[C#] SHGetFileInfo を利用してファイルの情報を取得する

このページのタグ:[C#] [シェルネームスペース]

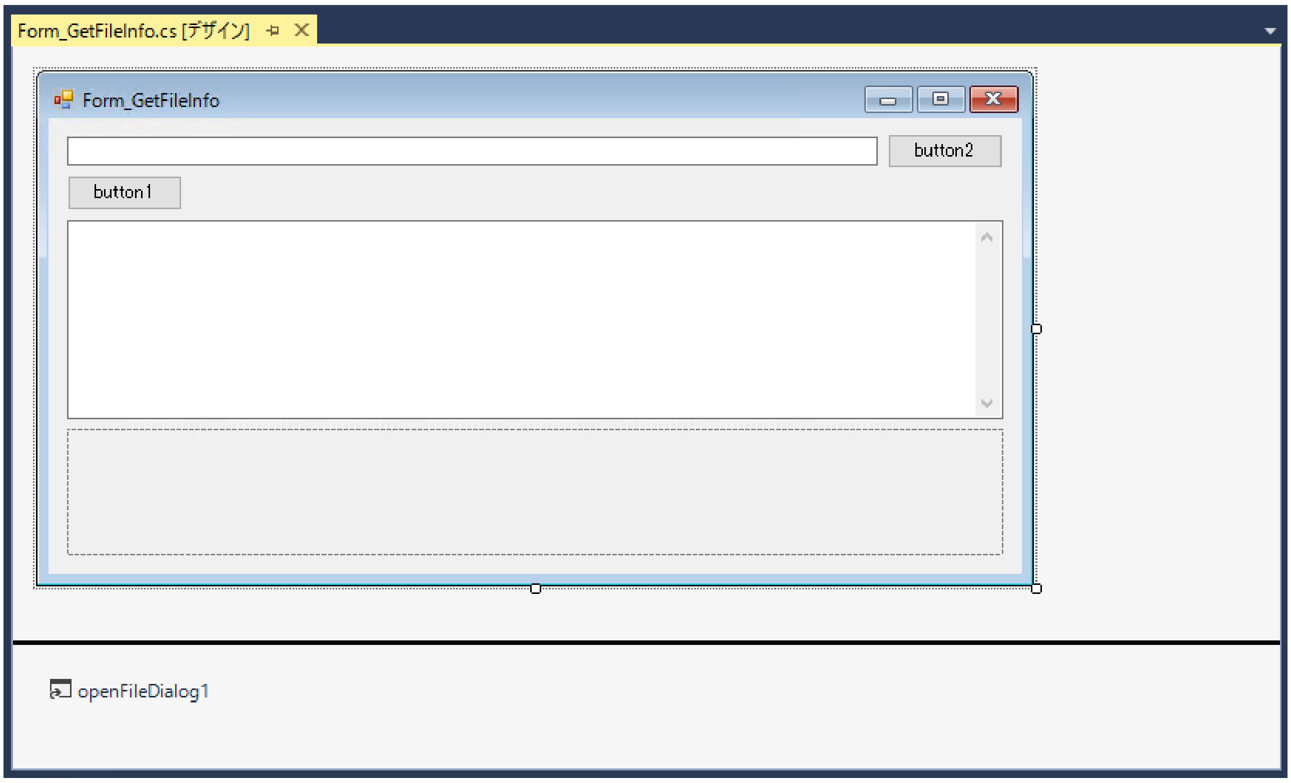
SHGetFileInfo を利用してファイルの情報を取得するコードを紹介します。

ファイルパスから情報を取得する場合

ファイルパスから情報を取得するコードを紹介します。

UI

下図のUIを作成します。
TextBox,TextBox複数行, Buttonを2つ、Panelを1つ、OpenFileDialog を配置します。



コード

下記のコードを記述します。ライブラリ部分のコードはこのページの末尾に記載します。

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using ShellNamespace;
using System.Runtime.InteropServices;

namespace ShellNamespaceDemo
```

```
{
  public partial class Form_GetFileInfo : Form
  {
    public Form_GetFileInfo()
    {
      InitializeComponent();
    }

    private void button2_Click(object sender, EventArgs e)
    {
      if (openFileDialog1.ShowDialog() == DialogResult.OK) {
        textBox1.Text = openFileDialog1.FileName;
      }
    }

    private void button1_Click(object sender, EventArgs e)
    {
      WindowsAPI.SHFILEINFO shinfo = new WindowsAPI.SHFILEINFO();
      string filename = textBox1.Text;
      WindowsAPI.SHGFI flag =
        WindowsAPI.SHGFI.SHGFI_ATTRIBUTES
        | WindowsAPI.SHGFI.SHGFI_ATTR_SPECIFIED
        | WindowsAPI.SHGFI.SHGFI_DISPLAYNAME
        | WindowsAPI.SHGFI.SHGFI_EXETYPE
        | WindowsAPI.SHGFI.SHGFI_ICON
        | WindowsAPI.SHGFI.SHGFI_ICONLOCATION
        | WindowsAPI.SHGFI.SHGFI_LARGEICON
        | WindowsAPI.SHGFI.SHGFI_OPENICON
        | WindowsAPI.SHGFI.SHGFI_OVERLAYINDEX
        | WindowsAPI.SHGFI.SHGFI_SHELLICONSIZE
        | WindowsAPI.SHGFI.SHGFI_SYSICONINDEX
        | WindowsAPI.SHGFI.SHGFI_TYPENAME
        ;

      IntPtr ret = WindowsAPI.SHGetFileInfo(filename, 0, out shinfo, (uint)Marshal.SizeOf(typeof(WindowsAPI.SHFILEINFO)), flag);

      if (ret == IntPtr.Zero) {
        Application.Exit();
      }

      textBox2.Text += string.Format("表示名:{0}\r\n", shinfo.szDisplayName);
      textBox2.Text += string.Format("タイプ名:{0}\r\n", shinfo.szTypeName);
      textBox2.Text += string.Format("アイコンのインデックス:{0:d}\r\n", shinfo.iIcon);
      textBox2.Text += string.Format("Attributes:{0:d}\r\n", shinfo.dwAttributes);

      bool cancopy = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_CANCOPY) == (uint)WindowsAPI.SFGAOF.SFGAO_CANCOPY) {
        cancopy = true;
      }
      textBox2.Text += string.Format("コピー可能:{0:b}\r\n", cancopy);

      bool canmove = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_CANMOVE) == (uint)WindowsAPI.SFGAOF.SFGAO_CANMOVE) {
        canmove = true;
      }
      textBox2.Text += string.Format("移動可能:{0:b}\r\n", canmove);

      bool canlink = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_CANLINK) == (uint)WindowsAPI.SFGAOF.SFGAO_CANLINK) {
        canlink = true;
      }
      textBox2.Text += string.Format("リンク可能:{0:b}\r\n", canlink);

      bool isstorage = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_STORAGE) == (uint)WindowsAPI.SFGAOF.SFGAO_STORAGE) {
        isstorage = true;
      }
      textBox2.Text += string.Format("ストレージである:{0:b}\r\n", isstorage);

      bool canrename = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_CANRENAME) == (uint)WindowsAPI.SFGAOF.SFGAO_CANRENAME) {
        canrename = true;
      }
      textBox2.Text += string.Format("リネーム可能:{0:b}\r\n", canrename);

      bool candelete = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_CANDELETE) == (uint)WindowsAPI.SFGAOF.SFGAO_CANDELETE) {
        candelete = true;
      }
      textBox2.Text += string.Format("削除可能:{0:b}\r\n", candelete);

      bool haspropsheet = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_HASPROPSHEET) == (uint)WindowsAPI.SFGAOF.SFGAO_HASPROPSHEET) {
        haspropsheet = true;
      }
      textBox2.Text += string.Format("プロパティシートがある:{0:b}\r\n", haspropsheet);

      bool droptarget = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_DROPTARGET) == (uint)WindowsAPI.SFGAOF.SFGAO_DROPTARGET) {
        droptarget = true;
      }
      textBox2.Text += string.Format("プロパティシートがある:{0:b}\r\n", droptarget);

      bool issystem = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_SYSTEM) == (uint)WindowsAPI.SFGAOF.SFGAO_SYSTEM) {
        issystem = true;
      }
      textBox2.Text += string.Format("システム要素である:{0:b}\r\n", issystem);

      bool encrypted = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_ENCRYPTED) == (uint)WindowsAPI.SFGAOF.SFGAO_ENCRYPTED) {
        encrypted = true;
      }
      textBox2.Text += string.Format("暗号化されている:{0:b}\r\n", encrypted);

      bool isslow = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_ISSLOW) == (uint)WindowsAPI.SFGAOF.SFGAO_ISSLOW) {
        isslow = true;
      }
      textBox2.Text += string.Format("低速アクセスである:{0:b}\r\n", isslow);

      bool ghosted = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_GHOSTED) == (uint)WindowsAPI.SFGAOF.SFGAO_GHOSTED) {
        ghosted = true;
      }
      textBox2.Text += string.Format("ユーザーは利用できない:{0:b}\r\n", ghosted);

      bool islink = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_LINK) == (uint)WindowsAPI.SFGAOF.SFGAO_LINK) {
        islink = true;
      }
      textBox2.Text += string.Format("リンクである:{0:b}\r\n", islink);

      bool shared = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_SHARE) == (uint)WindowsAPI.SFGAOF.SFGAO_SHARE) {
        shared = true;
      }
      textBox2.Text += string.Format("共有されている:{0:b}\r\n", shared);

      bool isreadonly = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_READONLY) == (uint)WindowsAPI.SFGAOF.SFGAO_READONLY) {
        isreadonly = true;
      }
      textBox2.Text += string.Format("リードオンリーである:{0:b}\r\n", isreadonly);

      bool ishidden = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_HIDDEN) == (uint)WindowsAPI.SFGAOF.SFGAO_HIDDEN) {
        ishidden = true;
      }
      textBox2.Text += string.Format("隠し属性である:{0:b}\r\n", ishidden);

      bool nonenumerated = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_NONENUMERATED) == (uint)WindowsAPI.SFGAOF.SFGAO_NONENUMERATED) {
        nonenumerated = true;
      }
      textBox2.Text += string.Format("列挙できない要素である:{0:b}\r\n", nonenumerated);

      bool newcontent = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_NEWCONTENT) == (uint)WindowsAPI.SFGAOF.SFGAO_NEWCONTENT) {
        newcontent = true;
      }
      textBox2.Text += string.Format("新規コンテンツである:{0:b}\r\n", newcontent);

      bool isstream = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_STREAM) == (uint)WindowsAPI.SFGAOF.SFGAO_STREAM) {
        isstream = true;
      }
      textBox2.Text += string.Format("ストリームがある(BindToObjectが可能):{0:b}\r\n", isstream);

      bool storageancestor = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_STORAGEANCESTOR) == (uint)WindowsAPI.SFGAOF.SFGAO_STORAGEANCESTOR) {
        storageancestor = true;
      }
      textBox2.Text += string.Format("ストリームの子要素である:{0:b}\r\n", storageancestor);

      bool removable = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_REMOVABLE) == (uint)WindowsAPI.SFGAOF.SFGAO_REMOVABLE) {
        removable = true;
      }
      textBox2.Text += string.Format("リムーバブルできる要素である:{0:b}\r\n", removable);

      bool compressed = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_COMPRESSED) == (uint)WindowsAPI.SFGAOF.SFGAO_COMPRESSED) {
        compressed = true;
      }
      textBox2.Text += string.Format("圧縮されている:{0:b}\r\n", compressed);

      bool browsable = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_BROWSABLE) == (uint)WindowsAPI.SFGAOF.SFGAO_BROWSABLE) {
        browsable = true;
      }
      textBox2.Text += string.Format("エクスプローラで閲覧可能である:{0:b}\r\n", browsable);

      bool fileancestor = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_FILESYSANCESTOR) == (uint)WindowsAPI.SFGAOF.SFGAO_FILESYSANCESTOR) {
        fileancestor = true;
      }
      textBox2.Text += string.Format("システムフォルダまたはファイルシステムである:{0:b}\r\n", fileancestor);

      bool isfolder = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_FOLDER) == (uint)WindowsAPI.SFGAOF.SFGAO_FOLDER) {
        isfolder = true;
      }
      textBox2.Text += string.Format("フォルダである:{0:b}\r\n", isfolder);

      bool isfilesystem = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_FILESYSTEM) == (uint)WindowsAPI.SFGAOF.SFGAO_FILESYSTEM) {
        isfilesystem = true;
      }
      textBox2.Text += string.Format("ファイルシステムである:{0:b}\r\n", isfilesystem);

      bool hassubfolder = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_HASSUBFOLDER) == (uint)WindowsAPI.SFGAOF.SFGAO_HASSUBFOLDER) {
        hassubfolder = true;
      }
      textBox2.Text += string.Format("子フォルダを持っている:{0:b}\r\n", hassubfolder);

      if (shinfo.hIcon != IntPtr.Zero) {
        Icon myIcon = Icon.FromHandle(shinfo.hIcon);
        Graphics g = Graphics.FromHwnd(panel1.Handle);
        g.DrawIcon(myIcon, 10, 10);
      }

