Contents

C# Compiler Errors

1/23/2019 • 2 minutes to read • Edit Online

Some C# compiler errors have corresponding topics that explain why the error is generated, and, in some cases, how to fix the error. Use one of the following steps to see whether help is available for a particular error message.

- Find the error number (for example, CS0029) in the Output Window, and then search for it on Microsoft Docs.
- Choose the error number (for example, CS0029) in the Output Window, and then choose the F1 key.
- In the Index, enter the error number in the **Look for** box.

If none of these steps leads to information about your error, go to the end of this page, and send feedback that includes the number or text of the error.

For information about how to configure error and warning options in C#, see Build Page, Project Designer (C#).

> **NOTE**
> Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. For more information, see Personalizing the IDE.

See also

- C# Compiler Options
- Sorry, we don't have specifics on this C# error
- Build Page, Project Designer (C#)
- /warn (C# Compiler Options)
- /nowarn (C# Compiler Options)

Compiler Error CS0001

5/4/2018 • 2 minutes to read • Edit Online

Internal compiler error

Try to determine whether the compiler is failing because of its inability to parse unexpected syntax. If you receive this error repeatedly, please contact Microsoft.

Compiler Error CS0006

5/4/2018 • 2 minutes to read • Edit Online

Metadata file 'dll_name' could not be found

The program was compiled and explicitly passed the name of a file that contained metadata; however, the .dll was not found. For more information, see /reference (C# Compiler Options).

Compiler Error CS0007

5/4/2018 • 2 minutes to read • Edit Online

Unexpected common language runtime initialization error — 'description'

This error occurs if the runtime could not be loaded. This could occur if the version of the common language runtime that the compiler attempts to load is not present on the machine, or if the common language runtime installation or configuration is corrupt.

This can happen if the `csc.exe.config` file was changed. This file is configured during setup and should not be changed. If there is a possibility that the `csc.exe.config` file was changed, check the file to make sure that the version of the runtime specified in the file is present on the machine. If the correct version is present, it may be corrupted. Reinstall the common language runtime.

Compiler Error CS0015

5/4/2018 • 2 minutes to read • Edit Online

The name of type 'type' is too long

The fully qualified name of a user-defined type must have fewer than 1024 characters, including the periods.

The following sample generates CS0015:

```
// CS0015.cs

// Remove a C from one of the namespace names or the class name to resolve the error by
// reducing the number of characters in the qualified class name to fewer than 1024.

namespace CC
{
    // The following namespace name has 510 characters.
    namespace
CCC510charsCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
    {
        // The following class name has 510 characters. The qualified class name,
        // CC.CCC510chars…C.CCCC510chars…C (namespace.namespace.class), has 1024
        // characters, which causes compiler error CS0015.
        public class
CCCC510charsCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
        {
            public static void Main()
            {
            }
        }
    }
}
```

Compiler Error CS0016

10/16/2018 • 2 minutes to read • Edit Online

Could not write to output file 'file' — 'reason'

The compiler could not write to an output file. Check the path to the file to make sure it exists. If a previously built file is already at the location, make sure it is writeable, and that no process currently has a lock on the file. For example, make sure your executable is not running when you attempt to build.

Compiler Error CS0019

1/23/2019 • 2 minutes to read • Edit Online

Operator 'operator' cannot be applied to operands of type 'type' and 'type'

A binary operator is applied to data types that do not support it. For example, you cannot use the || operator on strings, you cannot use + , -, < , or > operators on bool variables, and you cannot use the == operator with a `struct` type unless the type explicitly overloads that operator.

If you encounter this error with a class type, it is because the class does not overload the operator. For more information, see the operator keyword article and Overloadable Operators.

Example

In the following example, CS0019 is generated in two places because bool in C# is not convertible to int. CS0019 also is generated when the subtraction operator is applied to a string. The addition operator (+) can be used with string operands because that operator is overloaded by the `String` class to perform string concatenation.

```
static void Main()
{
    bool result = true;
    if (result > 0) //CS0019
    {
        // Do something.
    }

    int i = 1;
    // You cannot compare an integer and a boolean value.
    if (i == true) //CS0019
    {
        //Do something...
    }

    // The following use of == causes no error. It is the comparison of
    // an integer and a boolean value that causes the error in the
    // previous if statement.
    if (result == true)
    {
        //Do something...
    }

    string s = "Just try to subtract me.";
    float f = 100 - s; // CS0019
}
```

Example

In the following example, conditional logic must be specified outside the ConditionalAttribute. You can pass only one predefined symbol to the ConditionalAttribute.

The following sample generates CS0019.

```
// CS0019_a.cs
// compile with: /target:library
using System.Diagnostics;
public class MyClass
{
   [ConditionalAttribute("DEBUG" || "TRACE")]   // CS0019
   public void TestMethod() {}

   // OK
   [ConditionalAttribute("DEBUG"), ConditionalAttribute("TRACE")]
   public void TestMethod2() {}
}
```

See also

- Operators
- Implicit Numeric Conversions Table

Compiler Error CS0029

1/23/2019 • 3 minutes to read • Edit Online

Cannot implicitly convert type 'type' to 'type'

The compiler requires an explicit conversion. For example, you may need to cast an r-value to be the same type as an l-value. Or, you must provide conversion routines to support certain operator overloads.

Conversions must occur when assigning a variable of one type to a variable of a different type. When making an assignment between variables of different types, the compiler must convert the type on the right-hand side of the assignment operator to the type on the left-hand side of the assignment operator. Take the following the code:

```
int i = 50;
long lng = 100;
i = lng;
```

`i = lng;` makes an assignment, but the data types of the variables on the left and right-hand side of the assignment operator don't match. Before making the assignment the compiler is implicitly converting the variable `lng`, which is of type long, to an int. This is implicit because no code explicitly instructed the compiler to perform this conversion. The problem with this code is that this is considered a narrowing conversion, and the compiler does not allow implicit narrowing conversions because there could be a potential loss of data.

A narrowing conversion exists when converting to a data type that occupies less storage space in memory than the data type we are converting from. For example, converting a long to an int would be considered a narrowing conversion. A long occupies 8 bytes of memory while an int occupies 4 bytes. To see how data loss can occur, consider the following sample:

```
int i = 50;
long lng = 3147483647;
i = lng;
```

The variable `lng` now contains a value that cannot be stored in the variable `i` because it is too large. If we were to convert this value to an int type we would be losing some of our data and the converted value would not be the same as the value before the conversion.

A widening conversion would be the opposite of a narrowing conversion. With widening conversions, we are converting to a data type that occupies more storage space in memory than the data type we are converting from. Here is an example of a widening conversion:

```
int i = 50;
long lng = 100;
lng = i;
```

Notice the difference between this code sample and the first. This time the variable `lng` is on the left-hand side of the assignment operator, so it is the target of our assignment. Before the assignment can be made, the compiler must implicitly convert the variable `i`, which is of type int, to type long. This is a widening conversion since we are converting from a type that occupies 4 bytes of memory (an int) to a type that occupies 8 bytes of memory (a long). Implicit widening conversions are allowed because there is no potential loss of data. Any value that can be stored in an int can also be stored in a long.

We know that implicit narrowing conversions are not allowed, so to be able to compile this code we need to

explicitly convert the data type. Explicit conversions are done using casting. Casting is the term used in C# to describe converting one data type to another. To get the code to compile we would need to use the following syntax:

```
int i = 50;
long lng = 100;
i = (int) lng;   // cast to int
```

The third line of code tells the compiler to explicitly convert the variable `lng`, which is of type long, to an int before making the assignment. Remember that with a narrowing conversion, there is a potential loss of data. Narrowing conversions should be used with caution and even though the code will compile you may get unexpected results at run-time.

This discussion has only been for value types. When working with value types you work directly with the data stored in the variable. However, the .NET Framework also has reference types. When working with reference types you are working with a reference to a variable, not the actual data. Examples of reference types would be classes, interfaces and arrays. You cannot implicitly or explicitly convert one reference type to another unless the compiler allows the specific conversion or the appropriate conversion operators are implemented.

The following sample generates CS0029:

```
// CS0029.cs
public class MyInt
{
    private int x = 0;

    // Uncomment this conversion routine to resolve CS0029
    /*
    public static implicit operator int(MyInt i)
    {
       return i.x;
    }
    */

    public static void Main()
   {
      MyInt myInt = new MyInt();
      int i = myInt; // CS0029
   }
}
```

See also

- Conversion Operators

Compiler Error CS0034

5/4/2018 • 2 minutes to read • Edit Online

Operator 'operator' is ambiguous on operands of type 'type1' and 'type2'

An operator was used on two objects and the compiler found more than one conversion. Because conversions have to be unique, you either have to make a cast or to make one of the conversions explicit. For more information, see Conversion Operators.

