( dirData ),
                &dwReturn,
                NULL );

        *pbDir = dirData.bDir;
        return bRet;
}
```

Requirements

Header: Declared in AdvGPIO_IOCTL.h.

See Also

DeviceIoControl

5.2.3 IOCTL_ADVGPIO_SET_DIR

The **IOCTL_ADVGPIO_SET_DIR** control code to set the current GPIO direction. The input parameter structure, **ADVGPIO_DIR_DATA**, indicates which GPIO is going to set and the new direction.

To perform this operation, call the **DeviceIoControl** function with the following parameters.

```
BOOL DeviceIoControl(
  (HANDLE) hDevice,              // handle to device
 IOCTL_ADVGPIO_SET_DIR,          // dwIoControlCode
  (LPVOID) lpInBuffer,           // lpInBuffer
  (DWORD) nInBufferSize,         // nInBufferSize
  NULL,                          // output buffer
  0,                             // size of output buffer
  (LPDWORD) lpBytesReturned,     // number of bytes returned
  NULL,                          // OVERLAPPED structure
);
```

Parameters

hDevice

[in] Handle to the GPIO device. To obtain an GPIO device handle, call the **CreateFile** function.

dwIoControlCode

[in] Control code for the operation. Use **IOCTL_ADVGPIO_SET_DIR** for this operation.

lpInBuffer

[in] Pointer to an **ADVGPIO_DIR_DATA** structure.

nInBufferSize

[in] Size of the input buffer, in bytes.

lpOutBuffer

Not used with this operation; set to NULL.

nOutBufferSize

Not used with this operation; set to zero.

lpBytesReturned

[out] Pointer to a variable that receives the size of the data stored in the output buffer, in bytes.

lpOverlapped

NULL. *lpOverlapped* must be set to **NULL**.

Return Values

If the operation succeeds, **DeviceIoControl** returns a nonzero value.

If the operation fails, **DeviceIoControl** returns zero. To get extended error information, call **GetLastError** function.

Example Code

```
#include "Tchar.h"
#include "wtypes.h"
#include "winioctl.h"
#include "AdvGPIO_IOCTL.h"
// -------------------------------------------------------------------------
// DESCRIPTION: Sets the specified GPIO direction.
// -------------------------------------------------------------------------
BOOL GPIO_SetDirection ( HANDLE DriverHandle, UINT Index, BOOL bDir )
{
```

```
        ADVGPIO_DIR_DATA dirData = {0};

        dirData.uPinNumber = Index;
        dirData.bDir = bDir;

        DWORD dwReturn = 0;
        BOOL  bRet = DeviceIoControl(
                DriverHandle,
                IOCTL_ADVGPIO_SET_DIR,
                &dirData,
                sizeof( dirData ),
                NULL,
                0,
                &dwReturn,
                NULL );

        return bRet;
}
```

Requirements

Header: Declared in AdvGPIO_IOCTL.h.

See Also

DeviceIoControl

5.2.4 IOCTL_ADVGPIO_GET_STATUS

The **IOCTL_ADVGPIO_GET_STATUS** control code to get the current GPIO status.
The input parameter structure, **ADVGPIO_STATUS_DATA**, indicates which GPIO status is going to get.
The output parameter structure, **ADVGPIO_STATUS_DATA**, report the specified GPIO status.

To perform this operation, call the **DeviceIoControl** function with the following parameters.

```
BOOL DeviceIoControl(
```

```
  (HANDLE) hDevice,                 // handle to device
  IOCTL_ADVGPIO_GET_STATUS,         // dwIoControlCode
  (LPVOID) lpInBuffer,              // lpInBuffer
  (DWORD) nInBufferSize,            // nInBufferSize
  (LPVOID) lpOutBuffer,             // output buffer
  (DWORD) nOutBufferSize,           // size of output buffer
  (LPDWORD) lpBytesReturned,        // number of bytes returned
  NULL,                             // OVERLAPPED structure
);
```

Parameters

hDevice

[in] Handle to the GPIO device. To obtain an GPIO device handle, call the **CreateFile** function.

dwIoControlCode

[in] Control code for the operation. Use **IOCTL_ADVGPIO_GET_STATUS** for this operation.

lpInBuffer

[in] Pointer to an **ADVGPIO_STATUS_DATA** structure.

nInBufferSize

[in] Size of the input buffer, in bytes.

lpOutBuffer

[out] Pointer to an **ADVGPIO_STATUS_DATA** structure.

nOutBufferSize

[in] Size of the output buffer, in bytes.

lpBytesReturned

[out] Pointer to a variable that receives the size of the data stored in the output buffer, in bytes.

lpOverlapped

NULL. *lpOverlapped* must be set to **NULL**.

Return Values

If the operation succeeds, **DeviceIoControl** returns a nonzero value.

If the operation fails, **DeviceIoControl** returns zero. To get extended error information, call **GetLastError** function.

Example Code

```
#include "Tchar.h"
#include "wtypes.h"
#include "winioctl.h"
#include "AdvGPIO_IOCTL.h"
// -------------------------------------------------------------------------
// DESCRIPTION: Gets the specified GPIO status.
// -------------------------------------------------------------------------
BOOL GPIO_GetStatus ( HANDLE DriverHandle, UINT Index, BOOL *pbStatus )
{
        ADVGPIO_STATUS_DATA statusData = {0};


        statusData.uPinNumber = Index;

        DWORD dwReturn = 0;
        BOOL  bRet = DeviceIoControl(
                DriverHandle,
                IOCTL_ADVGPIO_GET_STATUS,
                &statusData,
                sizeof(statusData),
                &statusData,
                sizeof(statusData),
                &dwReturn,
                NULL );
        *pbStatus = statusData.bStatus;
        return bRet;
}
```

Requirements

Header: Declared in AdvGPIO_IOCTL.h.

See Also

DeviceIoControl

5.2.5 IOCTL_ADVGPIO_SET_STATUS

The **IOCTL_ADVGPIO_SET_STATUS** control code to set the current GPIO status. The input parameter structure, **ADVGPIO_STATUS_DATA**, indicates which GPIO is going to set and the new status.

To perform this operation, call the **DeviceIoControl** function with the following parameters.