    }
  }
}
```

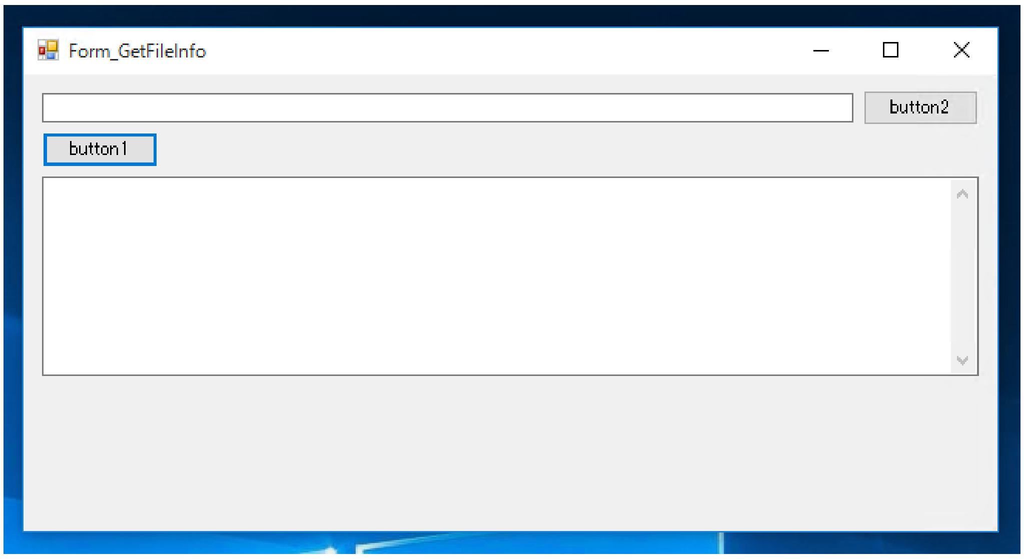
解説

```
IntPtr ret = WindowsAPI.SHGetFileInfo(filename, 0, out shinfo, (uint)Marshal.SizeOf(typeof(WindowsAPI.SHFILEINFO)), flag);
```

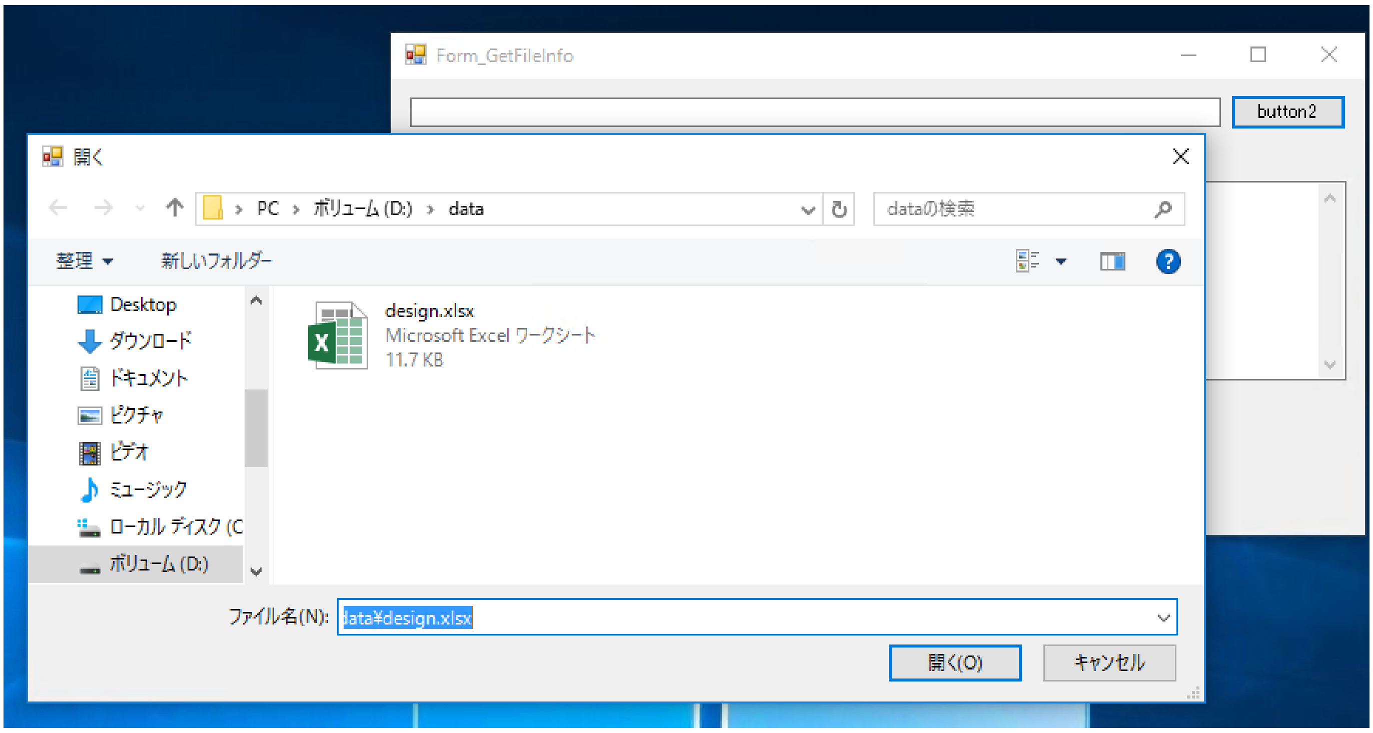
上記のSHGetFileInfoを呼び出すことでファイル名の情報を取得できます。第一引数には情報を取得するファイル名を与えます。第三引数には結果を返すためのSHFILEINFO構造体を与えます。第4引数は第三引数のSHFILEINFO構造体のサイズを指定します。第四引数はどの情報を取得するかを設定するフラグを与えます。
結果はSHFILEINFO構造体に格納されて返るので、SHFILEINFO構造体のメンバ変数を確認して結果を取得します。

実行結果

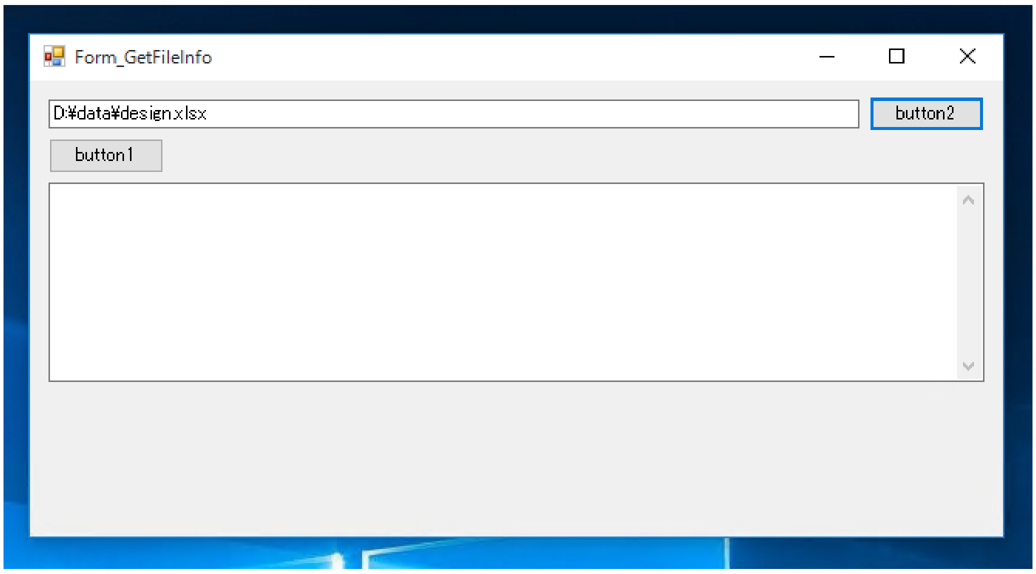
プロジェクトを実行します。下図のウィンドウが表示されます。



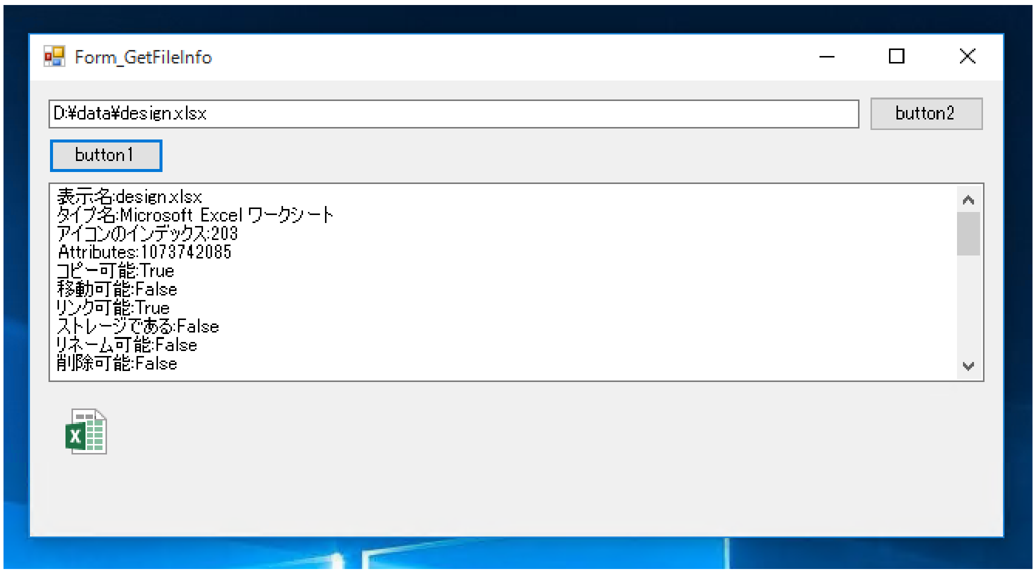
[button2]をクリックすると、ファイルを開くダイアログが表示されます。情報を取得したいファイルを選択します。



ファイルを選択すると、上部のテキストボックスにファイルパスが表示されます。なお、直接テキストボックスにファイルパスを入力しても動作します。

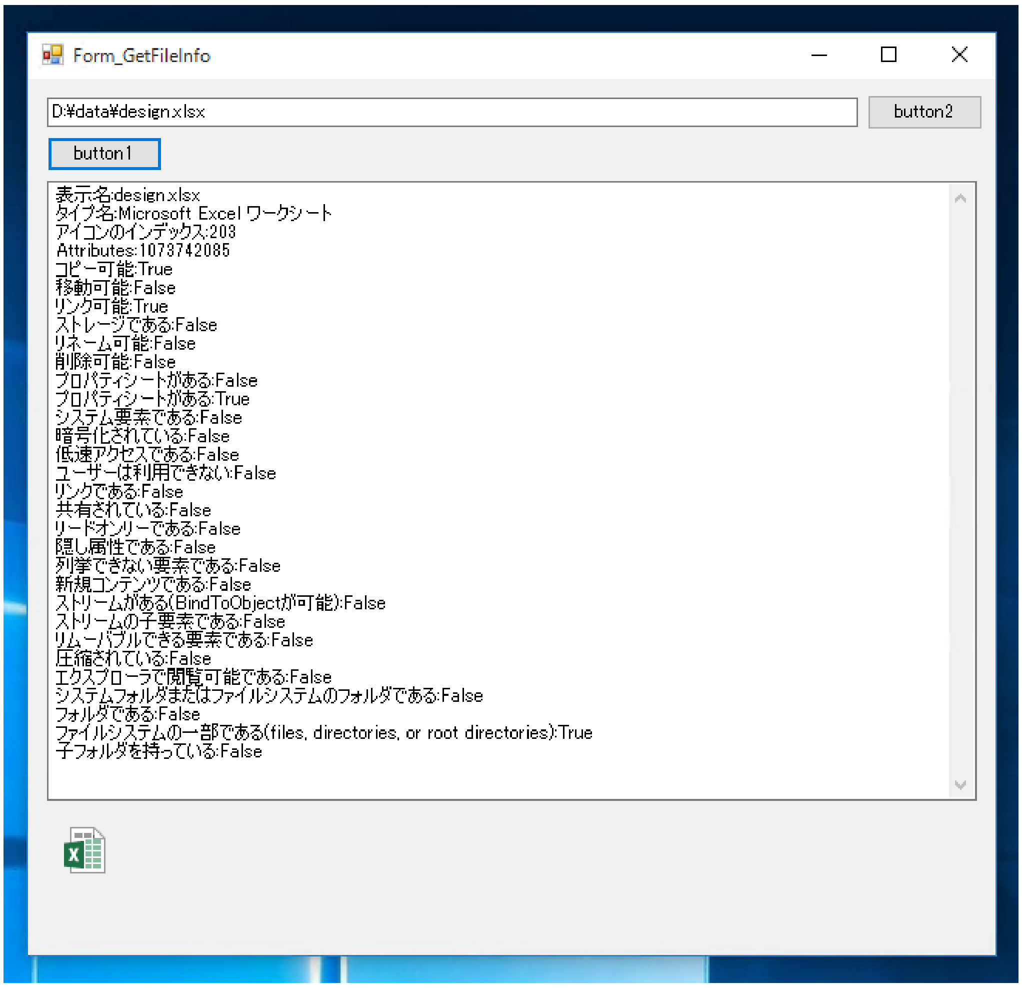


[button1]をクリックすると、ファイルの情報が表示されます。またウィンドウの下部のパネルにアイコンが表示されます。



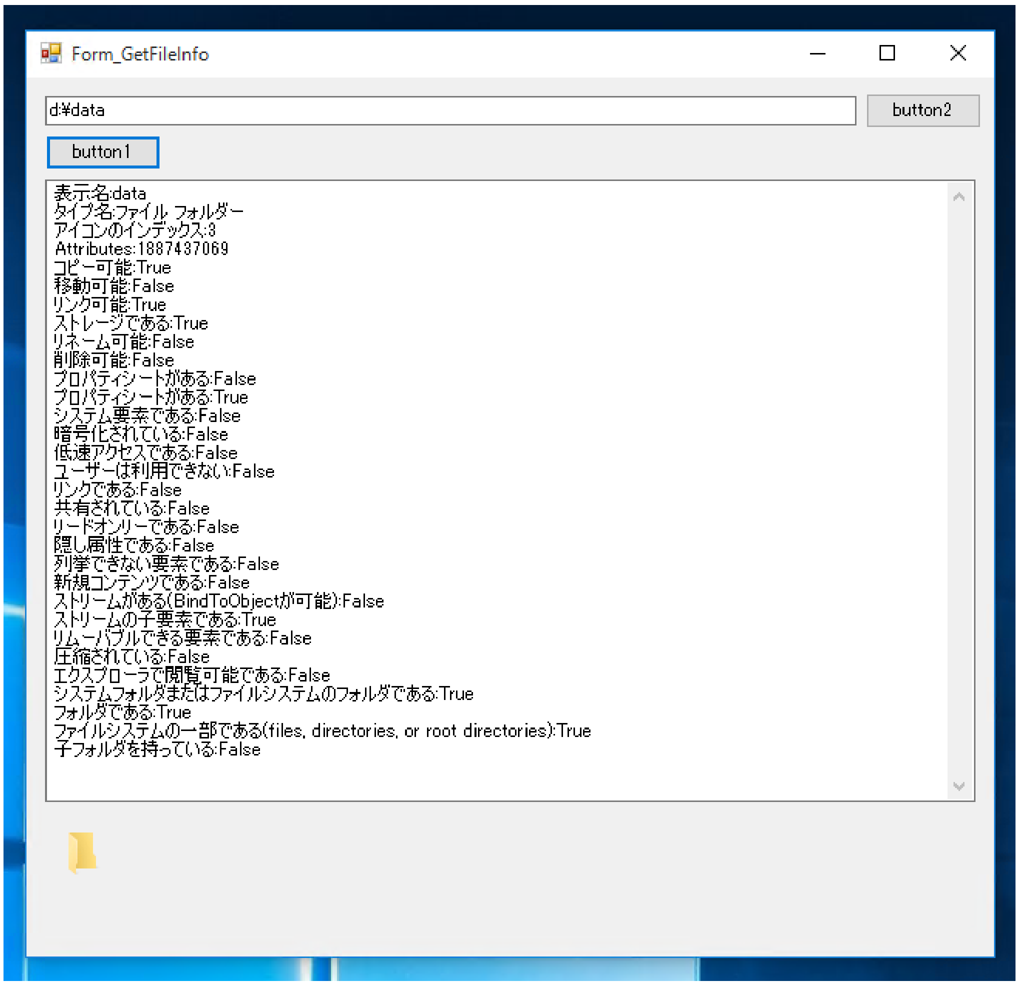
