WinInteropUtils

WinInteropUtils is a C# library written in .NET 8 that provides managed P/Invoke wrappers for tons of functions you can use, from reading file attributes to interacting with COM interfaces.

The base namespace is `FireBlade.WinInteropUtils`.

> **WARNING**
>
> WinInteropUtils is still in beta, so bugs may occur. If you experience any bugs, please report them on the Issues page.

Features

- WindowsFile file info wrapper
- Full enums (1600+ values) for HRESULTs and Win32 error codes
- Dialog boxes
- Useful utility User32 functions (SendMessage, SetWindowLongPtr...)
- Constants and Macros
- COM interfaces
- Icon management (DestroyIcon, ...)

Getting Started

WinInteropUtils provides many wrappers for P/Invoke functions, from simple functions to COM interfaces. This guide will show you how to install and use this library.

> **WARNING**
>
> WinInteropUtils is still in beta, so bugs may occur. If you experience any bugs, please report them on the Issues page.

Usage Examples

Reading file attributes

This example reads the attributes of a file named `C:\test.txt` and checks if the file is hidden.

```
// Get the WindowsFile
var winFile = Shell32.GetFileInfo(@"C:\test.txt");

if (winFile != null)
{
    // Always make sure to dispose WindowsFiles when you're done using them
    // to avoid leaks on GDI handles!
    using WindowsFile file = winFile;

    // Check the attributes
    Console.WriteLine(file.Attributes.HasFlag(WindowsFileAttributes.Hidden)
    ? "File is hidden!"
     : "File is visible...");
}
else
{
    Console.WriteLine("File not found or read error");
}
```

Showing dialog boxes

The following examples show how to display common dialog boxes.

```
// ShellAbout dialog box (for an example run the winver command)
// icon: either a System.Drawing.Icon or nint GDI icon handle

Shell32.ShellAbout(hWnd, "My app name", "Some additional info", myAppIcon);
```

```
Shell32.ShellAbout("My app name", "Some additional info"); // without an icon and hwnd

// Icon picker dialog (PickIconDlg)

Shell32.ShowPickIconDialog(@"%SYSTEMROOT%\System32\shell32.dll", 16, out Icon ic);
Shell32.ShowPickIconDialog(@"%SYSTEMROOT%\System32\imageres.dll", 48, out string path, out
int idx);

// File properties dialog (doesn't support hwnd because that's how Windows works)

Shell32.ShowFileProperties(@"C:\test.txt");
```

Using COM interfaces

This example shows how to use COM interfaces.

> **TIP**
>
> For more info, see Using COM.

```
using FireBlade.WinInteropUtils.ComponentObjectModel; // make sure to add this for COM!

HRESULT hr = COM.Initialize(CoInit.ApartmentThreaded);

if (Macros.Succeeded(hr))
{
    hr = COM.CreateInstance<IFileOpenDialog>(
        new Guid("84bccd23-5fde-4cdb-aea4-af64b83d78ab"),
        null,
        CreateInstanceContext.InprocServer,
        new Guid("84bccd23-5fde-4cdb-aea4-af64b83d78ab"),
        out IFileOpenDialog dlg);

    if (Macros.Succeeded(hr))
    {
        dlg.SetTitle("My File Dialog - Open a File");

        hr = dlg.Show(hWnd);

        if (Macros.Succeeded(hr))
        {
            Console.WriteLine("Dialog accepted!");
        }
```

```
    }

    COM.Uninitialize();
}
```

API

For more documentation, visit the API page.

Using COM

COM (Component-Object Model) is a Windows framework that allows for reusable components in your code. This guide will go over how to use COM in WinInteropUtils.

> **WARNING**
>
> **COM** is for advanced users, so there's manual memory management and other advanced things. Wrappers for COM interfaces will come in future releases, but for now this is the only option.

