

# GitHub Copilot introduction

---

- GitHub Copilot is an artificial intelligence program assistant launched by GitHub, aiming to improve developers' coding efficiency and quality.
- This tool uses large language models (LLM) trained by OpenAI to understand human language and generate code, helping developers write more accurate code faster.
- GitHub Copilot can provide code writing suggestions based on context, and can even automatically generate entire code based on user prompts.
- GitHub Copilot learns from millions of lines of public libraries to continuously improve the quality and accuracy of its code suggestions.

## Who is suitable to use?

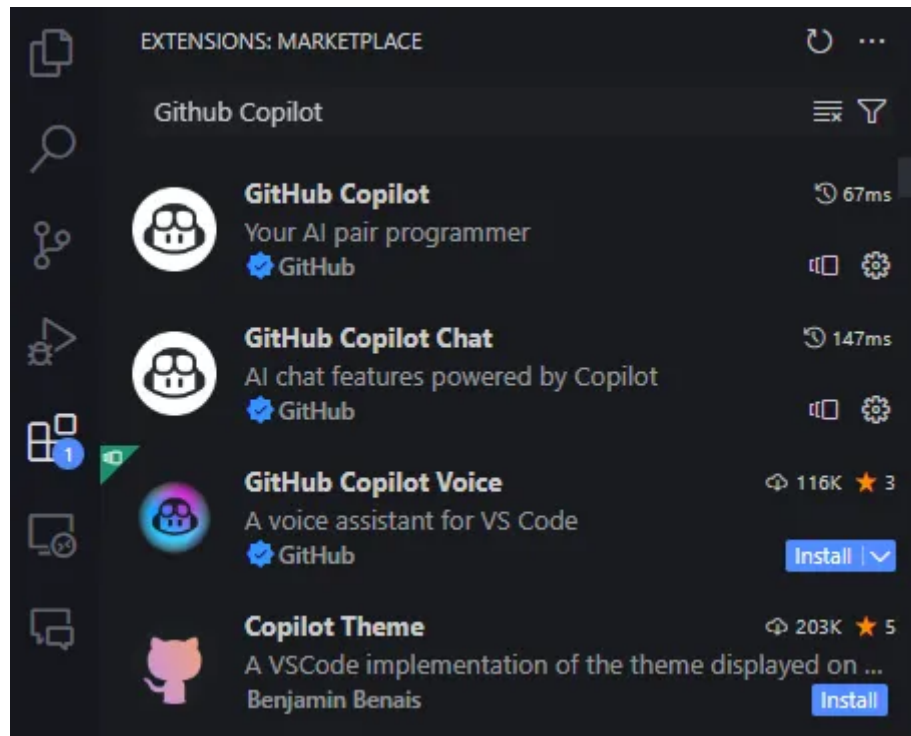
---

- Need to **write repetitive code** all the time: GitHub Copilot is very good at writing repetitive code, which can help developers save time and improve development efficiency
- You are **not familiar** with a certain programming language: GitHub Copilot can be a good assistant for you
- Need to frequently help others with **code reviews**: GitHub Copilot can help you with preliminary code reviews first, reducing the time you need to look at the program code.
- **Don't like to check the library** all the time online: GitHub Copilot will automatically fill in the code for you based on your context, saving a lot of time when you need to check the library all the time.
- Need to **write comments** frequently: GitHub Copilot can automatically generate comments for you based on the code snippets you give it, which is very practical in projects developed by multiple people.

# How to install?

---

- First, go to the GitHub Copilot official website and register. If you don't have a GitHub account yet, you need to create one first. After registration is complete, follow the instructions to install Github Copilot.
- Install Github Copilot and Github Chat on your VSCode



# How to use?

---

## - Auto fill source code

-- add New Module



Richard-Wang\_acer



I want to add new UEFI Application Module under Application folder , Module Name is MemTest

C ProvisionDataCopyFile.c63-84



GitHub Copilot

> Used 1 reference

To add a new UEFI Application Module named **MemTest** under the **Application** folder, follow these steps:

### Step 1: Create the Directory Structure

Create a new directory named **MemTest** under the **Application** folder:

```
mkdir .\Application\MemTest
```