```
表示名:design.xlsx
タイプ名:Microsoft Excel ワークシート
アイコンのインデックス:203
Attributes:1073742085
コピー可能:True
移動可能:False
リンク可能:True
ストレージである:False
リネーム可能:False
削除可能:False
```

ファイルの情報は以下の通りです。



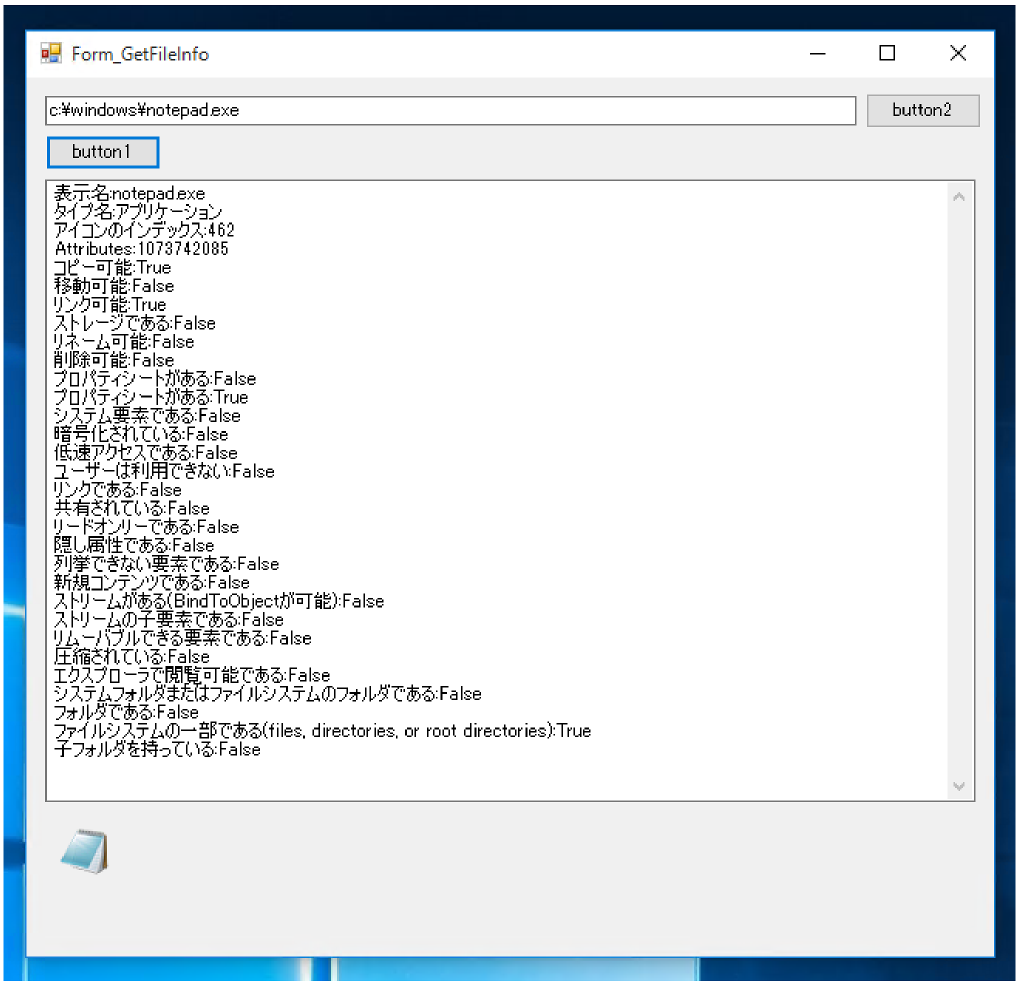
```
表示名:design.xlsx
タイプ名:Microsoft Excel ワークシート
アイコンのインデックス:203
Attributes:1073742085
コピー可能:True
移動可能:False
リンク可能:True
ストレージである:False
リネーム可能:False
削除可能:False
プロパティシートがある:False
プロパティシートがある:True
システム要素である:False
暗号化されている:False
低速アクセスである:False
ユーザーは利用できない:False
リンクである:False
共有されている:False
リードオンリーである:False
隠し属性である:False
列挙できない要素である:False
新規コンテンツである:False
ストリームがある(BindToObjectが可能):False
ストリームの子要素である:False
リムーバブルできる要素である:False
圧縮されている:False
エクスプローラで閲覧可能である:False
システムフォルダまたはファイルシステムのフォルダである:False
フォルダである:False
ファイルシステムの一部である(files, directories, or root directories):True
子フォルダを持っている:False
```

ディレクトリの情報を取得した場合は、下図となります。