Basic Setup

To begin using COM, you need to import the appropirate COM namespaces into your script:

```
using FireBlade.WinInteropUtils.ComponentObjectModel; // for base COM class and additional future COM non-interface wrappers
using FireBlade.WinInteropUtils.ComponentObjectModel.Interfaces; // for COM interfaces

using FireBlade.WinInteropUtils; // base WinInteropUtils namespace, required for Macros (useful for COM because we need to check a lot of HRESULTs)
```

Now we're ready to use COM. Over the course of this guide we will build up a sample of a file dialog. Before using COM interfaces, we need to initialize COM:

```
// if you're using v0.11 or earlier the function is named COM.CoInitialize instead
// if you're using v0.2 or earlier the enum is named CoInit instead (don't blame me for the compatibility and migration issues; that's what you get for using a library in beta)
HRESULT hr = COM.Initialize(COMInitOptions.ApartmentThreaded); // or MultiThreaded if inside a non-GUI/non-STA app
```

Now, after initializng COM, we need to check if the initialization succeeded:

```
if (Macros.Succeeded(hr))
{
    // init succeeded: proceeded with COM

    // our COM code...
}
```

Creating and Interacting with a COM instance

Now that COM is initialized, we need to actually create the COM instance, and once again check for success:

```
hr = COM.CreateInstance<IFileOpenDialog>(
    new Guid("DC1C5A9C-E88A-4dde-A5A1-60F82A20AEF7"),
    null,
    COM.CreateInstanceContext.InprocServer,
    new Guid("d57c7288-d4ad-4768-be02-9d969532d960"),
    out IFileOpenDialog dlg);

if (Macros.Succeeded(hr))
{
    // creation succeeded: proceed with interaction
}
```

And we can now proceed with interacting with the interface, which is a lot easier. In the following example, we configure the file dialog with a custom title and 2 filters:

> **NOTE**
> This code defines the `COMDLG_FILTERSPEC` struct itself, because WinInteropUtils doesn't support it yet.

```
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode)]
public struct COMDLG_FILTERSPEC
{
    public string pszName;
    public string pszSpec;
}
```

```
COMDLG_FILTERSPEC[] filters = new COMDLG_FILTERSPEC[]
{
    new COMDLG_FILTERSPEC { pszName = "Text Files", pszSpec = "*.txt" },
    new COMDLG_FILTERSPEC { pszName = "All Files", pszSpec = "*.*" }
};

// Allocate unmanaged memory (SetFileTypes requires pointer)
IntPtr ptr = Marshal.AllocHGlobal(Marshal.SizeOf<COMDLG_FILTERSPEC>() * filters.Length);

try
{
```

```
    // Copy each struct into unmanaged memory
    for (int i = 0; i < filters.Length; i++)
    {
        IntPtr structPtr = IntPtr.Add(ptr, i * Marshal.SizeOf<COMDLG_FILTERSPEC>());
        Marshal.StructureToPtr(filters[i], structPtr, false);
    }

    // ptr now points to the unmanaged array
    dlg.SetFileTypes((uint)filters.Length, ptr);
}
finally
{
    Marshal.FreeHGlobal(ptr);
}

dlg.SetTitle("Cool File Dialog");
```

And now we can actually SHOW the file dialog and read its results:

```
hr = dlg.Show(hWnd);

if (Macros.Succeeded(hr))
{
    Console.WriteLine("Dialog accepted!");

    hr = dlg.GetResult(out nint iptr);

    if (Macros.Succeeded(hr))
    {
        IShellItem item = (IShellItem)Marshal.GetTypedObjectForIUnknown(iptr,
typeof(IShellItem));

        hr = item.GetDisplayName(
            SIGDN.SIGDN_FILESYSPATH,
            out nint pptr);

        if (Macros.Succeeded(hr))
        {
            string path = Marshal.PtrToStringUni(pptr);

            Console.WriteLine("Chosen path: " + path);

            Marshal.FreeCoTaskMem(pptr);
        }
```

```
    }
}
```

Cleanup

Once you're done using COM, you need to deinitialize the COM library (if the initialization succeeded) and release any COM interfaces, using the Release method:

```
dlg.Release();
COM.Uninitialize();
```