### Step 2: Create the INF File

Create a file named **MemTest.inf** inside the **MemTest** directory with the following content:

```
## @file
# MemTest UEFI Application
#
# SPDX-License-Identifier: BSD-2-Clause-Patent
#
##

[Defines]
  INF_VERSION           = 0x00010006
  BASE_NAME             = MemTest
  FILE_GUID             = 12345678-1234-1234-1234-123456789
  MODULE_TYPE           = UEFI_APPLICATION
  VERSION_STRING        = 1.0
  ENTRY_POINT           = UefiMain

[Sources]
  MemTest.c

[Packages]
  MdePkg/MdePkg.dec
  MdeModulePkg/MdeModulePkg.dec

[Guids]
  gEfiGlobalVariableGuid ## CONSUMES ## GUID
```

### Step 3: Create the C Source File

Create a file named `MemTest.c` inside the `MemTest` directory with the following content:

```
#include <Uefi.h>
#include <Library/UefiBootServicesTableLib.h>
#include <Library/UefiLib.h>
#include <Library/PrintLib.h>

EFI_STATUS
EFIAPI
UefiMain (
    IN EFI_HANDLE      ImageHandle,
    IN EFI_SYSTEM_TABLE *SystemTable
)
{
    // Initialize the UEFI Library
    InitializeLib(ImageHandle, SystemTable);

    // Print a message to the UEFI console
    Print(L"Hello, UEFI World!\n");

    // Perform main tasks here

    // Return EFI_SUCCESS to indicate the application executed successfully
    return EFI_SUCCESS;
}
```

#### [Defines]

INF_VERSION	= 0x0001001B
BASE_NAME	= MemTest
FILE_GUID	= 12345678-1234-1234-1234-123456789ABC
MODULE_TYPE	= UEFI_APPLICATION
VERSION_STRING	= 1.0
ENTRY_POINT	= MemTestMain

#### [Sources]

MemTest.c

#### [Packages]

MdePkg/MdePkg.dec  
UefiApplicationPkg/UefiApplicationPkg.dec

#### [LibraryClasses]

UefiBootServicesTableLib  
UefiRuntimeServicesTableLib  
UefiLib

```
DebugLib
BaseLib
MemoryAllocationLib
```

```
[Guids]
    gEfiCallerIdGuid
```

```
[Protocols]
    gEfiLoadedImageProtocolGuid
```

```
[Pcd]
    gEfiMdePkgTokenSpaceGuid.PcdDebugPrintErrorLevel|0x80000000
```

```
#include <Uefi.h>
#include <Library/UefiBootServicesTableLib.h>
#include <Library/UefiLib.h>
#include <Library/MemoryAllocationLib.h>

EFI_STATUS
EFIAPI
UefiMain (
    IN EFI_HANDLE      ImageHandle,
    IN EFI_SYSTEM_TABLE *SystemTable
)
{
    EFI_STATUS Status;
    VOID *Memory;
    UINTN MemorySize = 1024 * 1024; // 1 MB

    // Allocate memory
    Status = gBS->AllocatePool(EfiBootServicesData, MemorySize,
    &Memory);
    if (EFI_ERROR(Status)) {
        Print(L"Failed to allocate memory: %r\n", Status);
        return Status;
    }

    // Perform memory test (simple write and read test)
    SetMem(Memory, MemorySize, 0xAA);
    for (UINTN i = 0; i < MemorySize; i++) {
        if (((UINT8*)Memory)[i] != 0xAA) {
```

```

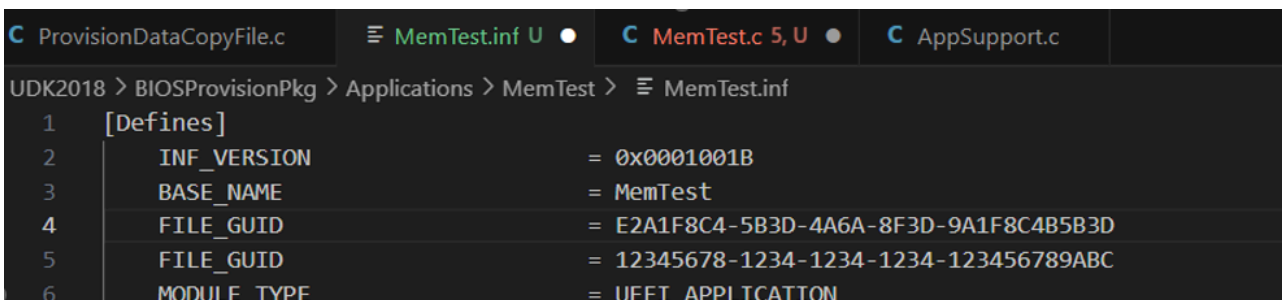
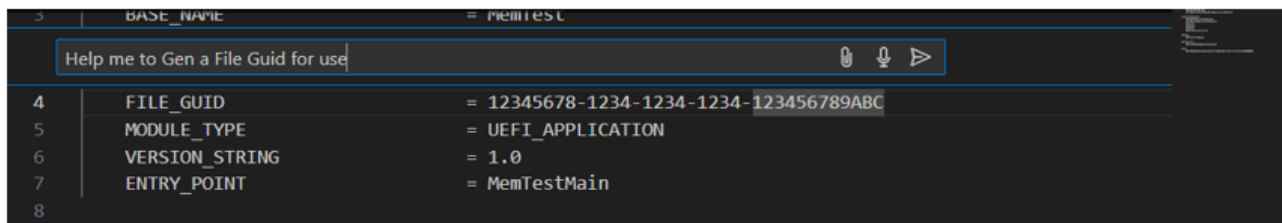
        Print(L"Memory test failed at byte %d\n", i);
        gBS->FreePool(Memory);
        return EFI_DEVICE_ERROR;
    }
}

Print(L"Memory test passed\n");

// Free allocated memory
gBS->FreePool(Memory);

return EFI_SUCCESS;
}