The following sample generates CS0034:

```
// CS0034.cs
public class A
{
   // allows for the conversion of A object to int
   public static implicit operator int (A s)
   {
      return 0;
   }

   public static implicit operator string (A i)
   {
      return null;
   }
}

public class B
{
   public static implicit operator int (B s)
   // one way to resolve this CS0034 is to make one conversion explicit
   // public static explicit operator int (B s)
   {
      return 0;
   }

   public static implicit operator string (B i)
   {
      return null;
   }

   public static implicit operator B (string i)
   {
      return null;
   }

   public static implicit operator B (int i)
   {
      return null;
   }
}

public class C
{
   public static void Main ()
   {
      A a = new A();
      B b = new B();
      b = b + a;   // CS0034
      // another way to resolve this CS0034 is to make a cast
      // b = b + (int)a;
   }
}
```

In C# 1.1 a compiler bug made it possible to define a class that has implicit user-defined conversions to both `int` and `bool`, and to use a bitwise `AND` operator (`&`) on objects of that type. In C# 2.0, this bug was fixed to bring the compiler into compliance with the C# specification, and therefore the following code will now cause CS0034:

```
namespace CS0034
{
    class TestClass2
    {
        public void Test()
        {
            TestClass o1 = new TestClass();
            TestClass o2 = new TestClass();
            TestClass o3 = o1 & o2; //CS0034
        }
    }

    class TestClass
    {
        public static implicit operator int(TestClass o)
        {
            return 1;
        }

        public static implicit operator TestClass(int v)
        {
            return new TestClass();
        }

        public static implicit operator bool(TestClass o)
        {
            return true;
        }
    }

}
```

Compiler Error CS0038

5/4/2018 • 2 minutes to read • Edit Online

Cannot access a nonstatic member of outer type 'type1' via nested type 'type2'

A field in a class is not automatically available to a nested class. To be available to a nested class, the field must be static. Otherwise, you must create an instance of the outer class. For more information, see Nested Types.

The following sample generates CS0038:

```
// CS0038.cs
class OuterClass
{
   public int count;
   // try the following line instead
   // public static int count;

   class InnerClass
   {
      void func()
      {
         // or, create an instance
         // OuterClass class_inst = new OuterClass();
         // int count2 = class_inst.count;
         int count2 = count;   // CS0038
      }
   }

   public static void Main()
   {
   }
}
```

Compiler Error CS0039

5/4/2018 • 2 minutes to read • Edit Online

Cannot convert type 'type1' to 'type2' via a reference conversion, boxing conversion, unboxing conversion, wrapping conversion, or null type conversion

A conversion with the as operator is allowed by inheritance, reference conversions, and boxing conversions. For more information, see Conversion Operators.

Example

The following example generates CS0039.

```
// CS0039.cs
using System;
class A
{
}
class B: A
{
}
class C: A
{
}
class M
{
    static void Main()
    {
        A a = new C();
        B b = new B();
        C c;

        // This is valid; there is a built-in reference
        // conversion from A to C.
        c = a as C;

        //The following generates CS0039; there is no
        // built-in reference conversion from B to C.
        c = b as C;  // CS0039
    }
}
```

Compiler Error CS0050

10/16/2018 • 2 minutes to read • Edit Online

Inconsistent accessibility: return type 'type' is less accessible than method 'method'

The return type and each of the types referenced in the formal parameter list of a method must be at least as accessible as the method itself. For more information, see Access Modifiers.

Example

The following sample generates CS0050 because no accessibility modifier is supplied for `MyClass` and its accessibility therefore defaults to `private`.

```
// CS0050.cs
class MyClass //accessibility defaults to private
// try the following line instead
// public class MyClass
{
}

public class MyClass2
{
    public static MyClass MyMethod()   // CS0050
    {
        return new MyClass();
    }

    public static void Main() { }
}
```

Compiler Error CS0051

5/4/2018 • 2 minutes to read • Edit Online

Inconsistent accessibility: parameter type 'type' is less accessible than method 'method'

The return type and each of the types referenced in the formal parameter list of a method must be at least as accessible as the method itself. Make sure the types used in method signatures are not accidentally private due to the omission of the `public` modifier. For more information, see Access Modifiers.

Example

The following sample generates CS0051:

```
// CS0051.cs
public class A
{
    // Try making B public since F is public
    // B is implicitly private here
    class B
    {
    }

    public static void F(B b)  // CS0051
    {
    }

    public static void Main()
    {
    }
}
```

Compiler Error CS0052

1/23/2019 • 2 minutes to read • Edit Online

Inconsistent accessibility: field type 'type' is less accessible than field 'field'

The type of a field cannot be less accessible than the field itself because all public constructs must return a publicly accessible object.

Example

The following sample generates CS0052:

```
// CS0052.cs
public class MyClass2
{
    // The following line causes an error because the field, M, is declared
    // as public, but the type, MyClass, is private. Therefore the type is
    // less accessible than the field.
    public MyClass M;   // CS0052

    private class MyClass
    {
    }
    // One way to resolve the error is to change the accessibility of the type
    // to public.
    //public class MyClass
    // Another solution is to change the accessibility of the field to private.
    // private MyClass M;
}

    public class MainClass
    {
        public static void Main()
        {
        }
    }
```

See also

- C# Keywords
- Access Modifiers
- Accessibility Levels
- Modifiers

Compiler Error CS0071

5/4/2018 • 2 minutes to read • Edit Online

An explicit interface implementation of an event must use event accessor syntax

When explicitly implementing an event that was declared in an interface, you must use manually provide the `add` and `remove` event accessors that are typically provided by the compiler. The accessor code can connect the interface event to another event in your class (shown later in this topic), or to its own delegate type. For more information, see How to: Implement Interface Events.

Example

The following sample generates CS0071.

```
// CS0071.cs
public delegate void MyEvent(object sender);

interface ITest
{
    event MyEvent Clicked;
}

class Test : Itest
{
    event MyEvent ITest.Clicked;  // CS0071

    // try the following code instead
/*
private MyEvent clicked;

    event MyEvent Itest.Clicked
    {
        add
        {
            clicked += value;
        }
        remove
        {
            clicked -= value;
        }
    }
*/
    public static void Main() { }
}
```

Compiler Error CS0103

5/4/2018 • 2 minutes to read • Edit Online

The name 'identifier' does not exist in the current context

An attempt was made to use a name that does not exist in the class, namespace, or scope. Check the spelling of the name and check your using directives and assembly references to make sure that the name that you are trying to use is available.

This error frequently occurs if you declare a variable in a loop or a `try` or `if` block and then attempt to access it from an enclosing code block or a separate code block, as shown in the following example.

```
using System;

class MyClass1
{
    public static void Main()
    {
        try
        {
            // The following declaration is only available inside the try block.
            MyClass1 conn = new MyClass1();
        }
        catch (Exception e)
        {
            // The following expression causes error CS0103, because variable
            // conn only exists in the try block.
            if (conn != null)
                Console.WriteLine("{0}", e);
        }
    }
}
```

The following example resolves the error.

```
using System;

class MyClass2
{
    public static void Main()
    {
        // To resolve the error in the example, the first step is to
        // move the declaration of conn out of the try block. The following
        // declaration is available throughout the Main method.
        MyClass2 conn = null;
        try
        {
            // Inside the try block, use the conn variable that you declared
            // previously.
            conn = new MyClass2();
        }
        catch (Exception e)
        {
            // The following expression no longer causes an error, because
            // the declaration of conn is in scope.
            if (conn != null)
                Console.WriteLine("{0}", e);
        }
    }
}
```

Compiler Error CS0106

10/3/2018 • 2 minutes to read • Edit Online

The modifier 'modifier' is not valid for this item

A class or interface member was marked with an invalid access modifier. The following examples describe some of these invalid modifiers:

- The static and public modifiers are not permitted on interface methods.
- The static modifier is not permitted on a local function.
- The `public` keyword is not allowed on an explicit interface declaration. In this case, remove the `public` keyword from the explicit interface declaration.
- The abstract keyword is not allowed on an explicit interface declaration because an explicit interface implementation can never be overridden.
- Access modifiers are not allowed on a local function. Local functions are always private.

In prior releases of Visual Studio, the `static` modifier was not permitted on a class, but `static` classes are allowed starting with Visual Studio 2005.

For more information, see Interfaces

Example

The following sample generates CS0106.

```
// CS0106.cs
namespace MyNamespace
{
   interface I
   {
      void m();
      static public void f();   // CS0106
   }

   public class MyClass
   {
      public void I.m() {}   // CS0106
      public static void Main() {}
   }
}
```

Compiler Error CS0115

5/4/2018 • 2 minutes to read • Edit Online

'function' : no suitable method found to override

A method was marked as an override, but the compiler found no method to override. For more information, see override, and Knowing When to Use Override and New Keywords.

Example

The following sample generates CS0115. You can resolve CS0115 in one of two ways:

- Remove the `override` keyword from the method in `MyClass2`.
- Use `MyClass1` as a base class for `MyClass2`.

```
// CS0115.cs
namespace MyNamespace
{
    abstract public class MyClass1
    {
        public abstract int f();
    }

    abstract public class MyClass2
    {
        public override int f()   // CS0115
        {
            return 0;
        }

        public static void Main()
        {
        }
    }
}
```

Compiler Error CS0116

1/23/2019 • 2 minutes to read • Edit Online

A namespace cannot directly contain members such as fields or methods.