```
表示名:data
タイプ名:ファイル フォルダー
アイコンのインデックス:3
Attributes:1887437069
コピー可能:True
移動可能:False
リンク可能:True
ストレージである:True
リネーム可能:False
削除可能:False
プロパティシートがある:False
プロパティシートがある:True
システム要素である:False
暗号化されている:False
低速アクセスである:False
ユーザーは利用できない:False
リンクである:False
共有されている:False
リードオンリーである:False
隠し属性である:False
列挙できない要素である:False
新規コンテンツである:False
ストリームがある(BindToObjectが可能):False
ストリームの子要素である:True
リムーバブルできる要素である:False
圧縮されている:False
エクスプローラで閲覧可能である:False
システムフォルダまたはファイルシステムのフォルダである:True
フォルダである:True
ファイルシステムの一部である(files, directories, or root directories):True
子フォルダを持っている:False
```

メモ帳の実行ファイルの情報を取得した場合は、下図となります。



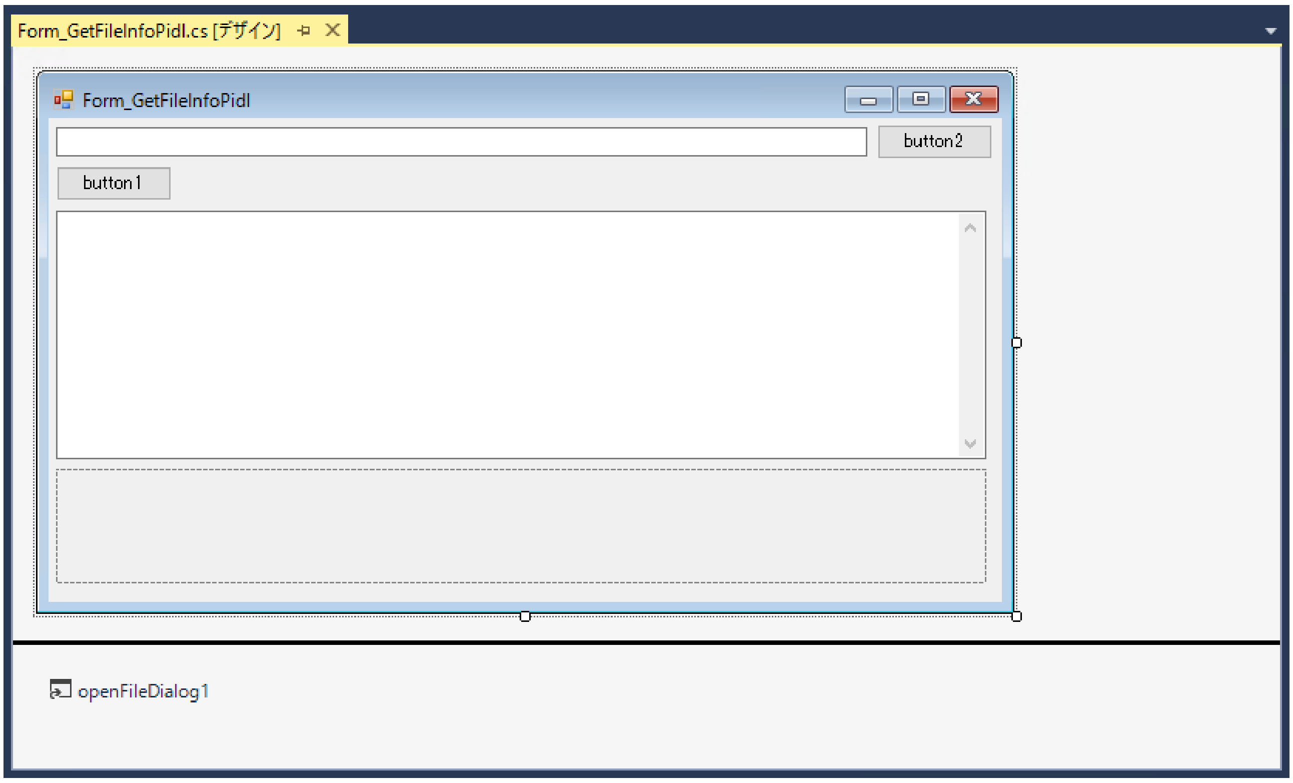
```
表示名:notepad.exe
タイプ名:アプリケーション
アイコンのインデックス:462
Attributes:1073742085
コピー可能:True
移動可能:False
リンク可能:True
ストレージである:False
リネーム可能:False
削除可能:False
プロパティシートがある:False
プロパティシートがある:True
システム要素である:False
暗号化されている:False
低速アクセスである:False
ユーザーは利用できない:False
リンクである:False
共有されている:False
リードオンリーである:False
隠し属性である:False
列挙できない要素である:False
新規コンテンツである:False
ストリームがある(BindToObjectが可能):False
ストリームの子要素である:False
リムーバブルできる要素である:False
圧縮されている:False
エクスプローラで閲覧可能である:False
システムフォルダまたはファイルシステムのフォルダである:False
フォルダである:False
ファイルシステムの一部である(files, directories, or root directories):True
子フォルダを持っている:False
```

PIDL(アイテムID)から情報を取得する場合

PIDL(アイテムID)から情報を取得するコードを紹介します。

UI

下図のUIを作成します。
TextBox,TextBox複数行, Buttonを2つ、Panelを1つ、OpenFileDialog を配置します。



コード

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using ShellNamespace;
using System.Runtime.InteropServices;

namespace ShellNamespaceDemo
{
  public partial class Form_GetFileInfoPidl : Form
  {
    private WindowsAPI.IShellFolder m_DesktopShellFolder = null;

    public Form_GetFileInfoPidl()
    {
      InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e)
    {
      string PathString = textBox1.Text;// @"c:\windows\notepad.exe";

      int hRes = WindowsAPI.SHGetDesktopFolder(ref m_DesktopShellFolder);
      if (hRes != 0) {
        Marshal.ThrowExceptionForHR(hRes);
      }

      uint pchEaten;
      IntPtr ppidl;
      WindowsAPI.SFGAOF attr = WindowsAPI.SFGAOF.SFGAO_FOLDER | WindowsAPI.SFGAOF.SFGAO_HASSUBFOLDER
        | WindowsAPI.SFGAOF.SFGAO_STREAM | WindowsAPI.SFGAOF.SFGAO_DROPTARGET | WindowsAPI.SFGAOF.SFGAO_FILESYSTEM | WindowsAPI.SFGAOF.SFGA
      m_DesktopShellFolder.ParseDisplayName(IntPtr.Zero, IntPtr.Zero, PathString, out pchEaten, out ppidl, ref attr);


      WindowsAPI.SHFILEINFO shinfo = new WindowsAPI.SHFILEINFO();
      WindowsAPI.SHGFI flag =
        WindowsAPI.SHGFI.SHGFI_ATTRIBUTES
        | WindowsAPI.SHGFI.SHGFI_ATTR_SPECIFIED
        | WindowsAPI.SHGFI.SHGFI_DISPLAYNAME
        | WindowsAPI.SHGFI.SHGFI_EXETYPE
        | WindowsAPI.SHGFI.SHGFI_ICON
        | WindowsAPI.SHGFI.SHGFI_ICONLOCATION
        | WindowsAPI.SHGFI.SHGFI_LARGEICON
        | WindowsAPI.SHGFI.SHGFI_OPENICON
        | WindowsAPI.SHGFI.SHGFI_OVERLAYINDEX
        | WindowsAPI.SHGFI.SHGFI_PIDL
        | WindowsAPI.SHGFI.SHGFI_SHELLICONSIZE
        | WindowsAPI.SHGFI.SHGFI_SYSICONINDEX
        | WindowsAPI.SHGFI.SHGFI_TYPENAME
        ;

      IntPtr ret = WindowsAPI.SHGetFileInfo(ppidl, 0, out shinfo, (uint)Marshal.SizeOf(typeof(WindowsAPI.SHFILEINFO)), flag);

      if (ret == IntPtr.Zero) {
        Application.Exit();
      }

      textBox2.Text += string.Format("表示名:{0}\r\n", shinfo.szDisplayName);
      textBox2.Text += string.Format("タイプ名:{0}\r\n", shinfo.szTypeName);
      textBox2.Text += string.Format("アイコンのインデックス:{0:d}\r\n", shinfo.iIcon);
      textBox2.Text += string.Format("Attributes:{0:d}\r\n", shinfo.dwAttributes);

      bool cancopy = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_CANCOPY) == (uint)WindowsAPI.SFGAOF.SFGAO_CANCOPY) {
        cancopy = true;
      }
      textBox2.Text += string.Format("コピー可能:{0:b}\r\n", cancopy);

      bool canmove = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_CANMOVE) == (uint)WindowsAPI.SFGAOF.SFGAO_CANMOVE) {
        canmove = true;
      }
      textBox2.Text += string.Format("移動可能:{0:b}\r\n", canmove);

      bool canlink = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_CANLINK) == (uint)WindowsAPI.SFGAOF.SFGAO_CANLINK) {
        canlink = true;
      }
      textBox2.Text += string.Format("リンク可能:{0:b}\r\n", canlink);

      bool isstorage = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_STORAGE) == (uint)WindowsAPI.SFGAOF.SFGAO_STORAGE) {
        isstorage = true;
      }
      textBox2.Text += string.Format("ストレージである:{0:b}\r\n", isstorage);

      bool canrename = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_CANRENAME) == (uint)WindowsAPI.SFGAOF.SFGAO_CANRENAME) {
        canrename = true;
      }
      textBox2.Text += string.Format("リネーム可能:{0:b}\r\n", canrename);

      bool candelete = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_CANDELETE) == (uint)WindowsAPI.SFGAOF.SFGAO_CANDELETE) {
        candelete = true;
      }
      textBox2.Text += string.Format("削除可能:{0:b}\r\n", candelete);

      bool haspropsheet = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_HASPROPSHEET) == (uint)WindowsAPI.SFGAOF.SFGAO_HASPROPSHEET) {
        haspropsheet = true;
      }
      textBox2.Text += string.Format("プロパティシートがある:{0:b}\r\n", haspropsheet);

      bool droptarget = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_DROPTARGET) == (uint)WindowsAPI.SFGAOF.SFGAO_DROPTARGET) {
        droptarget = true;
      }
      textBox2.Text += string.Format("プロパティシートがある:{0:b}\r\n", droptarget);

      bool issystem = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_SYSTEM) == (uint)WindowsAPI.SFGAOF.SFGAO_SYSTEM) {
        issystem = true;
      }
      textBox2.Text += string.Format("システム要素である:{0:b}\r\n", issystem);

      bool encrypted = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_ENCRYPTED) == (uint)WindowsAPI.SFGAOF.SFGAO_ENCRYPTED) {
        encrypted = true;
      }
      textBox2.Text += string.Format("暗号化されている:{0:b}\r\n", encrypted);

      bool isslow = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_ISSLOW) == (uint)WindowsAPI.SFGAOF.SFGAO_ISSLOW) {
        isslow = true;
      }
      textBox2.Text += string.Format("低速アクセスである:{0:b}\r\n", isslow);

      bool ghosted = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_GHOSTED) == (uint)WindowsAPI.SFGAOF.SFGAO_GHOSTED) {
        ghosted = true;
      }
      textBox2.Text += string.Format("ユーザーは利用できない:{0:b}\r\n", ghosted);

      bool islink = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_LINK) == (uint)WindowsAPI.SFGAOF.SFGAO_LINK) {
        islink = true;
      }
      textBox2.Text += string.Format("リンクである:{0:b}\r\n", islink);

      bool shared = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_SHARE) == (uint)WindowsAPI.SFGAOF.SFGAO_SHARE) {
        shared = true;
      }
      textBox2.Text += string.Format("共有されている:{0:b}\r\n", shared);

      bool isreadonly = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_READONLY) == (uint)WindowsAPI.SFGAOF.SFGAO_READONLY) {
        isreadonly = true;
      }
      textBox2.Text += string.Format("リードオンリーである:{0:b}\r\n", isreadonly);

      bool ishidden = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_HIDDEN) == (uint)WindowsAPI.SFGAOF.SFGAO_HIDDEN) {
        ishidden = true;
      }
      textBox2.Text += string.Format("隠し属性である:{0:b}\r\n", ishidden);

      bool nonenumerated = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_NONENUMERATED) == (uint)WindowsAPI.SFGAOF.SFGAO_NONENUMERATED) {
        nonenumerated = true;
      }
      textBox2.Text += string.Format("列挙できない要素である:{0:b}\r\n", nonenumerated);

      bool newcontent = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_NEWCONTENT) == (uint)WindowsAPI.SFGAOF.SFGAO_NEWCONTENT) {
        newcontent = true;
      }
      textBox2.Text += string.Format("新規コンテンツである:{0:b}\r\n", newcontent);

      bool isstream = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_STREAM) == (uint)WindowsAPI.SFGAOF.SFGAO_STREAM) {
        isstream = true;
      }
      textBox2.Text += string.Format("ストリームがある(BindToObjectが可能):{0:b}\r\n", isstream);

      bool storageancestor = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_STORAGEANCESTOR) == (uint)WindowsAPI.SFGAOF.SFGAO_STORAGEANCESTOR) {
        storageancestor = true;
      }
      textBox2.Text += string.Format("ストリームの子要素である:{0:b}\r\n", storageancestor);

      bool removable = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_REMOVABLE) == (uint)WindowsAPI.SFGAOF.SFGAO_REMOVABLE) {
        removable = true;
      }
      textBox2.Text += string.Format("リムーバブルできる要素である:{0:b}\r\n", removable);

      bool compressed = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_COMPRESSED) == (uint)WindowsAPI.SFGAOF.SFGAO_COMPRESSED) {
        compressed = true;
      }
      textBox2.Text += string.Format("圧縮されている:{0:b}\r\n", compressed);

      bool browsable = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_BROWSABLE) == (uint)WindowsAPI.SFGAOF.SFGAO_BROWSABLE) {
        browsable = true;
      }
      textBox2.Text += string.Format("エクスプローラで閲覧可能である:{0:b}\r\n", browsable);

      bool fileancestor = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_FILESYSANCESTOR) == (uint)WindowsAPI.SFGAOF.SFGAO_FILESYSANCESTOR) {
        fileancestor = true;
      }
      textBox2.Text += string.Format("システムフォルダまたはファイルシステムのフォルダである:{0:b}\r\n", fileancestor);

      bool isfolder = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_FOLDER) == (uint)WindowsAPI.SFGAOF.SFGAO_FOLDER) {
        isfolder = true;
      }
      textBox2.Text += string.Format("フォルダである:{0:b}\r\n", isfolder);

      bool isfilesystem = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_FILESYSTEM) == (uint)WindowsAPI.SFGAOF.SFGAO_FILESYSTEM) {
        isfilesystem = true;
      }
      textBox2.Text += string.Format("ファイルシステムの一部である(files, directories, or root directories):{0:b}\r\n", isfilesystem);

      bool hassubfolder = false;
      if ((shinfo.dwAttributes & (uint)WindowsAPI.SFGAOF.SFGAO_HASSUBFOLDER) == (uint)WindowsAPI.SFGAOF.SFGAO_HASSUBFOLDER) {
        hassubfolder = true;
      }
      textBox2.Text += string.Format("子フォルダを持っている:{0:b}\r\n", hassubfolder);

      if (shinfo.hIcon != IntPtr.Zero) {
        Icon myIcon = Icon.FromHandle(shinfo.hIcon);
        Graphics g = Graphics.FromHwnd(panel1.Handle);
        g.DrawIcon(myIcon, 10, 10);
      }
    }

    private void button2_Click(object sender, EventArgs e)
    {
      if (openFileDialog1.ShowDialog() == DialogResult.OK) {
        textBox1.Text = openFileDialog1.FileName;
      }

    }
  }
}
```