```



## - Explain

Below is a program which is BIOS Provision main function

```

EFI_STATUS
EFIAPI
UefiMain (
    IN EFI_HANDLE          ImageHandle,
    IN EFI_SYSTEM_TABLE    *SystemTable
)
{
    EFI_STATUS              Status;

```

```

Status = GetArg(ImageHandle);

if (EFI_ERROR(Status)) {
    Print(L"Please use UEFI SHELL to run this application!\n",
Status);
    return Status;
}
//Print(L"Argc = %d\n", Argc);
if (Argc < 2) {
    PrintUsage();
    return EFI_UNSUPPORTED;
}
if (StrCmp(Argv[1], L"-H") == 0 && Argc==2) {
    PrintUsage();
    Status = EFI_SUCCESS;
    return Status;
}

if (StrCmp(Argv[1], L"-v") == 0 && Argc==2) {
    Print(L"ProvisionDataApp version:");
    Print(Provision_Tool_Version);
    Print(L"\n");
    Status = EFI_SUCCESS;
    return Status;
}

if ((StrCmp(Argv[1], L"-F") == 0) || (StrCmp(Argv[1], L"-f") ==
0) ) {
    if (Argc != 3) {
        Print(L"ProvisionDataApp: Incorrect parameter count.\n");
        return RETURN_INVALID_PARAMETER;
    }
    Status = FileDump(Argv[2]);
    if(EFI_ERROR(Status))
        Print(L"FileDump Status = %r\n",Status);
    return Status;
}

if ((StrCmp(Argv[1], L"-I") == 0) || (StrCmp(Argv[1], L"-i") ==
0) ) {

```

```

    if (Argc < 3 || Argc > 7) {
        Print(L"ProvisionDataApp: Number of Index need >0 and <5\n");
        return RETURN_INVALID_PARAMETER;
    }
    Status = SetIndexArray(Argv);
    return Status;
}

if ((StrCmp(Argv[1], L"-P") == 0) || (StrCmp(Argv[1], L"-p") ==
0)) {
    if (Argc != 3) {
        Print(L"ProvisionDataApp: Incorrect parameter count.\n");
        return RETURN_INVALID_PARAMETER;
    }
    Status = ProvisionData(Argv[2]);
    return Status;
}

if ((StrCmp(Argv[1], L"-D") == 0) || (StrCmp(Argv[1], L"-d") ==
0)) {
    if (Argc != 2) {
        Print(L"ProvisionDataApp: Incorrect parameter count.\n");
        return RETURN_INVALID_PARAMETER;
    }
    Status = DumpCurrentVerb();
    return Status;
}

if ((StrCmp(Argv[1], L"-C") == 0) || (StrCmp(Argv[1], L"-c") ==
0)) {
    if (Argc != 4) {
        Print(L"ProvisionDataApp: Incorrect parameter count.\n");
        return RETURN_INVALID_PARAMETER;
    }
    Status = CheckCRCFile(Argv[2], Argv[3]);
    return Status;
}

if ((StrCmp(Argv[1], L"-T") == 0) || (StrCmp(Argv[1], L"-t") ==
0)) {
    if (Argc != 2) {
        Print(L"Enable Tender\n");
    }
}

