Interoperability Overview (C# Programming Guide)

Visual Studio 2015 Other Versions

The topic describes methods to enable interoperability between C# managed code and unmanaged code.

Platform Invoke

Platform invoke is a service that enables managed code to call unmanaged functions that are implemented in dynamic link libraries (DLLs), such as those in the Microsoft Win32 API. It locates and invokes an exported function and marshals its arguments (integers, strings, arrays, structures, and so on) across the interoperation boundary as needed.

For more information, see Consuming Unmanaged DLL Functions and How to: Use Platform Invoke to Play a Wave File (C# Programming Guide).

Note

The Common Language Runtime (CLR) (CLR) manages access to system resources. Calling unmanaged code that is outside the CLR bypasses this security mechanism, and therefore presents a security risk. For example, unmanaged code might call resources in unmanaged code directly, bypassing CLR security mechanisms. For more information, see .NET Framework Security.

C++ Interop

You can use C++ interop, also known as It Just Works (IJW), to wrap a native C++ class so that it can be consumed by code that is authored in C# or another .NET Framework language. To do this, you write C++ code to wrap a native DLL or COM component. Unlike other .NET Framework languages, Visual C++ has interoperability support that enables managed and unmanaged code to be located in the same application and even in the same file. You then build the C++ code by using the **/clr** compiler switch to produce a managed assembly. Finally, you add a reference to the assembly in your C# project and use the wrapped objects just as you would use other managed classes.

Exposing COM Components to C#

You can consume a COM component from a C# project. The general steps are as follows:

1. Locate a COM component to use and register it. Use regsvr32.exe to register or un–register a COM DLL.
2. Add to the project a reference to the COM component or type library.

   When you add the reference, Visual Studio uses the Tlbimp.exe (Type Library Importer), which takes a type library as input, to output a .NET Framework interop assembly. The assembly, also named a runtime callable wrapper (RCW), contains managed classes and interfaces that wrap the COM classes and interfaces that are in the type library. Visual Studio adds to the project a reference to the generated assembly.
3. Create an instance of a class that is defined in the RCW. This, in turn, creates an instance of the COM object.
4. Use the object just as you use other managed objects. When the object is reclaimed by garbage collection, the instance of the COM object is also released from memory.

For more information, see Exposing COM Components to the .NET Framework.

Exposing C# to COM

COM clients can consume C# types that have been correctly exposed. The basic steps to expose C# types are as follows:

bly Information Dialog Box.

You can modify Visual C# project properties to automatically register the C# assembly for COM interop. Visual Studio uses the Regasm.exe (Assembly Registration Tool), using the **/tlb** command-line switch, which takes a managed assembly as input, to generate a type library. This type library describes the **public** types in the assembly and adds registry entries so that COM clients can create managed classes.

For more information, see Exposing .NET Framework Components to COM and Example COM Class (C# Programming Guide).

See Also

Concepts

C# Programming Guide

Other Resources

Improving Interop Performance
Introduction to COM Interop
Marshaling between Managed and Unmanaged Code
Interoperating with Unmanaged Code
Advanced COM Interoperability