A namespace can contain other namespaces, structs, and classes. For more information, see the namespace keyword article.

Example

The following sample will cause Visual Studio to flag parts of the code as being in violation of CS0116. Attempting to build this code will result in build failure:

```
// CS0116.cs
namespace x
{
    // A namespace can be placed within another namespace
    using System;

    // These variables trigger the CS0116 error as they are declared outside of a struct or class
    public int latitude;
    public int longitude;
    Coordinate coord;

    // Autoproperties also fall under the definition of this rule
    public string LocationName { get; set; }

    // This method as well: if it isn't in a class or a struct, it's violating CS0116
    public void DisplayLatitude()
    {
        Console.WriteLine($"Lat: {latitude}");
    }

    public struct Coordinate
    {

    }

    public class CoordinatePrinter
    {
        public void DisplayLongitude()
        {
            Console.WriteLine($"Longitude: {longitude}");
        }

        public void DisplayLocation()
        {
            Console.WriteLine($"Location: {LocationName}");
        }
    }
}
```

Note that in C#, methods and variables must be declared and defined within a struct or class. For more information on program structure in C#, see the General Structure of a C# Program article. To fix this error, rewrite your code such that all methods and fields are contained within either a struct or a class:

```
namespace x
{
    // A namespace can be placed within another namespace
    using System;

    // These variables are now placed within a struct, so CS0116 is no longer violated
    public struct Coordinate
    {
        public int Latitude;
        public int Longitude;
    }

    // The methods and fields are now placed within a class, and the compiler is satisfied
    public class CoordinatePrinter
    {
        Coordinate coord;
        public string LocationName { get; set; }

        public void DisplayLatitude()
        {
            Console.WriteLine($"Lat: {coord.Latitude}");
        }

        public void DisplayLongitude()
        {
            Console.WriteLine($"Longitude: {coord.Longitude}");
        }

        public void DisplayLocation()
        {
            Console.WriteLine($"Location: {LocationName}");
        }
    }
}
```

See also

- General Structure of a C# Program
- Classes and Structs
- Namespaces

Compiler Error CS0120

1/23/2019 • 2 minutes to read • Edit Online

An object reference is required for the nonstatic field, method, or property 'member'

In order to use a non-static field, method, or property, you must first create an object instance. For more information about static methods, see Static Classes and Static Class Members. For more information about creating instances of classes, see Instance Constructors.

The following sample generates CS0120:

```
// CS0120_1.cs
public class MyClass
{
   // Non-static field
   public int i;
   // Non-static method
   public void f(){}
   // Non-static property
   int Prop
   {
      get
      {
         return 1;
      }
   }

   public static void Main()
   {
      i = 10;   // CS0120
      f();   // CS0120
      int p = Prop;   // CS0120
      // try the following lines instead
      // MyClass mc = new MyClass();
      // mc.i = 10;
      // mc.f();
      // int p = mc.Prop;
   }
}
```

CS0120 will also be generated if there is a call to a non-static method from a static method, as follows:

```
// CS0120_2.cs
// CS0120 expected
using System;

public class MyClass
{
   public static void Main()
   {
      TestCall();   // CS0120
      // To call a non-static method from Main,
      // first create an instance of the class.
      // Use the following two lines instead:
      // MyClass anInstanceofMyClass = new MyClass();
      // anInstanceofMyClass.TestCall();
   }

   public void TestCall()
   {
   }
}
```

Similarly, a static method cannot call an instance method unless you explicitly give it an instance of the class, as follows:

```
// CS0120_3.cs
using System;

public class MyClass
{
   public static void Main()
   {
      do_it("Hello There");   // CS0120
   }

   private void do_it(string sText)
   // You could also add the keyword static to the method definition:
   // private static void do_it(string sText)
   {
      Console.WriteLine(sText);
   }
}
```

See also

- Classes and Structs

Compiler Error CS0122

5/4/2018 • 2 minutes to read • Edit Online

'member' is inaccessible due to its protection level

The access modifier for a class member prevents accessing the member. For more information, see Access Modifiers.

One cause of this (not shown in the sample below) can be omitting the **/out** compiler flag on the target of a friend assembly. For more information, see Friend Assemblies and /out (C# Compiler Options)

Example

The following sample generates CS0122:

```
// CS0122.cs
public class MyClass
{
    // Make public to resolve CS0122
    void MyMethod()
    {
    }
}

public class MyClass2
{
    public static int Main()
    {
        MyClass a = new MyClass();
        // MyMethod is private
        a.MyMethod();   // CS0122
        return 0;
   }
}
```

Compiler Error CS0134

5/4/2018 • 2 minutes to read • Edit Online

'variable' is of type 'type'. A const field of a reference type other than string can only be initialized with null.

A constant-expression is an expression that can be fully evaluated at compile-time. Because the only way to create a non-null value of a reference-type is to apply the new operator, and because the new operator is not permitted in a constant-expression, the only possible value for constants of reference-types other than string is null.

If you encounter this error by trying to create a const string array, the solution is to make the array readonly, and initialize it in the constructor.

Example

The following example generates CS0134.

```
// CS0134.cs
// compile with: /target:library
class MyTest {}

class MyClass
{
   const MyTest test = new MyTest();   // CS0134

   //OK
   const MyTest test2 = null;
   const System.String test3 = "test";
}
```

Compiler Error CS0151

8/15/2018 • 2 minutes to read • Edit Online

A value of an integral type expected

A variable was used in a situation where an integral data type was required. For more information, see Types.

Example of ambiguous conversion

This error can occur when there is no conversion or if the available implicit conversions result in an ambiguous situation. The following example generates CS0151:

```
public class MyClass
{
   public static implicit operator int (MyClass aa)
   {
      return 0;
   }

   public static implicit operator long (MyClass aa)
   {
      return 0;
   }

   public static void Main()
   {
      MyClass a = new MyClass();

      // Compiler cannot choose between int and long.
      switch (a)   // CS0151
      // try the following line instead
      // switch ((int)a)
      {
         case 1:
            break;
      }
   }
}
```

Example of void method

A void method invocation in a switch match expression generates CS0151. You can fix the error by calling a method that returns an integral type such as int or long instead.

```
class C
{
    static void Main()
    {
        switch (M()) // CS0151
        {
            default:
                break;
        }
    }

    static void M()
    {
    }
}
```

Compiler Error CS0163

5/4/2018 • 2 minutes to read • Edit Online

Control cannot fall through from one case label ('label') to another

When a switch statement contains more than one switch section, you must explicitly terminate each section, including the last one, by using one of the following keywords:

- return
- goto
- break
- throw
- continue

If you want to implement "fall through" behavior from one section to the next, use `goto case #`. For more information and examples, see switch.

The following sample generates CS0163.

```
// CS0163.cs
public class MyClass
{
    public static void Main()
    {
        int i = 0;

        switch (i)   // CS0163
        {
            // Compiler error CS0163 is reported on the following line.
            case 1:
                i++;
            // To resolve the error, uncomment one of the following example statements.
            // return;
            // break;
            // goto case 3;

            case 2:
                i++;
                return;

            case 3:
                i = 0;
                return;

            // Compiler error CS0163 is reported on the following line.
            default:
                Console.WriteLine("Default");
                // To resolve the error, uncomment the following line:
            //break;
        }
    }
```

Compiler Error CS0165

10/13/2018 • 2 minutes to read • Edit Online

Use of unassigned local variable 'name'

The C# compiler does not allow the use of uninitialized variables. If the compiler detects the use of a variable that might not have been initialized, it generates compiler error CS0165. For more information, see Fields. Note that this error is generated when the compiler encounters a construct that might result in the use of an unassigned variable, even if your particular code does not. This avoids the necessity of overly-complex rules for definite assignment.

For more information, see Why does a recursive lambda cause a definite assignment error?.

Example

The following sample generates CS0165:

```
// CS0165.cs
using System;

class MyClass
{
    public int i;
}

class MyClass2
{
    public static void Main(string[] args)
    {
        // i and j are not initialized.
        int i, j;

        // You can provide a value for args[0] in the 'Command line arguments'
        // text box on the Debug tab of the project Properties window.
        if (args[0] == "test")
        {
            i = 0;
        }
        // If the following else clause is absent, i might not be
        // initialized.
        //else
        //{
        //    i = 1;
        //}

        // Because i might not have been initialized, the following
        // line causes CS0165.
        j = i;

        // To resolve the error, uncomment the else clause of the previous
        // if statement, or initialize i when you declare it.

        // The following example causes CS0165 because myInstance is
        // declared but not instantiated.
        MyClass myInstance;
        // The following line causes the error.
        myInstance.i = 0;

        // To resolve the error, replace the previous declaration with
        // the following line.
        //MyClass myInstance = new MyClass();
    }
}
```

Example

Compiler error CS0165 can occur in recursive delegate definitions. You can avoid the error by defining the delegate in two statements so that the variable is not used before it is initialized. The following example demonstrates the error and the resolution.

```
class Program
{
    delegate void Del();
    static void Main(string[] args)
    {
        // The following line causes CS0165 because variable d is used
        // as an argument before it has been initialized.
        Del d = delegate() { System.Console.WriteLine(d); };

        //// To resolve the error, initialize d in a separate statement.
        //Del d = null;
        //// After d is initialized, you can use it as an argument.
        //d = delegate() { System.Console.WriteLine(d); };
        //d();
    }
}
```

Compiler Error CS0173

8/15/2018 • 2 minutes to read • Edit Online

Type of conditional expression cannot be determined because there is no implicit conversion between 'class1' and 'class2'

Conversions between classes are useful when you want objects of different classes to work with the same code. However, two classes that work together cannot have mutual and redundant conversions, or no implicit conversions. The types of `class1` and `class2` are determined independently, and the more general type is selected as the type of the conditional expression. For more information about how types are determined, see Conditional operator cannot cast implicitly.

