C# Programming Guide

How to: Create and Use C# DLLs (C# Programming Guide)

A dynamic linking library (DLL) is linked to your program at run time. To demonstrate building and using a DLL, consider the following scenario:

- `MathLibrary.DLL`: The library file that contains the methods to be called at run time. In this example, the DLL contains two methods, `Add` and `Multiply`.
- `Add.cs`: The source file that contains the method `Add(long i, long j)`. It returns the sum of its parameters. The class `AddClass` that contains the method `Add` is a member of the namespace `UtilityMethods`.
- `Mult.cs`: The source code that contains the method `Multiply(long x, long y)`. It returns the product of its parameters. The class `MultiplyClass` that contains the method `Multiply` is also a member of the namespace `UtilityMethods`.
- `TestCode.cs`: The file that contains the `Main` method. It uses the methods in the DLL file to calculate the sum and the product of the run-time arguments.

Example

C# Copy Code

```
// File: Add.cs
namespace UtilityMethods
{
    public class AddClass
    {
        public static long Add(long i, long j)
        {
            return (i + j);
        }
    }
}
```

C# Copy Code

```
// File: Mult.cs
namespace UtilityMethods
{
    public class MultiplyClass
    {
        public static long Multiply(long x, long y)
        {
            return (x * y);
        }
    }
}
```

C# Copy Code

```
// File: TestCode.cs

using UtilityMethods;

class TestCode
{
    static void Main(string[] args)
    {
        System.Console.WriteLine("Calling methods from MathLibrary.DLL:");
```

```
        if (args.Length != 2)
        {
            System.Console.WriteLine("Usage: TestCode <num1> <num2>");
            return;
        }

        long num1 = long.Parse(args[0]);
        long num2 = long.Parse(args[1]);

        long sum = AddClass.Add(num1, num2);
        long product = MultiplyClass.Multiply(num1, num2);

        System.Console.WriteLine("{0} + {1} = {2}", num1, num2, sum);
        System.Console.WriteLine("{0} * {1} = {2}", num1, num2, product);
    }
}
/* Output (assuming 1234 and 5678 are entered as command line args):
    Calling methods from MathLibrary.DLL:
    1234 + 5678 = 6912
    1234 * 5678 = 7006652
*/
```

This file contains the algorithm that uses the DLL methods, `Add` and `Multiply`. It starts with parsing the arguments entered from the command line, `num1` and `num2`. Then it calculates the sum by using the `Add` method on the `AddClass` class, and the product by using the `Multiply` method on the `MultiplyClass` class.

Notice that the using directive at the beginning of the file enables you to use the unqualified class names to reference the DLL methods at compile time, as follows:

C# Copy Code

```
MultiplyClass.Multiply(num1, num2);
```

Otherwise, you have to use the fully qualified names, as follows:

C# Copy Code

```
UtilityMethods.MultiplyClass.Multiply(num1, num2);
```

Execution

To run the program, enter the name of the EXE file, followed by two numbers, as follows:

```
TestCode 1234 5678
```

Compiling the Code

To build the file `MathLibrary.DLL`, compile the two files `Add.cs` and `Mult.cs` using the following command line:

```
csc /target:library /out:MathLibrary.DLL Add.cs Mult.cs
```

The /target:library [http://msdn.microsoft.com/en-us/library/e13syb43.aspx] compiler option tells the compiler to output a DLL instead of an EXE file. The /out [http://msdn.microsoft.com/en-us/library/bw3t50f3.aspx] compiler option followed by a file name is used to specify the DLL file name. Otherwise, the compiler uses the first file (`Add.cs`) as the name of the DLL.

To build the executable file, `TestCode.exe`, use the following command line:

```
csc /out:TestCode.exe /reference:MathLibrary.DLL TestCode.cs
```

The **/out** compiler option tells the compiler to output an EXE file and specifies the name of the output file (`TestCode.exe`). This compiler option is optional. The /reference [http://msdn.microsoft.com/en-us/library/yabyz3h4.aspx] compiler option specifies the DLL file or files that this program uses.

See Also

Tasks

How to: Specify a Base Address for a DLL [http://msdn.microsoft.com/en-us/library/w4w1x6a7.aspx]

Concepts

C# Programming Guide [http://msdn.microsoft.com/en-us/library/67ef8sbd.aspx]
Creating a Class to Hold DLL Functions [http://msdn.microsoft.com/en-us/library/khbsw73t.aspx]

Tags: c# dlls

Community Content