```



```

        return RETURN_INVALID_PARAMETER;
    }
    Status = SetTender();
    return Status;
}

PrintUsage();

return Status;
}

```

we can use /explain to get below result

The provided code is a UEFI (Unified Extensible Firmware Interface) application written in C. The main function, `UefiMain`, serves as the entry point for the application. It takes two parameters: `ImageHandle`, which is a handle to the image, and `SystemTable`, which is a pointer to the EFI system table.

The function begins by declaring a variable `Status` of type `EFI_STATUS`, which will be used to store the status of various operations. The first operation is to call `GetArg` with `ImageHandle` as an argument. This function presumably retrieves the command-line arguments passed to the application. If `GetArg` returns an error, indicated by `EFI_ERROR(Status)`, the application prints an error message and returns the status.

Next, the code checks if the number of arguments (`Argc`) is less than 2. If so, it calls `PrintUsage` to display usage information and returns `EFI_UNSUPPORTED`. This ensures that the application has at least one argument to process.

The application then checks for various command-line options using `StrCmp` to compare the first argument (`Argv[1]`) with different strings. Each option corresponds to a different functionality:

1. `-H`: Displays usage information and returns `EFI_SUCCESS`.
2. `-V`: Prints the application version and returns `EFI_SUCCESS`.
3. `-F` or `-f`: Calls `FileDump` with the second argument (`Argv[2]`). If the argument count is incorrect, it prints an error message and returns `RETURN_INVALID_PARAMETER`.

4. **-I or -i**: Calls `SetIndexArray` with the arguments. It checks that the number of arguments is between 3 and 7. If not, it prints an error message and returns `RETURN_INVALID_PARAMETER`.
5. **-P or -p**: Calls `ProvisionData` with the second argument. If the argument count is incorrect, it prints an error message and returns `RETURN_INVALID_PARAMETER`.
6. **-D or -d**: Calls `DumpCurrentVerb`. If the argument count is incorrect, it prints an error message and returns `RETURN_INVALID_PARAMETER`.
7. **-C or -c**: Calls `CheckCRCFile` with the second and third arguments. If the argument count is incorrect, it prints an error message and returns `RETURN_INVALID_PARAMETER`.
8. **-T or -t**: Calls `SetTender`. If the argument count is incorrect, it prints an error message and returns `RETURN_INVALID_PARAMETER`.

If none of the recognized options are provided, the application calls `PrintUsage` to display usage information and returns the status.

Overall, the code is structured to handle various command-line options, each triggering a specific function or displaying relevant information. It ensures that the correct number of arguments is provided for each option and handles errors gracefully by printing appropriate messages and returning error statuses.

## - Auto write comment

```
/**
 * UefiMain - The entry point for the UEFI application.
 *
 * @param ImageHandle  The firmware allocated handle for the UEFI
image.
 * @param SystemTable  A pointer to the EFI System Table.
 *
 * @return EFI_STATUS  The status of the application execution.
 *
 * This function processes command-line arguments and performs
various actions
 * based on the provided arguments. The supported arguments are:
 *
 * - -H: Displays usage information.
 * - -V: Displays the application version.
```

```

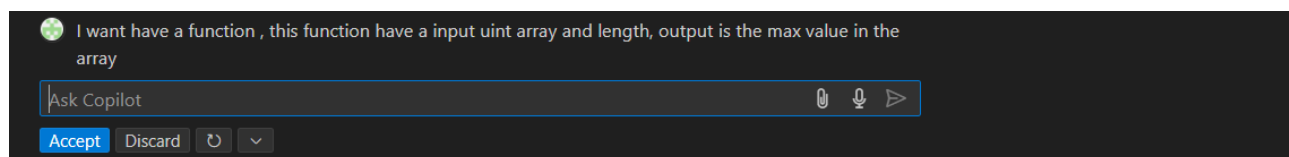
* - -F <filename>: Dumps the contents of the specified file.
* - -I <index1> <index2> ... <indexN>: Sets an array of indices (1
to 5 indices).
* - -P <filename>: Provisions data from the specified file.
* - -D: Dumps the current verb.
* - -C <file1> <file2>: Checks the CRC of the specified files.
* - -T: Enables the tender.
*
* The function returns appropriate EFI_STATUS codes based on the
success or
* failure of the operations.
*/