To resolve CS0173, verify that there is one and only one implicit conversion between `class1` and `class2`, regardless of which direction the conversion is in and regardless of which class the conversion is in. For more information, see Implicit Numeric Conversions Table and Conversion Operators.

Example

The following examples generate compiler error CS0173:

```
public class C {}

public class A
{
    // The following code defines an implicit conversion operator from
    // type C to type A.
    //public static implicit operator A(C c)
    //{
    //    A a = new A();
    //    a = c;
    //    return a;
    //}
}

public class MyClass
{
    public static void F(bool b)
    {
        A a = new A();
        C c = new C();

        // The following line causes CS0173 because there is no implicit
        // conversion from a to c or from c to a.
        object o = b ? a : c;

        // To resolve the error, you can cast a and c.
        // object o = b ? (object)a : (object)c;

        // Alternatively, you can add a conversion operator from class C to
        // class A, or from class A to class C, but not both.
    }

   public static void Main()
   {
      F(true);
   }
}
```

```
class M
{
    static int Main ()
    {
        int X = 1;
        // The following line causes CS0173.
        object o = (X == 0) ? null : null;
        return -1;
    }
}
```

Compiler Error CS0178

10/13/2018 • 2 minutes to read • Edit Online

Invalid rank specifier: expected ',' or ']'

An array initialization was ill-formed. For example, when specifying the array dimensions, you can specify the following:

- A number in brackets
- Empty brackets
- A comma enclosed in brackets

For more information, see Arrays and the C# specification (C# Language Specification) section on array initializers.

Example

The following sample generates CS0178.

```
// CS0178.cs
class MyClass
{
   public static void Main()
   {
      int a = new int[5][,][][5];   // CS0178
      int[,] b = new int[3,2];   // OK

      int[][] c = new int[10][];
      c[0] = new int[5][5];   // CS0178
      c[0] = new int[2];   // OK
      c[1] = new int[2]{1,2};   // OK
   }
}
```

Compiler Error CS0188

1/23/2019 • 2 minutes to read • Edit Online

The 'this' object cannot be used before all of its fields are assigned to

All fields in a `struct` have to be assigned by a constructor before the constructor can call a method in the `struct`.

If you see this error when trying to initialize a property in a struct constructor, the solution is to change the constructor parameter to specify the backing field instead of the property itself. Auto-implemented properties should be avoided in structs because they have no backing field and therefore cannot be initialized in any way from the constructor.

For more information, see Using Structs.

Example

The following sample generates CS0188:

```
// CS0188.cs
// compile with: /t:library
namespace MyNamespace
{
    class MyClass
    {
        struct S
        {
            public int a;

            void MyMethod()
            {
            }

            S(int i)
            {
                // a = i;
                MyMethod();  // CS0188
            }
        }
        public static void Main()
        { }

    }
}
```

See also

- Structs
- Auto-Implemented Properties

Compiler Error CS0201

1/23/2019 • 2 minutes to read • Edit Online

Only assignment, call, increment, decrement, and new object expressions can be used as a statement

The compiler generates an error when it encounters an invalid statement. An invalid statement is any line or series of lines ending in a semicolon that does not represent an assignment (=), method call (), new, -- or ++ operation. For more information, see Statements, Expressions, and Operators.

Example

The following sample generates CS0201 because 2 * 3 is an expression, not a statement. To make the code compile, try assigning the value of the expression to a variable.

```
// CS0201.cs
public class MainClass
{
   public static void Main()
   {
      2 * 3;   // CS0201
      // Try the following line instead.
      //   int i = 2 * 3;
   }
}
```

Example

The following sample generates CS0201 because checked by itself is not a statement, even though it is parameterized by an increment operation.

```
// CS0201_b.cs
// compile with: /target:library
public class MyList<T>
{
   public void Add(T x)
   {
      int i = 0;
      if ( (object)x == null)
      {
         checked(i++);   // CS0201

         // OK
         checked {
            i++;
         }
      }
   }
}
```

See also

- C# Compiler Errors

Compiler Error CS0229

12/8/2018 • 2 minutes to read • Edit Online

Ambiguity between 'member1' and 'member2'

Members of different interfaces have the same name. If you want to keep the same names, you must qualify the names. For more information, see Interfaces.

> **NOTE**
> In some cases, this ambiguity can be resolved by providing an explicit prefix to the identifier via a using alias.

Example

The following example generates CS0229:

```
// CS0229.cs

interface IList
{
    int Count
    {
        get;
        set;
    }

    void Counter();
}

interface ICounter
{
    double Count
    {
        get;
        set;
    }
}

interface IListCounter : IList, ICounter {}

class MyClass
{
    void Test(IListCounter x)
    {
        x.Count = 1;  // CS0229
        // Try one of the following lines instead:
        // ((IList)x).Count = 1;
        // or
        // ((ICounter)x).Count = 1;
    }

    public static void Main() {}
}
```

Compiler Error CS0233

5/4/2018 • 2 minutes to read • Edit Online

'identifier' does not have a predefined size, therefore sizeof can only be used in an unsafe context (consider using System.Runtime.InteropServices.Marshal.SizeOf)

The sizeof operator can only be used for types that are compile-time constants. If you are getting this error, make sure that the size of the identifier can be determined at compile time. If it cannot, then use SizeOf instead of `sizeof`.

Example

The following example generates CS0233:

```
// CS0233.cs
using System;
using System.Runtime.InteropServices;

[StructLayout(LayoutKind.Sequential)]
public struct S
{
    public int a;
}

public class MyClass
{
    public static void Main()
    {
        S myS = new S();
        Console.WriteLine(sizeof(S));   // CS0233
        // Try the following line instead:
        // Console.WriteLine(Marshal.SizeOf(myS));
   }
}
```

Compiler Error CS0234

5/4/2018 • 2 minutes to read • Edit Online

The type or namespace name 'name' does not exist in the namespace 'namespace' (are you missing an assembly reference?)

A type was expected. Possible causes for this error include the following:

- An assembly that contains the definition of a type was not referenced in the compilation; use /reference (Import Metadata) to specify the assembly
- You passed a variable name to the typeof operator.
- You tried to reference an assembly that is not part of your target .NET Framework profile. For more information, see Troubleshooting .NET Framework Targeting Errors.

If you see this error after moving code from one development machine to another, make sure that the project on the new machine has the correct references, and that the versions of the assemblies are the same as on the old machine. You can also use the Object Browser to inspect an assembly and verify whether it contains the types that you expect it to contain.

The following sample generates CS0234:

```
// CS0234.cs
public class C
{
   public static void Main()
   {
      System.DateTime x = new System.DateTim();   // CS0234
      // try the following line instead
      // System.DateTime x = new System.DateTime();
   }
}
```

Compiler Error CS0246

8/31/2018 • 4 minutes to read • Edit Online

The type or namespace name 'type/namespace' could not be found (are you missing a using directive or an assembly reference?)

A type or namespace that is used in the program was not found. You might have forgotten to reference (-reference) the assembly that contains the type, or you might not have added the required using directive. Or, there might be an issue with the assembly you are trying to reference.

The following situations cause compiler error CS0246.

- Did you misspell the name of the type or namespace? Without the correct name, the compiler cannot find the definition for the type or namespace. This often occurs because the casing used in the name of the type is not correct. For example, `Dataset ds;` generates CS0246 because the s in Dataset must be capitalized.

- If the error is for a namespace name, did you add a reference (-reference) to the assembly that contains the namespace? For example, your code might contain the directive `using Accessibility`. However, if your project does not reference the assembly Accessibility.dll, error CS0246 is reported. For more information, see Managing references in a project

- If the error is for a type name, did you include the proper using directive, or, alternatively, fully qualify the name of the type? Consider the following declaration: `DataSet ds`. To use the `DataSet` type, you need two things. First, you need a reference to the assembly that contains the definition for the `DataSet` type. Second, you need a `using` directive for the namespace where `DataSet` is located. For example, because `DataSet` is located in the **System.Data** namespace, you need the following directive at the beginning of your code: `using System.Data`.