Final Example

So now, our final example code looks like this:

```
using FireBlade.WinInteropUtils.ComponentObjectModel;
using FireBlade.WinInteropUtils.ComponentObjectModel.Interfaces;
using FireBlade.WinInteropUtils;

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode)]
public struct COMDLG_FILTERSPEC
{
    public string pszName;
    public string pszSpec;
}

HRESULT hr = COM.Initialize(COMInitOptions.ApartmentThreaded);

if (Macros.Succeeded(hr))
{
    hr = COM.CreateInstance<IFileOpenDialog>(
        new Guid("DC1C5A9C-E88A-4dde-A5A1-60F82A20AEF7"),
        null,
        COM.CreateInstanceContext.InprocServer,
        new Guid("d57c7288-d4ad-4768-be02-9d969532d960"),
        out IFileOpenDialog dlg);

    if (Macros.Succeeded(hr))
    {
        COMDLG_FILTERSPEC[] filters = new COMDLG_FILTERSPEC[]
        {
            new COMDLG_FILTERSPEC { pszName = "Text Files", pszSpec = "*.txt" },
            new COMDLG_FILTERSPEC { pszName = "All Files", pszSpec = "*.*" }
        };
```

```
        // Allocate unmanaged memory (SetFileTypes requires pointer)
        IntPtr ptr = Marshal.AllocHGlobal(Marshal.SizeOf<COMDLG_FILTERSPEC>()
* filters.Length);

        try
        {
            // Copy each struct into unmanaged memory
            for (int i = 0; i < filters.Length; i++)
            {
                IntPtr structPtr = IntPtr.Add(ptr, i * Marshal.SizeOf<COMDLG_FILTERSPEC>());
                Marshal.StructureToPtr(filters[i], structPtr, false);
            }

            // ptr now points to the unmanaged array
            dlg.SetFileTypes((uint)filters.Length, ptr);
        }
        finally
        {
            Marshal.FreeHGlobal(ptr);
        }

        dlg.SetTitle("Cool File Dialog");

        hr = dlg.Show(hWnd);

        if (Macros.Succeeded(hr))
        {
            Console.WriteLine("Dialog accepted!");

            hr = dlg.GetResult(out nint iptr);

            if (Macros.Succeeded(hr))
            {
                IShellItem item = (IShellItem)Marshal.GetTypedObjectForIUnknown(iptr,
typeof(IShellItem));

                hr = item.GetDisplayName(
                    SIGDN.SIGDN_FILESYSPATH,
                    out nint pptr);

                if (Macros.Succeeded(hr))
                {
                    string path = Marshal.PtrToStringUni(pptr);

                    Console.WriteLine("Chosen path: " + path);
```

```
                    Marshal.FreeCoTaskMem(pptr);
                }

                item.Release();
            }
        }

        dlg.Release();
    }

    COM.Uninitialize();
}
```

If we run this sample code, a file dialog should appear and, after picking a file, its path should be outputted.

Basic COM Interfaces

This article lists the basic COM interfaces.

> **NOTE**
> This article lists only COM interfaces defined by WinInteropUtils.
> For interfaces for the Common Item Dialog, see File Dialog COM Interfaces.

IUnknown

API
The base interface that all COM interfaces inherit from. This interface defines the base Release and AddRef methods.

IShellItem

API
Exposes methods that retrieve information about a Shell item, such as a file or directory. Third parties do not implement this interface; only use the implementation provided with the system.

IModalWindow

API
This interface represents a modal window, such as a file dialog. This interface only exposes the Show method.

File Dialog COM Interfaces

This article lists the COM interfaces for the Common Item Dialog, the modernized common file dialog introduced in Windows Vista.

> **NOTE**
> This article lists only COM interfaces defined by WinInteropUtils.
> For basic interfaces that are not associated with the Common Item Dialog, see Basic COM Interfaces.

IFileDialog

API
The base interface for file dialogs. This interface exposes methods that initialize, show, and get results from the common file dialog, and inherits from IModalWindow.

IFileOpenDialog

API
This interface exposes methods for the Open file dialog. The only methods exposed in this interface are GetResults and GetSelectedItems. It inherits from IFileDialog.

IFileSaveDialog

API
This interface exposes methods for the Save file dialog. It extends the IFileDialog interface by adding methods specific to the save dialog, which include those that provide support for the collection of metadata to be persisted with the file.

IFileDialogCustomize

API
This interface exposes methods that allow adding controls and customizing a file dialog.

> **WARNING**
> Controls are added to the dialog before the dialog is shown. Their layout is implied by the order in which they are added. Once the dialog is shown, controls cannot be added or removed, but the existing controls can be hidden or disabled at any time. Their labels can also be changed at any time.