```

## - Code Generate

We can describe a requirement and ask Github Copilot to Generate code

For Example : I want have a function , this function have a input uint array and length, output is the max value in the array



```

/**
Find the maximum value in an array.

@param[in]  Array      The input array.
@param[in]  Length     The length of the array.

@return The maximum value in the array.
**/
UINTN
FindMaxValue (
    IN UINTN *Array,
    IN UINTN Length
)
{
    UINTN MaxValue = 0;
    UINTN Index;

    if (Array == NULL || Length == 0) {

```

```

    return 0;
}

MaxValue = Array[0];
for (Index = 1; Index < Length; Index++) {
    if (Array[Index] > MaxValue) {
        MaxValue = Array[Index];
    }
}

return MaxValue;
}

```

## - Simplify

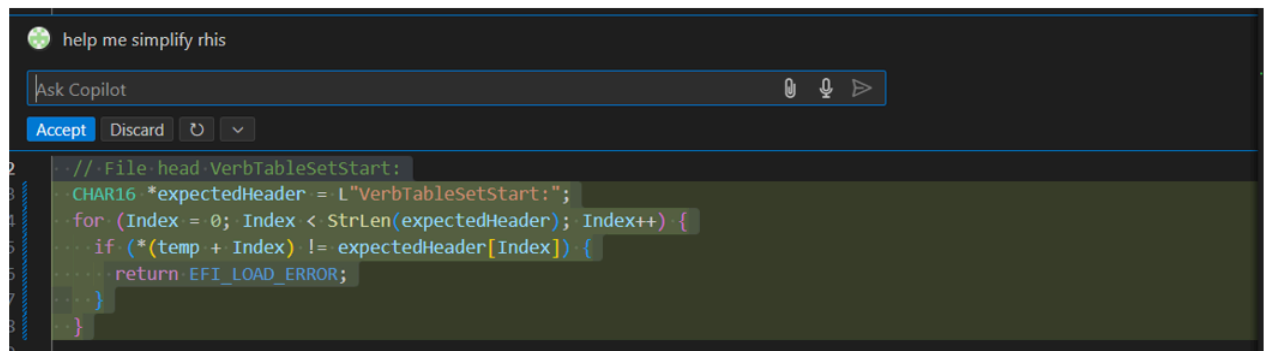
Below section is some code for read file header.

```

// File head VerbTableSetStart:
if(*temp != 'v') return EFI_LOAD_ERROR;
if(*(temp+1) != 'e') return EFI_LOAD_ERROR;
if(*(temp+2) != 'r') return EFI_LOAD_ERROR;
if(*(temp+3) != 'b') return EFI_LOAD_ERROR;
if(*(temp+4) != 'T') return EFI_LOAD_ERROR;
if(*(temp+5) != 'a') return EFI_LOAD_ERROR;
if(*(temp+6) != 'b') return EFI_LOAD_ERROR;
if(*(temp+7) != 'l') return EFI_LOAD_ERROR;
if(*(temp+8) != 'e') return EFI_LOAD_ERROR;
if(*(temp+9) != 'S') return EFI_LOAD_ERROR;
if(*(temp+10) != 'e') return EFI_LOAD_ERROR;
if(*(temp+11) != 't') return EFI_LOAD_ERROR;
if(*(temp+12) != 'S') return EFI_LOAD_ERROR;
if(*(temp+13) != 't') return EFI_LOAD_ERROR;
if(*(temp+14) != 'a') return EFI_LOAD_ERROR;
if(*(temp+15) != 'r') return EFI_LOAD_ERROR;
if(*(temp+16) != 't') return EFI_LOAD_ERROR;
if(*(temp+17) != ':') return EFI_LOAD_ERROR;