```
BOOL DeviceIoControl(
  (HANDLE) hDevice,                 // handle to device
  IOCTL_ADVGPIO_SET_STATUS,         // dwIoControlCode
  (LPVOID) lpInBuffer,              // lpInBuffer
  (DWORD) nInBufferSize,            // nInBufferSize
  NULL,                             // output buffer
  0,                                // size of output buffer
  (LPDWORD) lpBytesReturned,        // number of bytes returned
  NULL,                             // OVERLAPPED structure
);
```

Parameters

hDevice

[in] Handle to the GPIO device. To obtain an GPIO device handle, call the **CreateFile** function.

dwIoControlCode

[in] Control code for the operation. Use **IOCTL_ADVGPIO_SET_STATUS** for this operation.

lpInBuffer

[in] Pointer to an **ADVGPIO_DIR_DATA** structure.

nInBufferSize

[in] Size of the input buffer, in bytes.

lpOutBuffer

Not used with this operation; set to NULL.

nOutBufferSize

Not used with this operation; set to zero.

lpBytesReturned

[out] Pointer to a variable that receives the size of the data stored in the output buffer, in bytes.

lpOverlapped

NULL. *lpOverlapped* must be set to **NULL**.

Return Values

If the operation succeeds, **DeviceIoControl** returns a nonzero value.

If the operation fails, **DeviceIoControl** returns zero. To get extended error information, call **GetLastError** function.

Example Code

```
#include "Tchar.h"
#include "wtypes.h"
#include "winioctl.h"
#include "AdvGPIO_IOCTL.h"
// -------------------------------------------------------------------------
// DESCRIPTION: Sets the specified GPIO status (1-On/High, 0-Off/Low).
// -------------------------------------------------------------------------
BOOL GPIO_SetStatus ( HANDLE DriverHandle, UINT Index, BOOL bStatus )
{
	ADVGPIO_STATUS_DATA statusData = {0};

	statusData.uPinNumber = Index;
	statusData.bStatus = bStatus;

	DWORD dwReturn = 0;
	BOOL  bRet = DeviceIoControl(
		DriverHandle,
		IOCTL_ADVGPIO_SET_STATUS,
		&statusData,
		sizeof(statusData),
		NULL,
		0,
```

```
                &dwReturn,
                NULL );

        return bRet;
}
```

Requirements

Header: Declared in AdvGPIO_IOCTL.h.

See Also

DeviceIoControl

5.3 Data Structure

5.3.1 ADVGPIO_DIR_DATA

GPIO Direction Structure

DeviceIoControl's parameter uses this structure.

ADVGPIO_DIR_DATA structure is defined as follows:

```
typedef struct _ADVGPIO_DIR_DATA
{
        UCHAR         uPinNumber;
        BOOL          bDir;
} ADVGPIO_DIR_DATA, *PADVGPIO_DIR_DATA;
```

Members Description

uPinNumber

Specify the GPIO index in the range 0 to 7.

bDir

GPIO is Input(1) or Output(0) type.

5.3.2 ADVGPIO_STATUS_DATA

GPIO Status Structure

DeviceIoControl's parameter uses this structure.

ADVGPIO_STATUS_DATA structure is defined as follows:

```
typedef struct _ADVGPIO_STATUS_DATA
{
        UCHAR         uPinNumber;
        BOOL          bStatus;
} ADVGPIO_STATUS_DATA, *PADVGPIO_STATUS_DATA;
```

Members Description

uPinNumber

Specify the GPIO index in the range 0 to 7.

bStatus

GPIO status is 1-On/High or 0-Off/Low.

6. Software Utility & Programming Examples

Advantech GPIO Windows KMDF Driver package contains an example of Microsoft Visual C++ Program. You can refer to the example to develop applications.

KMDF:

Example Name	Description	Tool
GPIO_Sample	This example shows how to configure/manage of GPIO.	VC

6.1 Advantech GPIO Utility

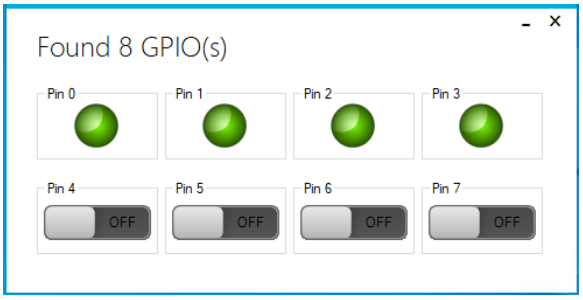
KMDF Source code (need to install PlatformSDK)

The sample code is located in the C:\Program Files\Advantech\PlatFormSDK\Sample\GPIO_Sample directory.

Binary File

File Name: GPIO Utility.exe

UI:



Pin0~n

There are up to n GPIOs and could control and monitor the direction (In / Out) and state (On / Off).

In/Out

"In/Out" button: The GPIO Utility will show the current direction of each GPIO and user can set Input(1) or Output(0) type of it.

On/Off

"On/Off" button: It will show the current status of each GPIO. If the GPIO is Output(0) type, user can set the status of the GPIO(1-On/High or 0-Off/Low). If the GPIO is Input(1) type, user can't change the status of the GPIO.