解説

```
     string PathString = textBox1.Text;// @"c:\windows\notepad.exe";

      int hRes = WindowsAPI.SHGetDesktopFolder(ref m_DesktopShellFolder);
      if (hRes != 0) {
        Marshal.ThrowExceptionForHR(hRes);
      }

      uint pchEaten;
      IntPtr ppidl;
      WindowsAPI.SFGAOF attr = WindowsAPI.SFGAOF.SFGAO_FOLDER | WindowsAPI.SFGAOF.SFGAO_HASSUBFOLDER
        | WindowsAPI.SFGAOF.SFGAO_STREAM | WindowsAPI.SFGAOF.SFGAO_DROPTARGET | WindowsAPI.SFGAOF.SFGAO_FILESYSTEM | WindowsAPI.SFGAOF.SFGA
      m_DesktopShellFolder.ParseDisplayName(IntPtr.Zero, IntPtr.Zero, PathString, out pchEaten, out ppidl, ref attr);
```

上記のコードが、ファイルパスからPIDLを取得するコードになります。PIDLの取得には IShellFolder.ParseDisplayName メソッドを利用しています。

```
      WindowsAPI.SHFILEINFO shinfo = new WindowsAPI.SHFILEINFO();
      WindowsAPI.SHGFI flag =
        WindowsAPI.SHGFI.SHGFI_ATTRIBUTES
        | WindowsAPI.SHGFI.SHGFI_ATTR_SPECIFIED
        | WindowsAPI.SHGFI.SHGFI_DISPLAYNAME
        | WindowsAPI.SHGFI.SHGFI_EXETYPE
        | WindowsAPI.SHGFI.SHGFI_ICON
```

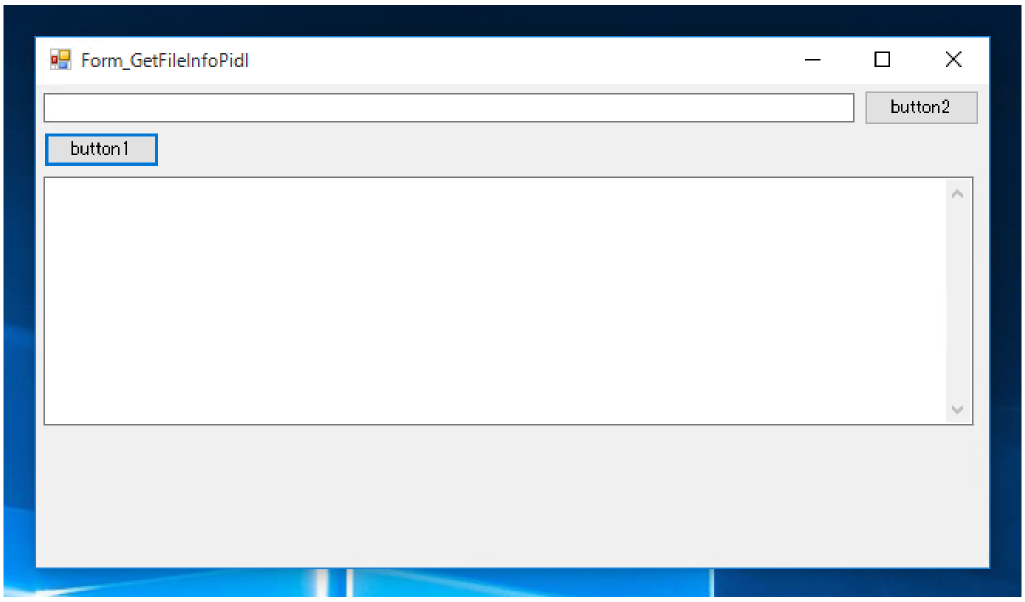
```
            | WindowsAPI.SHGFI.SHGFI_ICONLOCATION
            | WindowsAPI.SHGFI.SHGFI_LARGEICON
            | WindowsAPI.SHGFI.SHGFI_OPENICON
            | WindowsAPI.SHGFI.SHGFI_OVERLAYINDEX
            | WindowsAPI.SHGFI.SHGFI_PIDL
            | WindowsAPI.SHGFI.SHGFI_SHELLICONSIZE
            | WindowsAPI.SHGFI.SHGFI_SYSICONINDEX
            | WindowsAPI.SHGFI.SHGFI_TYPENAME
        ;

      IntPtr ret = WindowsAPI.SHGetFileInfo(ppidl, 0, out shinfo, (uint)Marshal.SizeOf(typeof(WindowsAPI.SHFILEINFO)), flag);
```

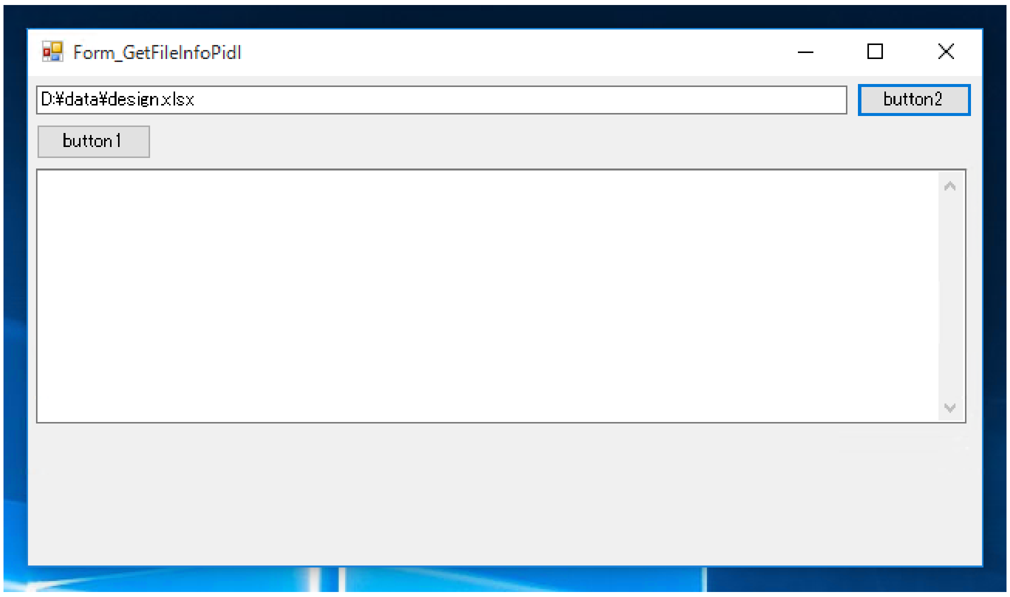
取得したPIDLから、ファイルの情報を取得します。PIDL(ITEMID)を用いる場合は、第一引数にPIDL(ITEMID)を与え、SHGetFileInfoの第5引数に、SHGFI_PIDLを指定する必要があります。SHGFI_PIDLを指定しないと、第一引数の値はファイルパスとして処理されるため、エラーになります。
その他のコードは、先のファイルパスを与えて、SHGetFileInfo()を呼び出す場合のコードと同様です。

実行結果

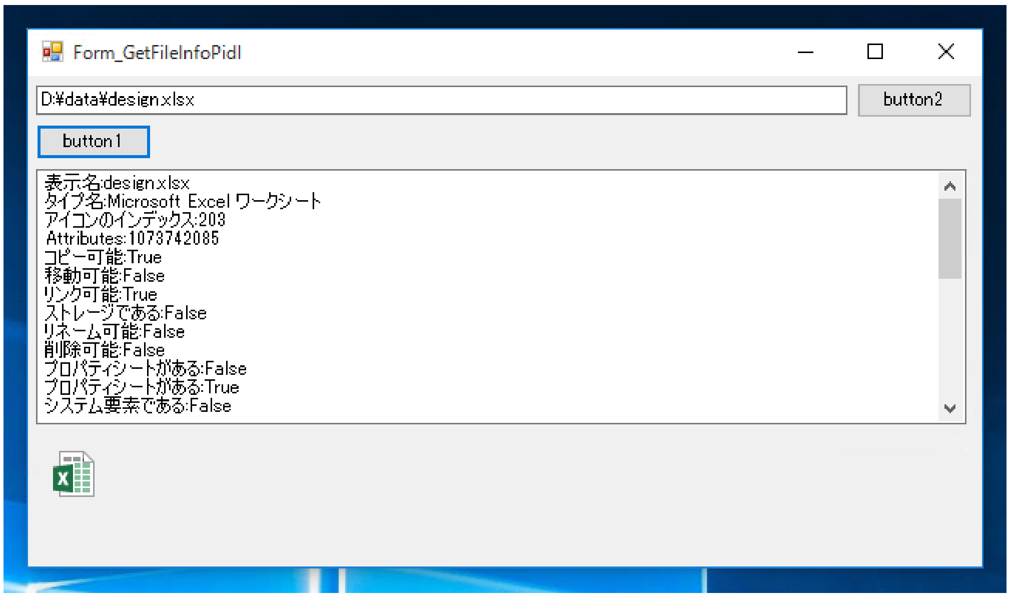
プロジェクトを実行します。下図のウィンドウが表示されます。



[button2]をクリックしてダイアログからファイルを選択するか、直接上部のテキストボックスに情報を取得したいファイルのフルパスを入力します。



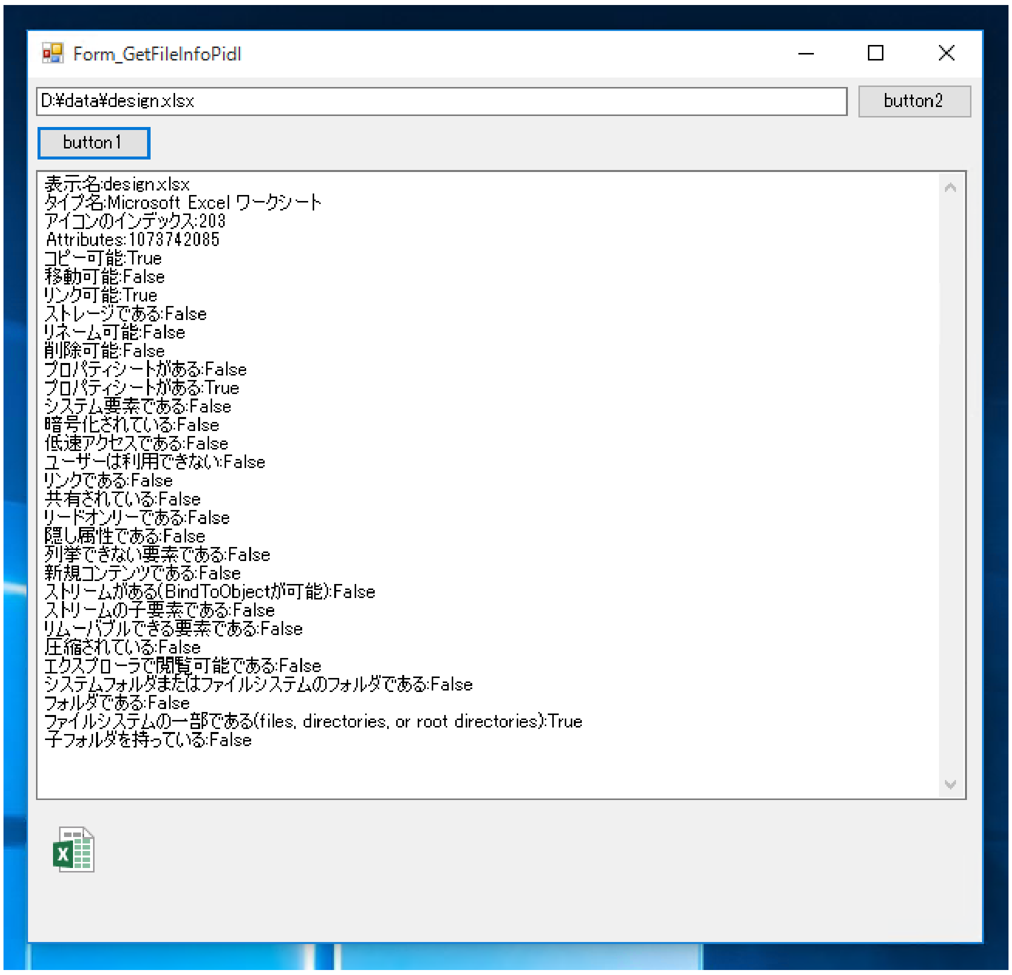
入力ができたら[button1]をクリックします。ファイルの情報が表示されます。ウィンドウ下部のパネルにはファイルのアイコンも表示されます。



```
D:¥data¥design.xlsx
```

```
表示名:design.xlsx
タイプ名:Microsoft Excel ワークシート
アイコンのインデックス:203
Attributes:1073742085
コピー可能:True
移動可能:False
リンク可能:True
ストレージである:False
リネーム可能:False
削除可能:False
プロパティシートがある:False
プロパティシートがある:True
システム要素である:False
```