```

We ask Copilot to Simplify it



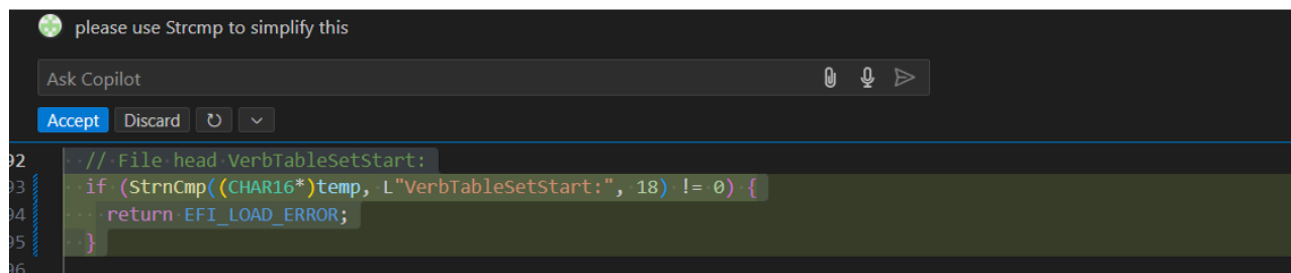
help me simplify this

Ask Copilot

Accept Discard ↻ ▾

```
02 // File head VerbTableSetStart:
03 CHAR16 *expectedHeader = L"VerbTableSetStart: ";
04 for (Index = 0; Index < StrLen(expectedHeader); Index++) {
05     if (*(temp + Index) != expectedHeader[Index]) {
06         return EFI_LOAD_ERROR;
07     }
08 }
```

Does this is the best expression ?



please use Strncmp to simplify this

Ask Copilot

Accept Discard ↻ ▾

```
02 // File head VerbTableSetStart:
03 if (StrnCmp((CHAR16*)temp, L"VerbTableSetStart:", 18) != 0) {
04     return EFI_LOAD_ERROR;
05 }
06 }
```

## Create a simple project with Github Copilot

### -Requirement

Build a tool for Get/Set UEFI variable under windows

Using MSFT C# to implement

Function requirement

- Command /s or /S to with Name, Guid and value
- Command /g or /G to get a specify variable by Name and Guid, and return value
- Command /d or /D to delete a variable by Name and Guid
- Command /h or /H to get help strings

### -Create a C# Project under Visual Studio

```
using System;
```

```
namespace AcerBIOSVariableConfig
```

```


{
    class Program
    {
        static void Main(string[] args)
        {




            Console.WriteLine("Hello world!");





        }
    }
}

```

## -Create a sub function for print help strings

 i want create a sub function to print help string for - Command /s or /S to with Name, Guid and Value  
 - Command /g or /G to get a specify variable by Name and Guid, and return Value - Command /d or /D to delete a variable by Name and Guid - Command /h or /H to get help strings

Ask Copilot   

Accept Discard    


0 references




```



4 static void PrintHelp()
5 {
6     Console.WriteLine("Usage:");
7     Console.WriteLine(" /s or /S - Set a variable with Name, Guid, and Value");
8     Console.WriteLine(" /g or /G - Get a specific variable by Name and Guid, and return Value");
9     Console.WriteLine(" /d or /D - Delete a variable by Name and Guid");
10    Console.WriteLine(" /h or /H - Display this help message");
11 }

```

## -Create a sub function Set UEFI Variable

 i want create a sub function to set UEFI variable input : Variable Name , Variable GUID , Variable Value and length Return : True or False for fail or success Variable value should be byte array , length is the length of byte array

Ask Copilot   

Accept Discard  

```

using System.Runtime.InteropServices;

[DllImport("kernel32.dll", SetLastError = true)]
0 references
private static extern bool SetFirmwareEnvironmentVariable(
    string lpName,
    string lpGuid,
    byte[] pValue,
    uint nSize);

0 references
static bool SetUEFIVariable(string name, string guid, byte[] value, uint length)
{
    bool result = SetFirmwareEnvironmentVariable(name, guid, value, length);
    if (!result)
    {
        int error = Marshal.GetLastWin32Error();
        Console.WriteLine($"Failed to set UEFI variable. Error code: {error}");
    }
    return result;
}

```

## -Create a sub function for Get UEFI Variable

i want create a sub function to get UEFI variable, print the result on console input : Variable Name, Variable GUID and Buffer Size Return : True or false for fail or success