  The `using` directive is not required. However, if you omit the directive, you must fully qualify the `DataSet` type when referring to it. Full qualification means that you specify both the namespace and the type each time you refer to the type in your code. If you omit the `using` directive in the previous example, you must write `System.Data.DataSet ds` to declare `ds` instead of `DataSet ds`.

- Did you use a variable or some other language element where a type was expected? For example, in an **is** statement, if you use a `Type` object instead of an actual type, you get error CS0246.

- Did you reference the assembly that was built against a higher framework version than the target framework of the program? Or did you reference the project that is targeting a higher framework version than the target framework of the program? For example, you work on the project that is targeting .NET Framework 4.6.1 and use the type from the project that is targeting .NET Framework 4.7.1. Then you get error CS0246.

- Did you use a *using alias directive* without fully qualifying the type name? A `using` alias directive does not use the `using` directives in the source code file to resolve types. The following example generates CS0246 because the type `List<int>` is not fully qualified. The `using` directive for `System.Collections.Generic` does not prevent the error.

```
using System.Collections.Generic;

// The following declaration generates CS0246.
using myAliasName = List<int>;

// To avoid the error, fully qualify List.
using myAliasName2 = System.Collections.Generic.List<int>;
```

If you get this error in code that was previously working, first look for missing or unresolved references in Solution Explorer. Do you need to re-install a NuGet package? For information about how the build system searches for references, see Resolving file references in team build. If all references seem to be correct, look in your source control history to see what has changed in your .csproj file and/or your local source file.

If you haven't successfully accessed the reference yet, use the Object Browser to inspect the assembly that is supposed to contain this namespace and verify that the namespace is present. If you verify with Object Browser that the assembly contains the namespace, try removing the `using` directive for the namespace and see what else breaks. The root problem may be with some other type in another assembly.

The following example generates CS0246 because a necessary `using` directive is missing.

```
// CS0246.cs
//using System.Diagnostics;

public class MyClass
{
    // The following line causes CS0246. To fix the error, uncomment
    // the using directive for the namespace for this attribute,
    // System.Diagnostics.
    [Conditional("A")]
    public void Test()
    {
    }

    public static void Main()
    {
    }
}
```

The following example causes CS0246 because an object of type `Type` was used where an actual type was expected.

```
// CS0246b.cs
using System;

class ExampleClass
{
    public bool supports(object o, Type t)
    {
        // The following line causes CS0246. You must use an
        // actual type, such as ExampleClass, String, or Type.
        if (o is t)
        {
            return true;
        }
        return false;
    }
}

class Program
{
    public static void Main()
    {
        ExampleClass myC = new ExampleClass();
        myC.supports(myC, myC.GetType());
    }
}
```

Compiler Error CS0260

1/23/2019 • 2 minutes to read • Edit Online

Missing partial modifier on declaration of type 'type'; another partial declaration of this type exists

This error indicates that you have declared multiple classes that have the same name. In addition, at least one but not all of the declarations contains the `partial` modifier. If you want to define a class in several parts, you must declare each part by using the keyword `partial`.

This error also occurs if you declare a class and accidentally give it the same name as a partial class that's declared elsewhere in the same namespace.

The following sample generates CS0260:

```
// CS0260.cs
// You must mark both parts of the definition of class C
// by using the partial keyword.

// The following line causes CS0260. To resolve the error, add
// the 'partial' keyword to the declaration.
class C
{
}

partial class C
{
}
```

See also

- Partial Classes and Methods

Compiler Error CS0266

5/4/2018 • 2 minutes to read • Edit Online

Cannot implicitly convert type 'type1' to 'type2'. An explicit conversion exists (are you missing a cast?)

This error occurs when your code tries to convert between two types that cannot be implicitly converted, but where an explicit conversion is available. For more information, see Casting and Type Conversions.

The following code shows examples that generate CS0266.

```
// CS0266.cs
class MyClass
{
    public static void Main()
    {
        // You cannot implicitly convert a double to an integer.
        double d = 3.2;

        // The following line causes compiler error CS0266.
        int i1 = d;

        // However, you can resolve the error by using an explicit conversion.
        int i2 = (int)d;

        // You cannot implicitly convert an object to a class type.
        object obj = "MyString";

        // The following assignment statement causes error CS0266.
        MyClass myClass = obj;

        // You can resolve the error by using an explicit conversion.
        MyClass c = ( MyClass )obj;

        // You cannot implicitly convert a base class object to a derived class type.
        MyClass mc = new MyClass();
        DerivedClass dc = new DerivedClass();

        // The following line causes compiler error CS0266.
        dc = mc;

        // You can resolve the error by using an explicit conversion.
        dc = (DerivedClass)mc;
    }
}

class DerivedClass : MyClass
{
}
```

Compiler Error CS0269

5/4/2018 • 2 minutes to read • Edit Online

Use of unassigned out parameter 'parameter'

The compiler could not verify that the out parameter was assigned a value before it was used; its value may be undefined when assigned. Be sure to assign a value to `out` parameters in the called method before accessing the value. If you need to use the value of the variable passed in, use a `ref` parameter instead. For more information, see Passing Parameters.

Example

The following sample generates CS0269:

```
// CS0269.cs
class C
{
    public static void F(out int i)
    // One way to resolve the error is to use a ref parameter instead
    // of an out parameter.
    //public static void F(ref int i)
    {
        // The following line causes a compiler error because no value
        // has been assigned to i.
        int k = i;  // CS0269
        i = 1;
        // The error does not occur if the order of the two previous
        // lines is reversed.
    }

    public static void Main()
    {
        int myInt = 1;
        F(out myInt);
        // If the declaration of method F is changed to require a ref
        // parameter, ref must be specified in the call as well.
        //F(ref myInt);
    }
}
```

Example

This could also occur if the initialization of a variable occurs in a try block, which the compiler is unable to verify will execute successfully:

```
// CS0269b.cs
class C
{
    public static void F(out int i)
    {
        try
        {
            // Assignment occurs, but compiler can't verify it
            i = 1;
        }
        catch
        {
        }

        int k = i;  // CS0269
        i = 1;
    }

    public static void Main()
    {
        int myInt;
        F(out myInt);
    }
}
```

Compiler Error CS0270

5/4/2018 • 2 minutes to read • Edit Online

Array size cannot be specified in a variable declaration (try initializing with a 'new' expression)

This error occurs when a size is specified as part of an array declaration. To resolve, use the new Operator expression.

The following example generates CS0270:

```
// CS0270.cs
// compile with: /t:module

public class Test
{
   int[10] a;   // CS0270
   // To resolve, use the following line instead:
   // int[] a = new int[10];
}
```

Compiler Error CS0304

1/23/2019 • 2 minutes to read • Edit Online

Cannot create an instance of the variable type 'type' because it does not have the new() constraint

When you implement a generic class, and you want to use the `new` keyword to create a new instance of any type that is supplied for a type parameter `T`, you must apply the new() constraint to `T` in the class declaration, as shown in the following example.

```
class C<T> where T : new()
```

The `new()` constraint enforces type safety by guaranteeing that any concrete type that is supplied for `T` has a default, parameterless constructor. CS0304 occurs if you attempt to use the `new` operator in the body of the class to create an instance of type parameter `T` when `T` does not specify the `new()` constraint. On the client side, if code attempts to instantiate the generic class with a type that has no default constructor, that code will generate Compiler Error CS0310.

The following example generates CS0304.

```
// CS0304.cs
// Compile with: /target:library.
class C<T>
{
    // The following line generates CS0304.
    T t = new T();
}
```

The `new` operator also is not allowed in methods of the class.

```
// Compile with: /target:library.
class C<T>
{
    public void ExampleMethod()
    {
        // The following line generates CS0304.
        T t = new T();
    }
}
```

To avoid the error, declare the class by using the `new()` constraint, as shown in the following example.

```
// Compile with: /target:library.
class C<T> where T : new()
{
    T t = new T();

    public void ExampleMethod()
    {
        T t = new T();
    }
}
```

See also

- C# Compiler Errors

Compiler Error CS0310

5/4/2018 • 2 minutes to read • Edit Online

The type 'typename' must be a non-abstract type with a public parameterless constructor in order to use it as parameter 'parameter' in the generic type or method 'generic'

The generic type or method defines a new constraint in its where clause, so any type must have a public parameterless constructor in order to be used as a type argument for that generic type or method. To avoid this error, make sure that the type has the correct constructor, or modify the constraint clause of the generic type or method.

Example

The following sample generates CS0310:

```
// CS0310.cs
using System;

class G<T> where T : new()
{
    T t;

    public G()
    {
        t = new T();
        Console.WriteLine(t);
    }
}

class B
{
    private B() { }
    // Try this instead:
    // public B() { }
}

class CMain
{
    public static void Main()
    {
        G<B> g = new G<B>();   // CS0310
        Console.WriteLine(g.ToString());
    }
}
```

Compiler Error CS0311

1/23/2019 • 2 minutes to read • Edit Online

The type 'type1' cannot be used as type parameter 'T' in the generic type or method '<name>'. There is no implicit reference conversion from 'type1' to 'type2'.