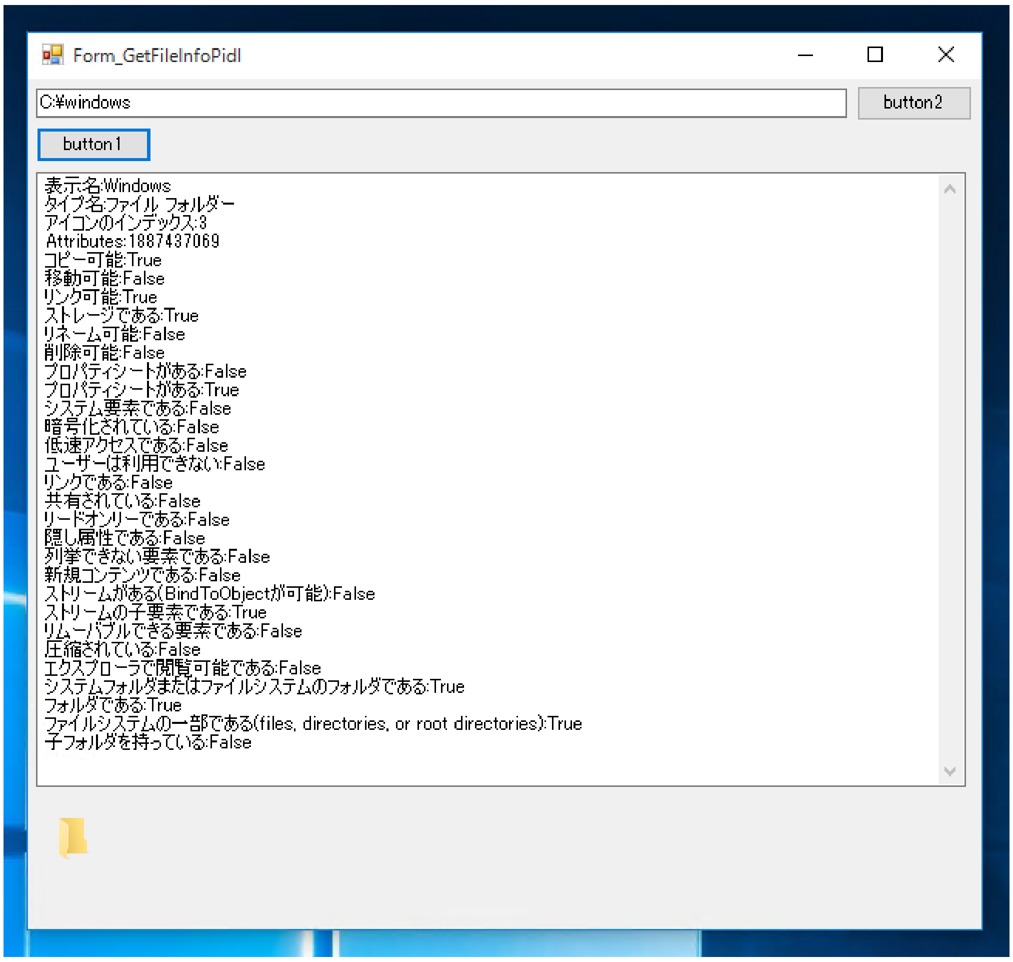
Excelワークシートを指定した場合の結果です。



```
D:¥data¥design.xlsx
```

```
表示名:design.xlsx
タイプ名:Microsoft Excel ワークシート
アイコンのインデックス:203
Attributes:1073742085
コピー可能:True
移動可能:False
リンク可能:True
ストレージである:False
リネーム可能:False
削除可能:False
プロパティシートがある:False
プロパティシートがある:True
システム要素である:False
暗号化されている:False
低速アクセスである:False
ユーザーは利用できない:False
リンクである:False
共有されている:False
リードオンリーである:False
隠し属性である:False
列挙できない要素である:False
新規コンテンツである:False
ストリームがある(BindToObjectが可能):False
ストリームの子要素である:False
リムーバブルできる要素である:False
圧縮されている:False
エクスプローラで閲覧可能である:False
システムフォルダまたはファイルシステムのフォルダである:False
フォルダである:False
ファイルシステムの一部である(files, directories, or root directories):True
子フォルダを持っている:False
```

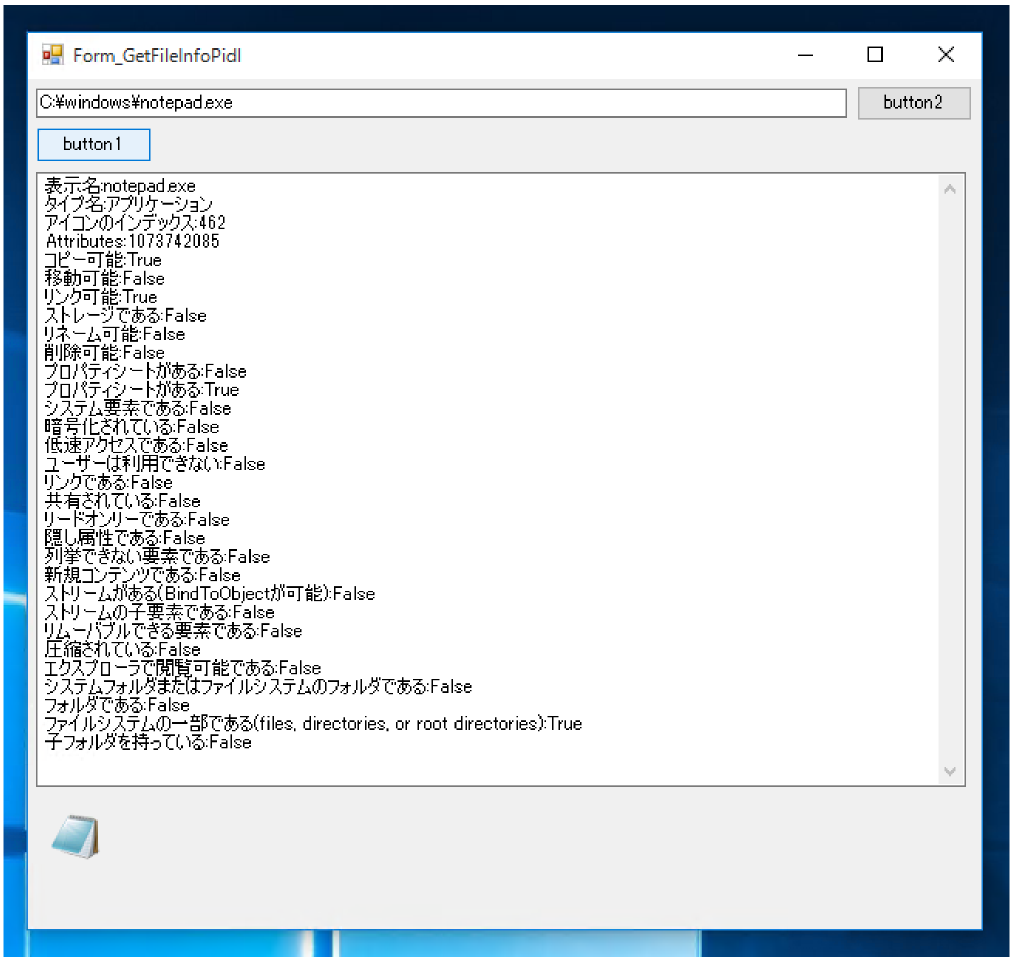
ファイルディレクトリを指定した場合の結果です。



```
C:¥windows
```

```
表示名:Windows
タイプ名:ファイル フォルダー
アイコンのインデックス:3
Attributes:1887437069
コピー可能:True
移動可能:False
リンク可能:True
ストレージである:True
リネーム可能:False
削除可能:False
プロパティシートがある:False
プロパティシートがある:True
システム要素である:False
暗号化されている:False
低速アクセスである:False
ユーザーは利用できない:False
リンクである:False
共有されている:False
リードオンリーである:False
隠し属性である:False
列挙できない要素である:False
新規コンテンツである:False
ストリームがある(BindToObjectが可能):False
ストリームの子要素である:True
リムーバブルできる要素である:False
圧縮されている:False
エクスプローラで閲覧可能である:False
システムフォルダまたはファイルシステムのフォルダである:True
フォルダである:True
ファイルシステムの一部である(files, directories, or root directories):True
子フォルダを持っている:False
```

実行ファイル(メモ帳)を指定した場合の結果です。



```
C:¥windows¥notepad.exe
```

```
表示名:notepad.exe
タイプ名:アプリケーション
アイコンのインデックス:462
Attributes:1073742085
コピー可能:True
移動可能:False
リンク可能:True
ストレージである:False
リネーム可能:False
削除可能:False
プロパティシートがある:False
プロパティシートがある:True
システム要素である:False
暗号化されている:False
低速アクセスである:False
ユーザーは利用できない:False
リンクである:False
共有されている:False
リードオンリーである:False
隠し属性である:False
列挙できない要素である:False
新規コンテンツである:False
ストリームがある(BindToObjectが可能):False
ストリームの子要素である:False
リムーバブルできる要素である:False
圧縮されている:False
エクスプローラで閲覧可能である:False
システムフォルダまたはファイルシステムのフォルダである:False
フォルダである:False
ファイルシステムの一部である(files, directories, or root directories):True
子フォルダを持っている:False
```

ファイルパスや、PIDL(ITEM ID)からファイルの情報を取得できました。

ライブラリ部のコード

WindowsAPI.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Runtime.InteropServices;

namespace ShellNamespace
{
  public static partial class WindowsAPI
  {
    public const Int32 FILE_ATTRIBUTE_NORMAL = 0x80;
    public static Guid IID_IShellFolder = new Guid("000214E6-0000-0000-C000-000000000046");

    [DllImport("user32.dll", CharSet = CharSet.Auto)]
    public static extern Int32 SendMessage(IntPtr pWnd, UInt32 uMsg, UInt32 wParam, IntPtr lParam);

    [DllImport("shell32.dll", CharSet = CharSet.Auto)]
    public static extern Int32 SHGetDesktopFolder(ref IShellFolder ppshf);

    [DllImport("shell32.dll", CharSet = CharSet.Auto)]
    public static extern IntPtr SHGetFileInfo(string pszPath, uint dwFileAttribs, out SHFILEINFO psfi, uint cbFileInfo, SHGFI uFlag

    [DllImport("shell32.dll", CharSet = CharSet.Auto)]
    public static extern IntPtr SHGetFileInfo(IntPtr pIDL, uint dwFileAttributes, out SHFILEINFO psfi, uint cbFileInfo, SHGFI uFlags

    [DllImport("shell32.dll", CharSet = CharSet.Auto)]
    public static extern Int32 SHGetSpecialFolderLocation(IntPtr hwndOwner, CSIDL nFolder, ref IntPtr ppidl);

    [DllImport("shell32.dll", CharSet = CharSet.Auto)]
    public static extern IntPtr ILCombine(IntPtr pIDLParent, IntPtr pIDLChild);

    [DllImport("shell32.dll", CharSet = CharSet.Auto)]
    public static extern IntPtr ILClone(IntPtr pidl);

    [DllImport("shell32.dll", CharSet = CharSet.Auto)]
    public static extern Int32 SHGetPathFromIDList(IntPtr pIDL, StringBuilder strPath);

    [DllImport("shell32.dll", CharSet = CharSet.Auto)]
    public static extern int SHGetFolderLocation(IntPtr hwndOwner, int nFolder, IntPtr hToken, uint dwReserved, out IntPtr ppidl);

    //[DllImport("shell32.dll")]
    //public static extern int SHGetDataFromIDList(IShellFolder psf, ref IntPtr pidl, SHGDFIL nFormat, IntPtr pv, int cb);

    [DllImport("shell32.dll", CharSet = CharSet.Auto)]
    public static extern int SHGetDataFromIDList(IShellFolder psf, IntPtr pidl, SHGDFIL nFormat, out WIN32_FIND_DATA pv, int cb);

    [DllImport("shell32.dll", EntryPoint = "#727")]
    public static extern int SHGetImageList(SHIL iImageList, ref Guid riid, out IntPtr ppv);


    //IMAGE LIST
    public static Guid IID_IImageList = new Guid("46EB5926-582E-4017-9FDF-E8998DAA0950");
    public static Guid IID_IImageList2 = new Guid("192B9D83-50FC-457B-90A0-2B82A8B5DAE1");

    [Flags]
    public enum SHIL
    {
      SHIL_JUMBO = 0x0004,
      SHIL_EXTRALARGE = 0x0002
    }

    /*
    [DllImport("shell32.dll", CharSet = CharSet.Auto)]
    public static extern uint SHGetDataFromIDList(IShellFolder psf, IntPtr pidl, SHGDFIL nFormat, out WIN32_FIND_DATA pv, int cb);
    */
    /*
    public struct ITEMIDLIST
    {
      public SHITEMID mkid;
    }

    [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode)]
    public struct SHITEMID
    {
      public ushort cb; // The size of identifier, in bytes, including cb itself.