Ask Copilot

Accept Discard ↺ ▼

```
[DllImport("kernel32.dll", SetLastError = true)]
0 references
private static extern bool GetFirmwareEnvironmentVariable(
    string lpName,
    string lpGuid,
    byte[] pBuffer,
    uint nSize);

0 references
static bool GetUEFIVariable(string name, string guid, uint bufferSize)
{
    byte[] buffer = new byte[bufferSize];
    bool result = GetFirmwareEnvironmentVariable(name, guid, buffer, bufferSize);
    if (result)
    {
        Console.WriteLine($"UEFI Variable Value: {BitConverter.ToString(buffer)}");
    }
    else
    {
        int error = Marshal.GetLastWin32Error();
        Console.WriteLine($"Failed to get UEFI variable. Error code: {error}");
    }
    return result;
}
```

## -Create a sub function for Delete UEFI Variable

i want create a sub function to delete UEFI variable input : Variable Name, Variable GUID Return : True or false for fail or success

Ask Copilot

Accept Discard ↺ ▼

```
[DllImport("kernel32.dll", SetLastError = true)]
0 references
private static extern bool SetFirmwareEnvironmentVariable(
    string lpName,
    string lpGuid,
    IntPtr pValue,
    uint nSize);

0 references
static bool DeleteUEFIVariable(string name, string guid)
{
    bool result = SetFirmwareEnvironmentVariable(name, guid, IntPtr.Zero, 0);
    if (!result)
    {
        int error = Marshal.GetLastWin32Error();
        Console.WriteLine($"Failed to delete UEFI variable. Error code: {error}");
    }
    return result;
}
```

## -Modify Main function

Ask Copilot :

modify main function

- Command /s or /S to call SetUEFIVariable ,check arg length is 4,  
arg2 is variable name , arg3 is variable GUID , arg4 is value
- Command /g or /G to call GetUEFIVariable , check arg length is 3  
arg2 is variable name , arg3 is variable GUID
- Command /d or /D to call DeleteUEFIVariable, check arg length is 3  
arg2 is variable name , arg3 is variable GUID
- Command /h or /H print help

result:

```
using System;

namespace AcerBIOSVariableConfig
{
    class Program
    {
        static void Main(string[] args)
        {

            if (args.Length == 0)
            {
                PrintHelp();
                return;
            }

            switch (args[0].ToLower())
            {
                case "/s":
                    if (args.Length == 4)
                    {
                        string name = args[1];
                        string guid = args[2];
                        byte[] value =
System.Text.Encoding.UTF8.GetBytes(args[3]);
```



```

        SetUEFIVariable(name, guid, value,
(uint)value.Length);
    }
    else
    {
        Console.WriteLine("Invalid arguments for /s
command.");

        Printhelp();
    }
    break;

case "/g":
    if (args.Length == 3)
    {
        string name = args[1];
        string guid = args[2];
        GetUEFIVariable(name, guid, 1024); //
Assuming buffer size of 1024
    }
    else
    {
        Console.WriteLine("Invalid arguments for /g
command.");

        Printhelp();
    }
    break;

case "/d":
    if (args.Length == 3)
    {
        string name = args[1];
        string guid = args[2];
        DeleteUEFIVariable(name, guid);
    }
    else
    {
        Console.WriteLine("Invalid arguments for /d
command.");

        Printhelp();
    }
    break;

```

```

        case "/h":
            PrintHelp();
            break;

        default:
            Console.WriteLine("Unknown command.");
            PrintHelp();
            break;
    }

    Console.WriteLine("Hello world!");
}

static void PrintHelp()
{
    Console.WriteLine("Usage:");
    Console.WriteLine("  /s or /S - Set a variable with
Name, Guid, and value");
    Console.WriteLine("  /g or /G - Get a specific variable
by Name and Guid, and
                                return value");
    Console.WriteLine("  /d or /D - Delete a variable by
Name and Guid");
    Console.WriteLine("  /h or /H - Display this help
message");
}

using System.Runtime.InteropServices;

[DllImport("kernel32.dll", SetLastError = true)]
private static extern bool SetFirmwareEnvironmentVariable(
    string lpName,
    string lpGuid,
    byte[] pValue,
    uint nSize);

static bool SetUEFIVariable(string name, string guid,
byte[] value, uint length)
{
    bool result = SetFirmwareEnvironmentVariable(name,
guid, value, length);