When a constraint is applied to a generic type parameter, an implicit identity or reference conversion must exist from the concrete argument to the type of the constraint.

To correct this error

1. Change the argument you are using to create the class.
2. If you own the class, you can remove the constraint or else do something to enable an implicit reference or identity conversion. For example, you can make the second type inherit from the first.

Example

```
// cs0311.cs
class B {}
class C {}
class Test<T> where T : C
{ }

class Program
{
    static void Main()
    {
        Test<B> test = new Test<B>(); //CS0311
    }
}
```

If this error occurs when trying to use a value-type argument, notice that an implicit numeric conversion, for example from `short` to `int`, does not satisfy a generic type parameter.

See also

- Constraints on Type Parameters

Compiler Error CS0413

5/4/2018 • 2 minutes to read • Edit Online

The type parameter 'type parameter' cannot be used with the 'as' operator because it does not have a class type constraint nor a 'class' constraint

This error occurs if a generic type uses the as operator, but that generic type does not have a class type constraint. The `as` operator is only allowed with reference types, so the type parameter must be constrained to guarantee that it is not a value type. To avoid this error, use a class type constraint or a reference type constraint.

This is because the `as` operator could return `null`, which is not a possible value for a value type, and the type parameter must be treated as a value type unless it is a class type constraint or a reference type constraint.

Example

The following sample generates CS0413.

```
// CS0413.cs
// compile with: /target:library
class A {}
class B : A {}

class CMain
{
   A a = null;
   public void G<T>()
   {
      a = new A();
      System.Console.WriteLine (a as T);  // CS0413
   }

   // OK
   public void H<T>() where T : A
   {
      a = new A();
      System.Console.WriteLine (a as T);
   }
}
```

Compiler Error CS0417

1/23/2019 • 2 minutes to read • Edit Online

'identifier': cannot provide arguments when creating an instance of a variable type

This error occurs if a call to the `new` operator on a type parameter has arguments. The only constructor that can be called by using the `new` operator on an unknown parameter type is a constructor that has no arguments. If you need to call another constructor, consider using a class type constraint or interface constraint.

Example

The following example generates CS0417:

```
// CS0417
class ExampleClass<T> where T : new()
{
    // The following line causes CS0417.
    T instance1 = new T(1);

    // The following line doesn't cause the error.
    T instance2 = new T();
}
```

See also

- Constraints on Type Parameters

Compiler Error CS0433

5/4/2018 • 2 minutes to read • Edit Online

The type TypeName1 exists in both TypeName2 and TypeName3

Two different assemblies referenced in your application contain the same namespace and type, which produces ambiguity.

To resolve this error, use the alias feature of the /reference (C# Compiler Options) compiler option or do not reference one of your assemblies.

Example

This code creates the DLL with the first copy of the ambiguous type.

```
// CS0433_1.cs
// compile with: /target:library
namespace TypeBindConflicts
{
   public class AggPubImpAggPubImp {}
}
```

Example

This code creates the DLL with the second copy of the ambiguous type.

```
// CS0433_2.cs
// compile with: /target:library
namespace TypeBindConflicts
{
   public class AggPubImpAggPubImp {}
}
```

Example

The following example generates CS0433.

```
// CS0433_3.cs
// compile with: /reference:cs0433_1.dll /reference:cs0433_2.dll
using TypeBindConflicts;
public class Test
{
   public static void Main()
   {
      AggPubImpAggPubImp n6 = new AggPubImpAggPubImp();   // CS0433
   }
}
```

Example

The following example shows how you can use the alias feature of the **/reference** compiler option to resolve this CS0433 error.

```
// CS0433_4.cs
// compile with: /reference:cs0433_1.dll /reference:TypeBindConflicts=cs0433_2.dll
using TypeBindConflicts;
public class Test
{
   public static void Main()
   {
      AggPubImpAggPubImp n6 = new AggPubImpAggPubImp();
   }
}
```

Compiler Error CS0445

5/4/2018 • 2 minutes to read • Edit Online

Cannot modify the result of an unboxing conversion

The result of an unboxing conversion is a temporary variable. The compiler prevents you from modifying such variables because any modification would go away when the temporary variable goes away. To fix this, declare a new value-type variable to store the intermediate expression, and assign the result of the unboxing conversion to that variable.

The following code generates CS0455.

```
// CS0445.CS
class UnboxingTest
{
    public static void Main()
    {
        Point p;
        p.x = 1;
        p.y = 2;
        object obj = p;
        // The following line generates CS0445, because the result
        // of unboxing obj is a temporary variable.
        ((Point)obj).x = 2;

        // The following lines resolve the error.

        // Store the result of the unboxing conversion in p2.
        Point p2;
        p2 = (Point)obj;
        // Then you can modify the unboxed value.
        p2.x = 2;
    }
}

struct Point
{
    public int x, y;
}
```

Compiler Error CS0446

5/4/2018 • 2 minutes to read • Edit Online

Foreach cannot operate on a 'Method or Delegate'. Did you intend to invoke the 'Method or Delegate'?

This error is caused by specifying a method without parentheses or an anonymous method without parentheses in the part of the `foreach` statement where you would normally put a collection class. Note that it is valid, though unusual, to put a method call in that location, if the method returns a collection class.

Example

The following code will generate CS0446.

```
// CS0446.cs
using System;
class Tester
{
    static void Main()
    {
        int[] intArray = new int[5];
        foreach (int i in M) { } // CS0446
    }
    static void M() { }
}
```

Compiler Error CS0504

5/4/2018 • 2 minutes to read • Edit Online

The constant 'variable' cannot be marked static

If a variable is const, it is also static. If you want a **const** and **static** variable, just declare that variable as **const**; if all you want is a **static** variable, just mark it **static**.

The following sample generates CS0504:

```
// CS0504.cs
namespace x
{
   abstract public class clx
   {
      static const int i = 0;   // CS0504, cannot be both static and const
      abstract public void f();
   }
}
```

Compiler Error CS0507

5/4/2018 • 2 minutes to read • Edit Online

'function1' : cannot change access modifiers when overriding 'access' inherited member 'function2'

An attempt was made to change the access specification in a method override.

Example

The following sample generates CS0507.

```
// CS0507.cs
abstract public class clx
{
   virtual protected void f() {}
}

public class cly : clx
{
   public override void f() {}   // CS0507
   public static void Main() {}
}
```

Example

CS0507 can also occur if a class attempts to override a method marked as `protected internal` defined in referenced metadata. In this situation, the overriding method should be marked as `protected`.

```
// CS0507_b.cs
// compile with: /target:library
abstract public class clx
{
   virtual protected internal void f() {}
}
```

Example

The following sample generates CS0507.

```
// CS0507_c.cs
// compile with: /reference:cs0507_b.dll
public class cly : clx
{
   protected internal override void f() {}   // CS0507
   // try the following line instead
   // protected override void f() {}   // OK

   public static void Main() {}
}
```

Compiler Error CS0518

5/4/2018 • 2 minutes to read • Edit Online

Predefined type 'type' is not defined or imported

The main cause for this problem is that the project is not importing mscorlib.dll, which defines the entire System namespace. This can be caused by one of the following:

- The /nostdlib option from the command line compiler has been specified. The /nostdlib option prevents the import of mscorlib.dll. Use this option if you want to define or create a user-specific System namespace.
- An incorrect mscorlib.dll is referenced.
- A corrupt Visual Studio .NET or .NET Framework common language runtime installation exists.
- Residual components from an earlier installation that are incompatible with the latest installation remain.

To resolve this problem, take one of the following actions:

- Do not specify the /nostdlib option from the command line compiler.
- Make sure that the project refers to the correct mscorlib.dll.
- Reinstall the .NET Framework common language runtime (if the previous solutions do not solve the problem).

Compiler Error CS0523

5/4/2018 • 2 minutes to read • Edit Online

Struct member 'struct2 field' of type 'struct1' causes a cycle in the struct layout

The definitions of two structs include recursive references. Change the struct definitions such that each does not define itself on the other. This limitation applies only to structs, since structs are value types. If using recursive references, declare your types as classes.

The following sample generates CS0523:

```
// CS0523.cs
// compile with: /target:library
struct RecursiveLayoutStruct1
{
   public RecursiveLayoutStruct2 field;
}

struct RecursiveLayoutStruct2
{
   public RecursiveLayoutStruct1 field;   // CS0523
}
```

Compiler Error CS0545

5/4/2018 • 2 minutes to read • Edit Online

'function' : cannot override because 'property' does not have an overridable get accessor

A try was made to define an override for a property accessor when the base class has no such definition to override. You can resolve this error by:

- Adding a `set` accessor in the base class.
- Removing the `set` accessor from the derived class.
- Hiding the base class property by adding the new keyword to a property in a derived class.
- Making the base class property virtual.

For more information, see Using Properties.