      //[MarshalAs(UnmanagedType.ByValArray, SizeConst = 1)]
      [MarshalAs(UnmanagedType.LPStr)]
      public byte[] abID; // A variable-length item identifier.
    }
    */

    [Flags()]
    public enum SHGDFIL :int
    {
      SHGDFIL_FINDDATA = 1,
      SHGDFIL_NETRESOURCE = 2,
      SHGDFIL_DESCRIPTIONID = 3
    }
```

```
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Auto)]
public struct WIN32_FIND_DATA
{
  public uint dwFileAttributes;
  public System.Runtime.InteropServices.ComTypes.FILETIME ftCreationTime;
  public System.Runtime.InteropServices.ComTypes.FILETIME ftLastAccessTime;
  public System.Runtime.InteropServices.ComTypes.FILETIME ftLastWriteTime;
  public uint nFileSizeHigh;
  public uint nFileSizeLow;
  public uint dwReserved0;
  public uint dwReserved1;
  [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 260)]
  public string cFileName;
  [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 14)]
  public string cAlternateFileName;
}


[Flags]
public enum SHGDN : uint
{
  SHGDN_NORMAL = 0x0000,                  // Default (display purpose)
  SHGDN_INFOLDER = 0x0001,                // Displayed under a folder (relative)
  SHGDN_FOREDITING = 0x1000,              // For in-place editing
  SHGDN_FORADDRESSBAR = 0x4000,           // UI friendly parsing name (remove ugly stuff)
  SHGDN_FORPARSING = 0x8000,              // Parsing name for ParseDisplayName()
}

[Flags]
public enum SHCONTF : uint
{
  SHCONTF_CHECKING_FOR_CHILDREN = 0x0010,
  SHCONTF_FOLDERS = 0x0020,               // Only want folders enumerated (SFGAO_FOLDER)
  SHCONTF_NONFOLDERS = 0x0040,            // Include non folders
  SHCONTF_INCLUDEHIDDEN = 0x0080,         // Show items normally hidden
  SHCONTF_INIT_ON_FIRST_NEXT = 0x0100,    // Allow EnumObject() to return before validating enum
  SHCONTF_NETPRINTERSRCH = 0x0200,        // Hint that client is looking for printers
  SHCONTF_SHAREABLE = 0x0400,             // Hint that client is looking sharable resources (remote shares)
  SHCONTF_STORAGE = 0x0800,               // Include all items with accessible storage and their ancestors
  SHCONTF_NAVIGATION_ENUM = 0x01000,
  SHCONTF_FASTITEMS = 0x02000,
  SHCONTF_FLATLIST = 0x04000,
  SHCONTF_ENABLE_ASYNC = 0x08000,
  SHCONTF_INCLUDESUPERHIDDEN = 0x10000,
}

[Flags]
public enum SFGAOF : uint
{
  SFGAO_CANCOPY = 0x1,                    // Objects can be copied  (DROPEFFECT_COPY)
  SFGAO_CANMOVE = 0x2,                    // Objects can be moved   (DROPEFFECT_MOVE)
  SFGAO_CANLINK = 0x4,                    // Objects can be linked  (DROPEFFECT_LINK)
  SFGAO_STORAGE = 0x00000008,             // Supports BindToObject(IID_IStorage)
  SFGAO_CANRENAME = 0x00000010,           // Objects can be renamed
  SFGAO_CANDELETE = 0x00000020,           // Objects can be deleted
  SFGAO_HASPROPSHEET = 0x00000040,        // Objects have property sheets
  SFGAO_DROPTARGET = 0x00000100,          // Objects are drop target
  SFGAO_CAPABILITYMASK = 0x00000177,
  SFGAO_SYSTEM = 0x00001000,
  SFGAO_ENCRYPTED = 0x00002000,           // Object is encrypted (use alt color)
  SFGAO_ISSLOW = 0x00004000,              // 'Slow' object
  SFGAO_GHOSTED = 0x00008000,             // Ghosted icon
  SFGAO_LINK = 0x00010000,                // Shortcut (link)
  SFGAO_SHARE = 0x00020000,               // Shared
  SFGAO_READONLY = 0x00040000,            // Read-only
  SFGAO_HIDDEN = 0x00080000,              // Hidden object
  SFGAO_DISPLAYATTRMASK = 0x000FC000,
  SFGAO_FILESYSANCESTOR = 0x10000000,     // May contain children with SFGAO_FILESYSTEM
  SFGAO_FOLDER = 0x20000000,              // Support BindToObject(IID_IShellFolder)
  SFGAO_FILESYSTEM = 0x40000000,          // Is a win32 file system object (file/folder/root)
  SFGAO_HASSUBFOLDER = 0x80000000,        // May contain children with SFGAO_FOLDER
  SFGAO_CONTENTSMASK = 0x80000000,
  SFGAO_VALIDATE = 0x01000000,            // Invalidate cached information
  SFGAO_REMOVABLE = 0x02000000,           // Is this removeable media?
  SFGAO_COMPRESSED = 0x04000000,          // Object is compressed (use alt color)
  SFGAO_BROWSABLE = 0x08000000,           // Supports IShellFolder, but only implements CreateViewObject() (non-folder view)
  SFGAO_NONENUMERATED = 0x00100000,       // Is a non-enumerated object
  SFGAO_NEWCONTENT = 0x00200000,          // Should show bold in explorer tree
  SFGAO_CANMONIKER = 0x00400000,          // Defunct
  SFGAO_HASSTORAGE = 0x00400000,          // Defunct
  SFGAO_STREAM = 0x00400000,              // Supports BindToObject(IID_IStream)
  SFGAO_STORAGEANCESTOR = 0x00800000,     // May contain children with SFGAO_STORAGE or SFGAO_STREAM
  SFGAO_STORAGECAPMASK = 0x70C50008,      // For determining storage capabilities, ie for open/save semantics
}

[Flags]
public enum STRRET : uint
{
  STRRET_WSTR = 0,
  STRRET_OFFSET = 0x1,
  STRRET_CSTR = 0x2,
}

[Flags]
public enum SHGFI
{
  SHGFI_ICON = 0x000000100,
  SHGFI_DISPLAYNAME = 0x000000200,
  SHGFI_TYPENAME = 0x000000400,
  SHGFI_ATTRIBUTES = 0x000000800,
  SHGFI_ICONLOCATION = 0x000001000,
  SHGFI_EXETYPE = 0x000002000,
  SHGFI_SYSICONINDEX = 0x000004000,
  SHGFI_LINKOVERLAY = 0x000008000,
  SHGFI_SELECTED = 0x000010000,
  SHGFI_ATTR_SPECIFIED = 0x000020000,
  SHGFI_LARGEICON = 0x000000000,
  SHGFI_SMALLICON = 0x000000001,
  SHGFI_OPENICON = 0x000000002,
  SHGFI_SHELLICONSIZE = 0x000000004,
  SHGFI_PIDL = 0x000000008,
  SHGFI_USEFILEATTRIBUTES = 0x000000010,
    SHGFI_ADDOVERLAYS = 0x000000020,
    SHGFI_OVERLAYINDEX = 0x000000040
  }

  /*
  [Flags]
  public enum SHGDNF : uint
  {
    SHGDN_NORMAL = 0,
    SHGDN_INFOLDER = 0x1,
    SHGDN_FOREDITING = 0x1000,
    SHGDN_FORADDRESSBAR = 0x4000,
    SHGDN_FORPARSING = 0x8000
  }
  */

  [Flags]
  public enum CSIDL : uint
  {
    CSIDL_DESKTOP = 0x0000,    // <desktop>
    CSIDL_INTERNET = 0x0001,    // Internet Explorer (icon on desktop)
    CSIDL_PROGRAMS = 0x0002,    // Start Menu\Programs
    CSIDL_CONTROLS = 0x0003,    // My Computer\Control Panel
    CSIDL_PRINTERS = 0x0004,    // My Computer\Printers
    CSIDL_PERSONAL = 0x0005,    // My Documents
    CSIDL_FAVORITES = 0x0006,    // <user name>\Favorites
    CSIDL_STARTUP = 0x0007,    // Start Menu\Programs\Startup
    CSIDL_RECENT = 0x0008,    // <user name>\Recent
    CSIDL_SENDTO = 0x0009,    // <user name>\SendTo
    CSIDL_BITBUCKET = 0x000a,    // <desktop>\Recycle Bin
    CSIDL_STARTMENU = 0x000b,    // <user name>\Start Menu
    CSIDL_MYDOCUMENTS = 0x000c,    // logical "My Documents" desktop icon
    CSIDL_MYMUSIC = 0x000d,    // "My Music" folder
    CSIDL_MYVIDEO = 0x000e,    // "My Videos" folder
    CSIDL_DESKTOPDIRECTORY = 0x0010,    // <user name>\Desktop
    CSIDL_DRIVES = 0x0011,    // My Computer
    CSIDL_NETWORK = 0x0012,    // Network Neighborhood (My Network Places)
    CSIDL_NETHOOD = 0x0013,    // <user name>\nethood
    CSIDL_FONTS = 0x0014,    // windows\fonts
    CSIDL_TEMPLATES = 0x0015,
    CSIDL_COMMON_STARTMENU = 0x0016,    // All Users\Start Menu
    CSIDL_COMMON_PROGRAMS = 0X0017,    // All Users\Start Menu\Programs
    CSIDL_COMMON_STARTUP = 0x0018,    // All Users\Startup
    CSIDL_COMMON_DESKTOPDIRECTORY = 0x0019,    // All Users\Desktop
    CSIDL_APPDATA = 0x001a,    // <user name>\Application Data
    CSIDL_PRINTHOOD = 0x001b,    // <user name>\PrintHood

    CSIDL_LOCAL_APPDATA = 0x001c,    // <user name>\Local Settings\Applicaiton Data (non roaming)

    CSIDL_ALTSTARTUP = 0x001d,    // non localized startup
    CSIDL_COMMON_ALTSTARTUP = 0x001e,    // non localized common startup
    CSIDL_COMMON_FAVORITES = 0x001f,

    CSIDL_INTERNET_CACHE = 0x0020,
    CSIDL_COOKIES = 0x0021,
    CSIDL_HISTORY = 0x0022,
    CSIDL_COMMON_APPDATA = 0x0023,    // All Users\Application Data
    CSIDL_WINDOWS = 0x0024,    // GetWindowsDirectory()
    CSIDL_SYSTEM = 0x0025,    // GetSystemDirectory()
    CSIDL_PROGRAM_FILES = 0x0026,    // C:\Program Files
    CSIDL_MYPICTURES = 0x0027,    // C:\Program Files\My Pictures

    CSIDL_PROFILE = 0x0028,    // USERPROFILE
    CSIDL_SYSTEMX86 = 0x0029,    // x86 system directory on RISC
    CSIDL_PROGRAM_FILESX86 = 0x002a,    // x86 C:\Program Files on RISC

    CSIDL_PROGRAM_FILES_COMMON = 0x002b,    // C:\Program Files\Common

    CSIDL_PROGRAM_FILES_COMMONX86 = 0x002c,    // x86 Program Files\Common on RISC
    CSIDL_COMMON_TEMPLATES = 0x002d,    // All Users\Templates

    CSIDL_COMMON_DOCUMENTS = 0x002e,    // All Users\Documents
    CSIDL_COMMON_ADMINTOOLS = 0x002f,    // All Users\Start Menu\Programs\Administrative Tools
    CSIDL_ADMINTOOLS = 0x0030,    // <user name>\Start Menu\Programs\Administrative Tools

    CSIDL_CONNECTIONS = 0x0031,    // Network and Dial-up Connections
    CSIDL_COMMON_MUSIC = 0x0035,    // All Users\My Music
    CSIDL_COMMON_PICTURES = 0x0036,    // All Users\My Pictures
    CSIDL_COMMON_VIDEO = 0x0037,    // All Users\My Video

    CSIDL_CDBURN_AREA = 0x003b    // USERPROFILE\Local Settings\Application Data\Microsoft\CD Burning
  }

  /*
  [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode)]
  public struct SHFILEINFO
  {
    public IntPtr hIcon;
    public int iIcon;
    public uint dwAttributes;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 260)]
    public string szDisplayName;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 80)]
    public string szTypeName;
  }
   */
  [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Auto)]
  public struct SHFILEINFO
  {
    public IntPtr hIcon;
    public int iIcon;
    public uint dwAttributes;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 260)]
    public string szDisplayName;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 80)]
    public string szTypeName;
  }
  /// <summary>
  ///  managed equivalent of IShellFolder interface
  ///  Pinvoke.net / Mod by Arik Poznanski - pooya parsa
  ///  Msdn:      http://msdn.microsoft.com/en-us/library/windows/desktop/bb775075(v=vs.85).aspx
  ///  Pinvoke:   http://pinvoke.net/default.aspx/Interfaces/IShellFolder.html
```