```

```

        if (!result)
        {
            int error = Marshal.GetLastWin32Error();
            Console.WriteLine($"Failed to set UEFI variable.
Error code: {error}");
        }
        return result;
    }

[DllImport("kernel32.dll", SetLastError = true)]
private static extern bool GetFirmwareEnvironmentVariable(
    string lpName,
    string lpGuid,
    byte[] pBuffer,
    uint nSize);

static bool GetUEFIVariable(string name, string guid, uint
bufferSize)
{
    byte[] buffer = new byte[bufferSize];
    bool result = GetFirmwareEnvironmentVariable(name,
guid, buffer, bufferSize);
    if (result)
    {
        Console.WriteLine($"UEFI variable value:
{BitConverter.ToString(buffer)}");
    }
    else
    {
        int error = Marshal.GetLastWin32Error();
        Console.WriteLine($"Failed to get UEFI variable. Error
code: {error}");
    }
    return result;
}

[DllImport("kernel32.dll", SetLastError = true)]
private static extern bool SetFirmwareEnvironmentVariable(
    string lpName,
    string lpGuid,
    IntPtr pValue,
    uint nSize);

```

```

        static bool DeleteUEFIVariable(string name, string guid)
        {
            bool result = SetFirmwareEnvironmentVariable(name,
guid, IntPtr.Zero, 0);
            if (!result)
            {
                int error = Marshal.GetLastWin32Error();
                Console.WriteLine($"Failed to delete UEFI variable.
Error code: {error}");
            }
            return result;
        }
    }
}

```

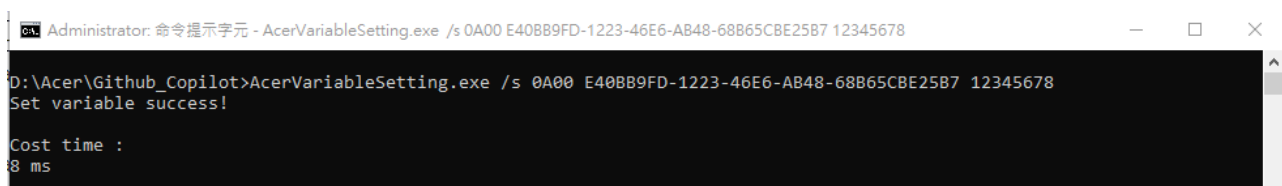
## -After fine tune the code...

Build and Run the program, test by below commands

```

AcerVariableSetting.exe /h
AcerVariableSetting.exe /s 0A00 E40BB9FD-1223-46E6-AB48-
68B65CBE25B7 12345678
AcerVariableSetting.exe /g 0A00 E40BB9FD-1223-46E6-AB48-
68B65CBE25B7
AcerVariableSetting.exe /d 0A00 E40BB9FD-1223-46E6-AB48-
68B65CBE25B7
AcerVariableSetting.exe /g 0A00 E40BB9FD-1223-46E6-AB48-
68B65CBE25B7

```



```

Administrator: 命令提示字元 - AcerVariableSetting.exe /s 0A00 E40BB9FD-1223-46E6-AB48-68B65CBE25B7 12345678
D:\Acer\Github_Copilot>AcerVariableSetting.exe /s 0A00 E40BB9FD-1223-46E6-AB48-68B65CBE25B7 12345678
Set variable success!
Cost time :
8 ms

```

```

D:\Acer\Github_Copilot>AcerVariableSetting.exe /g 0A00 E40BB9FD-1223-46E6-AB48-68B65CBE25B7
Get variable success!
12 34 56 78
Cost time :
1 ms

```

```
D:\Acer\Github_Copilot>AcerVariableSetting.exe /d 0A00 E40BB9FD-1223-46E6-AB48-68B65CBE25B7
Delete variable success!

Cost time :
2 ms
```

```
D:\Acer\Github_Copilot>AcerVariableSetting.exe /g 0A00 E40BB9FD-1223-46E6-AB48-68B65CBE25B7
Get variable fail!
```

## Summary

---

Github Copilot is a strong utility for develop program, not only suitable for new comer and also could apply on Senior Engineer.

In the past, I had to look up many libraries and piece them together. Now with Copilot, I save a lot of time looking for libraries. I think Copilot literally means "co-pilot". It can effectively assist you in program development. If you haven't tried using Copilot for collaborative development, I sincerely recommend that readers give it a try. I hope this article will help. This is the starting point for readers and AI to collaboratively develop and write code.