Example

The following sample generates CS0545.

```
// CS0545.cs
// compile with: /target:library
// CS0545
public class a
{
   public virtual int i
   {
      set {}

      // Uncomment the following line to resolve.
      // get { return 0; }
   }
}

public class b : a
{
   public override int i
   {
      get { return 0; }
      set {}   // OK
   }
}
```

Compiler Error CS0552

5/4/2018 • 2 minutes to read • Edit Online

'conversion routine' : user defined conversion to/from interface

You cannot create a user-defined conversion to or from an interface. If you need the conversion routine, resolve this error by making the interface a class or derive a class from the interface.

The following sample generates CS0552:

```
// CS0552.cs
public interface ii
{
}

public class a
{
   // delete the routine to resolve CS0552
   public static implicit operator ii(a aa) // CS0552
   {
      return new ii();
   }

   public static void Main()
   {
   }
}
```

Compiler Error CS0563

8/31/2018 • 2 minutes to read • Edit Online

One of the parameters of a binary operator must be the containing type

The method declaration for an operator overload must follow certain guidelines.

Example

The following sample generates CS0563:

```
// CS0563.cs
public class iii
{
    public static implicit operator int(iii x)
    {
        return 0;
    }
    public static implicit operator iii(int x)
    {
        return null;
    }
    public static int operator +(int aa, int bb)   // CS0563
    // Use the following line instead:
    // public static int operator +(int aa, iii bb)
    {
        return 0;
    }
    public static void Main()
    {
    }
}
```

Compiler Error CS0570

5/4/2018 • 2 minutes to read • Edit Online

Property, indexer, or event 'name' is not supported by the language; try directly calling accessor method 'name!'

This error occurs when using imported metadata that was generated by another compiler. Your code attempted to use a class member that the compiler cannot process.

Example

The following C++ program uses an attribute, RequiredAttributeAttribute, which may not be consumed by other languages.

```
// CPP0570.cpp
// compile with: /clr /LD

using namespace System;
using namespace System::Runtime::CompilerServices;

namespace CS0570_Server {
   [RequiredAttributeAttribute(Int32::typeid)]
   public ref struct Scenario1 {
      int intVar;
   };

   public ref struct CS0570Class {
      Scenario1 ^ sc1_field;

      property virtual Scenario1 ^ sc1_prop {
         Scenario1 ^ get() { return sc1_field; }
      }

      Scenario1 ^ sc1_method() { return sc1_field; }
   };
};
```

Example

The following sample generates CS0570.

```
// CS0570.cs
// compile with: /reference:CPP0570.dll
using System;
using CS0570_Server;

public class C {
   public static int Main() {
      CS0570Class r = new CS0570Class();
      r.sc1_field = null;   // CS0570
      object o = r.sc1_prop;   // CS0570
      r.sc1_method();   // CS0570
   }
}
```

Compiler Error CS0571

5/4/2018 • 2 minutes to read • Edit Online

'function' : cannot explicitly call operator or accessor

Certain operators have internal names. For example, **op_Increment** is the internal name of the ++ operator. You should not use or explicitly call such method names.

The following sample generates CS0571:

```
// CS0571.cs
public class MyClass
{
   public static MyClass operator ++ (MyClass c)
   {
      return null;
   }

   public static int prop
   {
      get
      {
         return 1;
      }
      set
      {
      }
   }

   public static void Main()
   {
      op_Increment(null);   // CS0571
      // use the increment operator as follows
      // MyClass x = new MyClass();
      // x++;

      set_prop(1);      // CS0571
      // try the following line instead
      // prop = 1;
   }
}
```

Compiler Error CS0579

5/4/2018 • 2 minutes to read • Edit Online

Duplicate 'attribute' attribute

It is not possible to specify the same attribute more than once unless the attribute specifies **AllowMultiple=true** in its AttributeUsage.

Example

The following example generates CS0579.

```
// CS0579.cs
using System;
public class MyAttribute : Attribute
{
}

[AttributeUsage(AttributeTargets.All,AllowMultiple=true)]
public class MyAttribute2 : Attribute
{
}

public class z
{
    [MyAttribute, MyAttribute]     // CS0579
    public void zz()
    {
    }

    [MyAttribute2, MyAttribute2]   // OK
    public void zzz()
    {
    }

    public static void Main()
    {
    }
}
```

Compiler Error CS0592

1/23/2019 • 2 minutes to read • Edit Online

Attribute 'attribute' is not valid on this declaration type. It is valid on 'type' declarations only.

When you define an attribute, you define what constructs it can be applied to by specifying an `AttributeTargets` value. In the following example, the `MyAttribute` attribute can be applied to interfaces only, because the `AttributeUsage` attribute specifies `AttributeTargets.Interface`. The error is generated because the attribute is applied to a class (class `A`).

Example

The following sample generates CS0592:

```
// CS0592.cs
using System;

[AttributeUsage(AttributeTargets.Interface)]
public class MyAttribute : Attribute
{
}

[MyAttribute]
// Generates CS0592 because MyAttribute is not valid for a class.
public class A
{
    public static void Main()
    {
    }
}
```

See also

- Attributes

Compiler Error CS0616

5/4/2018 • 2 minutes to read • Edit Online

'class' is not an attribute class

An attempt was made to use a non-attribute class in an attribute block. All the attribute types need to be inherited from System.Attribute.

Example

The following sample generates CS0616.

```
// CS0616.cs
// compile with: /target:library
[CMyClass(i = 5)]   // CS0616
public class CMyClass {}
```

Example

The following sample shows how you might define an attribute:

```
// CreateAttrib.cs
// compile with: /target:library
using System;

[AttributeUsage(AttributeTargets.Class|AttributeTargets.Interface)]
public class MyAttr : Attribute
{
   public int Name = 0;
   public int Count = 0;

   public MyAttr (int iCount, int sName)
   {
      Count = iCount;
      Name = sName;
   }
}

[MyAttr(5, 50)]
class Class1 {}

[MyAttr(6, 60)]
interface Interface1 {}
```

Compiler Error CS0650

1/23/2019 • 2 minutes to read • Edit Online

Bad array declarator: To declare a managed array the rank specifier precedes the variable's identifier. To declare a fixed size buffer field, use the fixed keyword before the field type.

An array was declared incorrectly. In C#, unlike in C and C++, the square brackets follow the type, not the variable name. Also, realize that the syntax for a fixed size buffer differs from that of an array.

Example

The following example code generates CS0650.

```
// CS0650.cs
public class MyClass
{
   public static void Main()
   {
// Generates CS0650. Incorrect array declaration syntax:
      int myarray[2];

      // Correct declaration.
      int[] myarray2;

      // Declaration and initialization in one statement
      int[] myArray3= new int[2] {1,2}

      // Access an array element.
      myarray3[0] = 0;
    }
}
```

Example

The following example shows how to use the `fixed` keyword when you create a fixed size buffer:

```
// This code must appear in an unsafe block.
public struct MyArray
{
    public fixed char pathName[128];
}
```

See also

- Fixed Size Buffers
- fixed Statement
- Arrays

Compiler Error CS0686

7/18/2018 • 2 minutes to read • Edit Online

Accessor 'accessor' cannot implement interface member 'member' for type 'type'. Use an explicit interface implementation.

Suggested: This error can occur when implementing an interface that contains method names which conflict with the auto-generated methods associated with a property or event. The get/set methods for properties are generated as get_property and set_property, and the add/remove methods for events are generated as add_event and remove_event. If an interface contains either of these methods, a conflict occurs. To avoid this error, implement the methods using an explicit interface implementation. To do this, specify the function as:

```
Interface.get_property() { /* */ }
Interface.set_property() { /* */ }
```

Example

The following sample generates CS0686:

```
// CS0686.cs
interface I
{
    int get_P();
}

class C : I
{
    public int P
    {
        get { return 1; }  // CS0686
    }
}
// But the following is valid:
class D : I
{
    int I.get_P() { return 1; }
    public static void Main() {}
}
```

Example

This error can also occur when declaring events. The event construct automatically generates the `add_event` and `remove_event` methods, which could conflict with the methods of the same name in an interface, as in the following sample:

```
// CS0686b.cs
using System;

interface I
{
    void add_OnMyEvent(EventHandler e);
}

class C : I
{
    public event EventHandler OnMyEvent
    {
        add { }  // CS0686
        remove { }
    }
}

// Correct (using explicit interface implementation):
class D : I
{
    void I.add_OnMyEvent(EventHandler e) {}
    public static void Main() {}
}
```

Compiler error CS0702

1/23/2019 • 2 minutes to read • Edit Online

Constraint cannot be special class 'identifier'

The following types may not be used as constraints: Object, Array, or ValueType.

Example

The following sample generates CS0702:

```
// CS0702.cs
class C<T> where T : System.Array  // CS0702
{
}
```

See also

- Constraints on Type Parameters

Compiler Error CS0703

5/4/2018 • 2 minutes to read • Edit Online

Inconsistent accessibility: constraint type 'identifier' is less accessible than 'identifier'

A constraint may not force the generic parameter to be less accessible than the generic class itself. In the following example, while the generic class C<T> is declared public, the constraint attempts to force T to implement an internal interface. Even if this were allowed, only clients with internal access would be able to create the parameter for the class, so in effect, the class could be used only by clients with internal access.