```
/// </summary>
[ComImport]
[InterfaceType(ComInterfaceType.InterfaceIsIUnknown)]
[Guid("000214E6-0000-0000-C000-000000000046")]
public interface IShellFolder
{
  /// <summary>
  /// Translates a file object's or folder's display name into an item identifier list.
  /// Return value: error code, if any
  /// </summary>
  /// <param name="hwnd">Optional window handle</param>
  /// <param name="pbc">Optional bind context that controls the parsing operation. This parameter is normally set to NULL. </param
  /// <param name="pszDisplayName">Null-terminated UNICODE string with the display name</param>
  /// <param name="pchEaten">Pointer to a ULONG value that receives the number of characters of the display name that was parsed.<
  /// <param name="ppidl"> Pointer to an ITEMIDLIST pointer that receives the item identifier list for the object.</param>
  /// <param name="pdwAttributes">Optional parameter that can be used to query for file attributes.this can be values from the SFG
  //uint ParseDisplayName(IntPtr hwnd, IntPtr pbc, String pszDisplayName, out UInt32 pchEaten, out IntPtr ppidl, ref UInt32 pdwAttribut
  uint ParseDisplayName(IntPtr hwnd, IntPtr pbc, String pszDisplayName, out UInt32 pchEaten, out IntPtr ppidl, ref SFGAOF pdwAttri


  /// <summary>
  ///Allows a client to determine the contents of a folder by creating an item identifier enumeration object and returning its IEnumID
  ///Return value: error code, if any
  /// </summary>
  /// <param name="hwnd">If user input is required to perform the enumeration, this window handle should be used by the enumeration
  /// <param name="grfFlags">Flags indicating which items to include in the  enumeration. For a list of possible values, see the SH
  /// <param name="ppenumIDList">Address that receives a pointer to the IEnumIDList interface of the enumeration object created by
  uint EnumObjects(IntPtr hwnd, SHCONTF grfFlags, out IEnumIDList ppenumIDList);
  //IEnumIDList EnumObjects(IntPtr hwnd, SHCONTF grfFlags);

  /// <summary>
  ///Retrieves an IShellFolder object for a subfolder.
  // Return value: error code, if any
  /// </summary>
  /// <param name="pidl">Address of an ITEMIDLIST structure (PIDL) that identifies the subfolder.</param>
  /// <param name="pbc">Optional address of an IBindCtx interface on a bind context object to be used during this operation.</para
  /// <param name="riid">Identifier of the interface to return. </param>
  /// <param name="ppv">Address that receives the interface pointer.</param>
  uint BindToObject(IntPtr pidl, IntPtr pbc, [In]ref Guid riid, out IShellFolder ppv);

  /// <summary>
  /// Requests a pointer to an object's storage interface.
  /// Return value: error code, if any
  /// </summary>
  /// <param name="pidl">Address of an ITEMIDLIST structure that identifies the subfolder relative to its parent folder. </param>
  /// <param name="pbc">Optional address of an IBindCtx interface on a bind context object to be  used during this operation.</para
  /// <param name="riid">Interface identifier (IID) of the requested storage interface.</param>
  /// <param name="ppv"> Address that receives the interface pointer specified by riid.</param>
  uint BindToStorage(IntPtr pidl, IntPtr pbc, [In]ref Guid riid, out IntPtr ppv);

  /// <summary>
  /// Determines the relative order of two file objects or folders, given
  /// their item identifier lists. Return value: If this method is
  /// successful, the CODE field of the HRESULT contains one of the
  /// following values (the code can be retrived using the helper function
  /// GetHResultCode): Negative A negative return value indicates that the first item should precede the second (pidl1 < pidl2).
   ////
   ///Positive A positive return value indicates that the first item should
   ///follow the second (pidl1 > pidl2).  Zero A return value of zero
  ///indicates that the two items are the same (pidl1 = pidl2).
  /// </summary>
  /// <param name="lParam">Value that specifies how the comparison  should be performed. The lower Sixteen bits of lParam define th
  ///  The upper sixteen bits of lParam are used for flags that modify the sorting rule. values can be from  the SHCIDS enum
  /// </param>
  /// <param name="pidl1">Pointer to the first item's ITEMIDLIST structure.</param>
  /// <param name="pidl2"> Pointer to the second item's ITEMIDLIST structure.</param>
  /// <returns></returns>
  [PreserveSig]
  Int32 CompareIDs(Int32 lParam, IntPtr pidl1, IntPtr pidl2);

  /// <summary>
  /// Requests an object that can be used to obtain information from or interact
  /// with a folder object.
  /// Return value: error code, if any
  /// </summary>
  /// <param name="hwndOwner">Handle to the owner window.</param>
  /// <param name="riid">Identifier of the requested interface.</param>
  /// <param name="ppv">Address of a pointer to the requested interface. </param>
  uint CreateViewObject(IntPtr hwndOwner, [In] ref Guid riid, out IntPtr ppv);

  /// <summary>
  /// Retrieves the attributes of one or more file objects or subfolders.
  /// Return value: error code, if any
  /// </summary>
  /// <param name="cidl">Number of file objects from which to retrieve attributes. </param>
  /// <param name="apidl">Address of an array of pointers to ITEMIDLIST structures, each of which  uniquely identifies a file objec
  /// <param name="rgfInOut">Address of a single ULONG value that, on entry contains the attributes that the caller is
  /// requesting. On exit, this value contains the requested attributes that are common to all of the specified objects. this value ca
  /// </param>
  uint GetAttributesOf(UInt32 cidl, [MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 0)]IntPtr[] apidl, ref SFGAOF rgfInOut);

  /// <summary>
  /// Retrieves an OLE interface that can be used to carry out actions on the
  /// specified file objects or folders. Return value: error code, if any
  /// </summary>
  /// <param name="hwndOwner">Handle to the owner window that the client should specify if it displays a dialog box or message box
  /// <param name="cidl">Number of file objects or subfolders specified in the apidl parameter. </param>
  /// <param name="apidl">Address of an array of pointers to ITEMIDLIST  structures, each of which  uniquely identifies a file obje
  /// <param name="riid">Identifier of the COM interface object to return.</param>
  /// <param name="rgfReserved"> Reserved. </param>
  /// <param name="ppv">Pointer to the requested interface.</param>
  uint GetUIObjectOf(IntPtr hwndOwner, UInt32 cidl, [MarshalAs(UnmanagedType.LPArray, SizeParamIndex = 1)]IntPtr[] apidl, [In] ref G

  /// <summary>
  /// Retrieves the display name for the specified file object or subfolder.
  /// Return value: error code, if any
  /// </summary>
  /// <param name="pidl">Address of an ITEMIDLIST structure (PIDL)  that uniquely identifies the file  object or subfolder relative
  /// <param name="uFlags">Flags used to request the type of display name to return. For a list of possible values. </param>
  /// <param name="pName"> Address of a STRRET structure in which to return the display name.</param>
  uint GetDisplayNameOf(IntPtr pidl, SHGDN uFlags, out STRRET pName);
```

```
    /// <summary>
    /// Sets the display name of a file object or subfolder, changing the item
    /// identifier in the process.
    /// Return value: error code, if any
    /// </summary>
    /// <param name="hwnd"> Handle to the owner window of any dialog or message boxes that the client displays.</param>
    /// <param name="pidl"> Pointer to an ITEMIDLIST structure that uniquely identifies the file object or subfolder relative to the
    /// <param name="pszName"> Pointer to a null-terminated string that specifies the new display name.</param>
    /// <param name="uFlags">Flags indicating the type of name specified by  the lpszName parameter. For a list of possible values, s
    /// <param name="ppidlOut"></param>
    uint SetNameOf(IntPtr hwnd, IntPtr pidl, String pszName, SHGDN uFlags, out IntPtr ppidlOut);
  }


  [ComImport]
  [InterfaceType(ComInterfaceType.InterfaceIsIUnknown)]
  [Guid("000214F2-0000-0000-C000-000000000046")]
  public interface IEnumIDList
  {

    // Retrieves the specified number of item identifiers in the enumeration sequence and advances the current position by the number of
    /*
    [PreserveSig()]
    uint Next(
        uint celt,                // Number of elements in the array pointed to by the rgelt parameter.
        out IntPtr rgelt,         // Address of an array of ITEMIDLIST pointers that receives the item identifiers. The implementation mu
      // The calling application is responsible for freeing the item identifiers using the Shell's allocator.
        out Int32 pceltFetched    // Address of a value that receives a count of the item identifiers actually returned in rgelt. The cou
        );
    */
    [PreserveSig()]
    uint Next(
        uint celt,                // Number of elements in the array pointed to by the rgelt parameter.
        [In(), Out(), MarshalAs(UnmanagedType.LPArray)] IntPtr[] rgelt,         // Address of an array of ITEMIDLIST pointers that rece
      // The calling application is responsible for freeing the item identifiers using the Shell's allocator.
        out Int32 pceltFetched    // Address of a value that receives a count of the item identifiers actually returned in rgelt. The co
        );


    // Skips over the specified number of elements in the enumeration sequence.
    [PreserveSig()]
    uint Skip(
        uint celt                 // Number of item identifiers to skip.
        );

    // Returns to the beginning of the enumeration sequence.
    [PreserveSig()]
    uint Reset();

    // Creates a new item enumeration object with the same contents and state as the current one.
    [PreserveSig()]
    uint Clone(
        out IEnumIDList ppenum    // Address of a pointer to the new enumeration object. The calling application must eventually free t
        );

    /*
    [PreserveSig()]
    uint Next(
      uint celt,
      [In(), Out(), MarshalAs(UnmanagedType.LPArray)] IntPtr[] rgelt,
      out uint pceltFetched);

    IEnumIDList Clone();
    */
  }


  /// <summary>
  /// A standard OLE enumerator used by a client to determine the available search objects for a folder.
  /// </summary>
  [ComImport, InterfaceType(ComInterfaceType.InterfaceIsIUnknown)]
  [Guid("0E700BE1-9DB6-11d1-A1CE-00C04FD75D13")]
  public interface IEnumExtraSearch
  {
    /// <summary>
    /// Used to request information on one or more search objects.
    /// </summary>
    /// <param name="celt">The number of search objects to be enumerated, starting from the current object. If celt is too large, the
    /// <param name="rgelt">A pointer to an array of pceltFetched EXTRASEARCH structures containing information on the enumerated obj
    /// <param name="pceltFetched">The number of objects actually enumerated. This may be less than celt.</param>
    /// <returns>Returns S_OK if successful, or a COM-defined error code otherwise.</returns>
    [PreserveSig]
    int Next(uint celt, [MarshalAs(UnmanagedType.LPArray)] EXTRASEARCH[] rgelt, out uint pceltFetched);

    /// <summary>
    /// Skip a specified number of objects.
    /// </summary>
    /// <param name="celt">The number of objects to skip.</param>
    /// <returns>Returns S_OK if successful, or a COM-defined error code otherwise.</returns>
    [PreserveSig]
    int Skip(uint celt);

    /// <summary>
    /// Used to reset the enumeration index to zero.
    /// </summary>
    /// <returns>Returns S_OK if successful, or a COM-defined error code otherwise.</returns>
    [PreserveSig]
    int Reset();

    /// <summary>
    /// Used to request a duplicate of the enumerator object to preserve its current state.
    /// </summary>
    /// <param name="ppenum">A pointer to the IEnumExtraSearch interface of a new enumerator object.</param>
    /// <returns>Returns S_OK if successful, or a COM-defined error code otherwise.</returns>
    [PreserveSig]
    int Clone(out IEnumExtraSearch ppenum);
  }

  /// <summary>
  /// Used by an IEnumExtraSearch enumerator object to return information on the search objects supported by a Shell Folder object.
```

```
    /// </summary>
    [StructLayout(LayoutKind.Sequential)]
    public struct EXTRASEARCH
    {
      /// <summary>
      /// A search object's GUID.
      /// </summary>
      public Guid guidSearch;

      /// <summary>
      /// A Unicode string containing the search object's friendly name. It will be used to identify the search engine on the Search Assis
      /// </summary>
      [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 80)]
      public string wszFriendlyName;

      /// <summary>
      /// The URL that will be displayed in the search pane.
      /// </summary>
      [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 2048)]
      public string wszUrl;
    }
  }
}
```

登録日 :2015-06-17　最終更新日 :2015-09-28

この記事に関連するページ

- [C#] サイズの大きいZIPファイルの属性をIShellFolder.GetAttributesOf やSHGetFileInfo で取得すると時間がかかる
- [C#] 完全PIDLを使用して IShellFolder.GetAttributesOf で属性を素得すると正しい属性値が取得できない
- [C#] IEnumIDList.Next メソッドで取得した pidl を保持する - IEnumIDList.Next メソッドで取得した pidlを保持して利用するとメモリアクセスエラーになる
- [C#] ファイルパスから PIDL(シェルネームスペースITEM ID)を取得する
- [C#] IShellFolder.SetNameOf を利用してファイル名・オブジェクト名を変更する

このページのタグ:[C#] [シェルネームスペース]