

# **USB EasyPOD Library for VC**

## **User Manual**

Index

**User Manual**..... 1

1. Introduction..... 3

2. Files Needed..... 3

3. How to Use EasyPOD DLL..... 3

4. How to use the Internal Built Function..... 4

    4.1. Connect..... 4

    4.2. Disconnect ..... 4

    4.3. WriteData ..... 4

    4.4. ReadData ..... 4

    4.5. ClearPODBuffer ..... 4

5. Example ..... 5

6. Revision History ..... 6

## 1. Introduction

This document explains how to use EasyPOD DLL in the Visual C++ 6.0 environment for EasyPOD data transmission and reception.

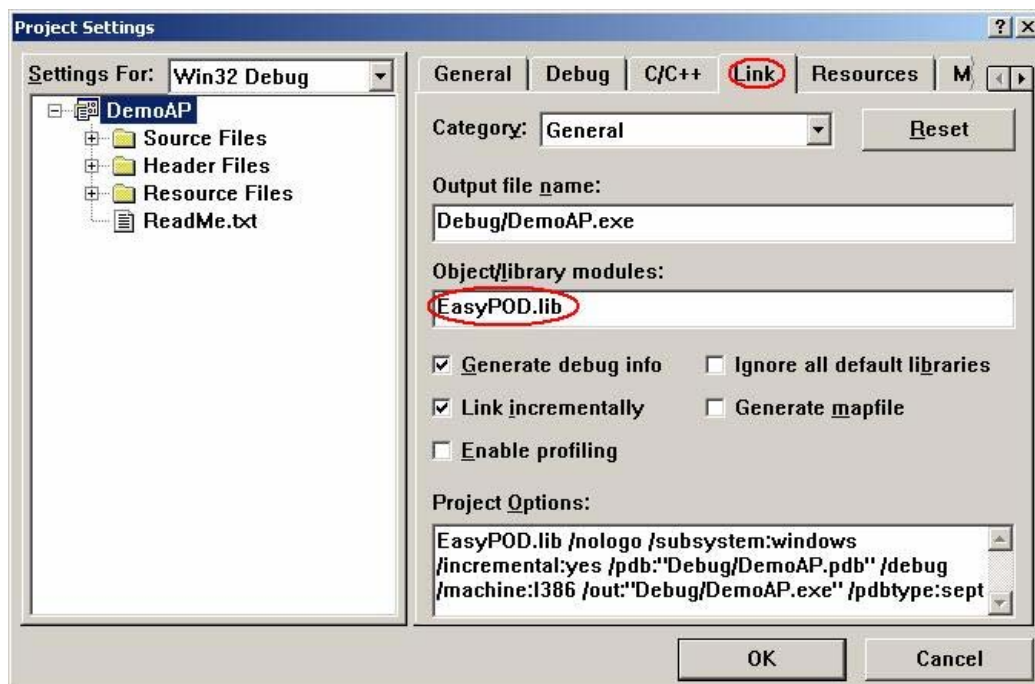
## 2. Files Needed

- 2.1. EasyPod.DLL
- 2.2. EasyPod.LIB
- 2.3. EasyPod.H

## 3. How to Use EasyPOD DLL

Activate the Visual C++(6.0) software for the development of the application program. Please follow the following steps:

- 3.1. Create a new project. Copy the files mentioned in #2(Files Needed Section) on to your project directory.
- 3.2. In your main source code, make sure that you write: #include "EasyPod.H" as your first line.
- 3.3. The next step is to add export library. Go to Project Menu then choose settings. Choose the link menu as shown in the figure on following. Write down the library module: EasyPod.lib



## 4. How to use the Internal Built Function

The DLL functions are described as following:

### 4.1. Connect

Function : `DWORD ConnectPod( MW_EasyPod * pEasyPod, DWORD Index );`  
Return : `ERROR_SUCCESS(0)` means connection is successful. A non 0 means connection fails.  
Parameter : `pEasyPOD`, a pointer of struct `MW_EasyPOD`.  
            Index, device index of EasyPOD, Index start at 1.

### 4.2. Disconnect

Function : `DWORD DisconnectPod( MW_EasyPod * pEasyPod );`  
Return : `ERROR_SUCCESS(0)` means disconnection is successful otherwise disconnection fails.  
Parameter : `pEasyPod`, a pointer of struct `MW_EasyPOD`.

### 4.3. WriteData

Function : `WriteData( MW_EasyPod * pEasyPod, LPBYTE lpBuffer , DWORD nNumberOfBytesToWrite , LPDWORD lpNumberOfBytesWritten );`  
Return : `ERROR_SUCCESS(0)` means writing is successful otherwise writing fails.  
Parameter : `pEasyPOD`, a pointer of struct `MW_EasyPOD`.  
            `lpBuffer`, Pointer to the buffer containing the data to write to the device.  
            `nNumberOfBytesToWrite`, number of Bytes to write.  
            `lpNumberOfBytesWritten`, actual number of bytes written.

### 4.4. ReadData

Function : `ReadData( MW_EasyPod * pEasyPod, LPBYTE lpBuffer, DWORD nNumberOfBytesToRead, LPDWORD lpNumberOfBytesRead );`  
Return : `ERROR_SUCCESS(0)` means reading is successful otherwise reading fails.  
Parameter : `pEasyPOD`, a pointer of struct `MW_EasyPOD`.  
            `lpBuffer`, Pointer to the buffer containing the data to read from the device.  
            `nNumberOfBytesToRead`, number of Bytes to read.  
            `lpNumberOfBytesRead`, actual number of bytes read.

### 4.5. ClearPODBuffer

Function : `ClearPODBuffer( MW_EasyPOD * pEasyPOD );`  
Return : `ERROR_SUCCESS(0)` means buffer clearing is successful otherwise the clearing fails.  
Parameter : `pEasyPOD`, a pointer of struct `MW_EasyPOD`.

## 5. Example

```
BYTE Buffer[2] = {0xA1,0x02} ;
DWORD dwResult;
DWORD dwByteWritten;
DWORD dwByteRead;
int m_ByteCount = 2;
BYTE * pRxBuf = (BYTE *)malloc(m_ByteCount);
MW_EasyPod NewEasyPod;

NewEasyPod.VID = 0x0E6A;
NewEasyPod.PID = 0x0317;

dwResult = ConnectPod(&NewEasyPod, 1);

dwResult = ClearPODBuffer(&NewEasyPod);

if (dwResult == ERROR_SUCCESS)
{
    NewEasyPod.ReadTimeOut = 500;
    NewEasyPod.WriteTimeOut = 500;
    WriteData(&NewEasyPod, Buffer, 2, &dwByteWritten);
    ReadData(&NewEasyPod, pRxBuf, 2, &dwByteRead);
    DisconnectPod(&NewEasyPod);
}
```

## 6. Revision History

Revision	Description	Date
v1.00	Release version	2008/01/30
v1.10	Balance I/O in Windows 2000	2009/01/09