To get rid of this error, make sure the access level of the generic class is not less restrictive than any classes or interfaces that appear in the bounds.

The following sample generates CS0703:

```
// CS0703.cs
internal interface I {}
public class C<T> where T : I  // CS0703 – I is internal; C<T> is public
{
}
```

Compiler Error CS0731

5/15/2018 • 2 minutes to read • Edit Online

The type forwarder for type 'type' in assembly 'assembly' causes a cycle

This error can only occur with improperly formed imported metadata. It cannot occur with only C# source.

Example

The following sample generates CS0731. The example consists of three files:

1. Circular.IL
2. Circular2.IL
3. CS0731.cs

First compile the .IL files as libraries, and then compile the .cs code referencing the two files.

```
// Circular.il
// compile with: /DLL /out=Circular.dll
.assembly extern circular2
{
  .ver 0:0:0:0
}
.assembly extern circular3
{
  .ver 0:0:0:0
}
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )   // .z\V.4..
  .ver 2:0:0:0
}
.assembly Circular
{
  .custom instance void [mscorlib]System.Runtime.CompilerServices.CompilationRelaxationsAttribute::.ctor(int32)
= ( 01 00 08 00 00 00 00 00 )
  .hash algorithm 0x00008004
  .ver 0:0:0:0
}
.class extern forwarder Circular.Referenced.TypeForwarder
{
  .assembly extern circular2
}
.module Circular.dll
// MVID: {880C2329-C915-42A0-83E9-9D10C3E6DBD0}
.imagebase 0x00400000
.file alignment 0x00000200
.stackreserve 0x00100000
.subsystem 0x0003       // WINDOWS_CUI
.corflags 0x00000001    //  ILONLY
// Image base: 0x04E40000
// ======== CLASS MEMBERS DECLARATION =========
.class public abstract auto ansi sealed beforefieldinit User
       extends [mscorlib]System.Object
{
  .method public hidebysig static class [circular2]Circular.Referenced.TypeForwarder
          F() cil managed
  {
    .maxstack  1
    newobj     instance void [circular2]Circular.Referenced.TypeForwarder::.ctor()
    ret
  }
}
```

```
// Circular2.il
// compile with: /DLL /out=Circular2.dll
.assembly extern mscorlib
{
  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )   // .z\V.4..
  .ver 2:0:0:0
}
.assembly extern Circular
{
  .ver 0:0:0:0
}
.assembly circular2
{
  .custom instance void [mscorlib]System.Runtime.CompilerServices.CompilationRelaxationsAttribute::.ctor(int32)
= ( 01 00 08 00 00 00 00 00 )
  .hash algorithm 0x00008004
  .ver 0:0:0:0
}
.class extern forwarder Circular.Referenced.TypeForwarder
{
  .assembly extern Circular
}
.module circular2.dll
// MVID: {8B3BE5C8-DBE1-49C4-BC72-DF35F0387C21}
.imagebase 0x00400000
.file alignment 0x00000200
.stackreserve 0x00100000
.subsystem 0x0003       //   WINDOWS_CUI
.corflags 0x00000001    //  ILONLY
// Image base: 0x04E40000
```

```
// CS0731.cs
// compile with: /reference:circular.dll /reference:circular2.dll
// CS0731 expected
class A {
   public static void Main() {
      User.F();
   }
}
```

Compiler Error CS0826

1/23/2019 • 2 minutes to read • Edit Online

No best type found for implicitly typed array.

Array elements must all be the same type or implicitly convertible to the same type according to the type inference rules used by the compiler. The best type must be one of the types present in the array expression. Elements will not be converted to a new type such as `object`. For an implicitly typed array, the compiler must infer the array type based on the type of elements assigned to it.

To correct this error

- Give the array an explicit type.
- Give all array elements the same type.
- Provide explicit casts on those elements that might be causing the problem.

Example

The following code generates CS0826 because the array elements are not all the same type, and the compiler's type inference logic does not find a single best type:

```
// cs0826.cs
public class C
{
    public static int Main()
    {
        var x = new[] { 1, "str" }; // CS0826

        char c = 'c';
        short s1 = 0;
        short s2 = -0;
        short s3 = 1;
        short s4 = -1;

        var array1 = new[] { s1, s2, s3, s4, c, '1' }; // CS0826
        return 1;
    }
}
```

See also

- Implicitly Typed Local Variables

Compiler Error CS0834

5/4/2018 • 2 minutes to read • Edit Online

A lambda expression must have an expression body to be converted to an expression tree.

Lambdas that are translated to expression trees must be expression lambdas; statement lambdas and anonymous methods can only be converted to delegate types.

To correct this error

1. Remove the statement from the lambda expression.

Example

The following example generates CS0834:

```
// cs0834.cs
using System;
using System.Linq;
using System.Linq.Expressions;

public class C
{
    public static int Main()
    {
        Expression<Func<int, int>> e = x => { return x; }; // CS0834
    }
}
```

Compiler Error CS0840

1/23/2019 • 2 minutes to read • Edit Online

'Property name' must declare a body because it is not marked abstract or extern. Automatically implemented properties must define both get and set accessors.

Unless a regular property is marked as `abstract` or `extern`, or is a member of a `partial` type, it must supply a body. Auto-implemented properties do not provide accessor bodies, but they must specify both accessors. To create a read-only auto-implemented property, make the set accessor `private`.

To correct this error

1. Supply the missing body or accessor or else use the abstract, extern, or partial (Type) modifiers on it and/or its enclosing type.

Example

The following example generates CS0840:

```
// cs0840.cs
// Compile with /target:library
using System;
class Test
{
    public int myProp { get; } // CS0840

    // to create a read-only property
    // try the following line instead
    public int myProp2 { get; private set; }

}
```

See also

- Auto-Implemented Properties

Compiler Error CS0843

5/4/2018 • 2 minutes to read • Edit Online

Backing field for automatically implemented property 'name' must be fully assigned before control is returned to the caller. Consider calling the default constructor from a constructor initializer.

To assign a value to an automatically-implemented property from a constructor, you must first invoke the default constructor to create the object.

To correct this error

1. Add a call to the default constructor in a constructor initializer as shown in the following example. Note the use of `: this()`. For more information, see this.

Example

The following code generates CS0843:

```
// cs0843.cs
struct S
{
    public int AIProp { get; set; }
    public S(int i){} //CS0843
    // Try the following lines instead.
   // public S(int i) : this()
   // {
   //     AIProp = i;
   // }
}

class Test
{
    static int Main()
    {
        return 1;
    }
}
```

Compiler Error CS0845

5/4/2018 • 2 minutes to read • Edit Online

An expression tree lambda may not contain a coalescing operator with a null literal left-hand side.

Because null by itself does not have a type, the null coalescing operator cannot operate on it.

To correct this error

1. Cast the null literal to an object.

Example

The following code generates CS0845:

```
// cs0845.cs
using System;
using System.Linq;
using System.Linq.Expressions;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Expression<Func<object>> e = () => null ?? null; // CS0845
            // Try the following line instead.
            // Expression<Func<object>> e = () => (object)null ?? null;
        }
    }
}
```

Compiler Error CS1001

1/23/2019 • 2 minutes to read • Edit Online

Identifier expected

You did not supply an identifier. An identifier is the name of a class, struct, namespace, method, variable and so on that you provide.

The following example declares a simple class but does not give the class a name:

```
//cs1001.cs
public class              //CS1001
    {
        public int Num {get; set;}
        void MethodA(){}
    }
```

The following sample generates CS1001 because, when declaring an enum, you must specify members:

```
// CS1001.cs
public class clx
{
   enum Colors : int
   {
      'a', 'b' // CS1001, 'a' is not a valid int identifier
       // The following line shows examples of valid identifiers:
       // Blue, Red, Orange
   };

   public static void Main()
   {
   }
}
```

Parameter names are required even if the compiler doesn't use them, for example, in an interface definition. These parameters are required so that programmers who are consuming the interface know something about the what the parameters mean.

```
// CS1001-2.cs
// compile with: /target:library
interface IMyTest
{
   void TestFunc1(int, int);  // CS1001
   // Use the following line instead:
   // void TestFunc1(int a, int b);
}

class CMyTest : IMyTest
{
   void IMyTest.TestFunc1(int a, int b)
   {
   }
}
```

See also

- Statements, Expressions, and Operators
- Types

Compiler Error CS1009

1/23/2019 • 2 minutes to read • Edit Online

Unrecognized escape sequence

An unexpected character follows a backslash (\) in a string. The compiler expects one of the valid escape characters. For more information, see Character Escapes.

The following sample generates CS1009.

```
// CS1009-a.cs
class MyClass
{
   static void Main()
   {
      // The following line causes CS1009.
      string a = "\m";
      // Try the following line instead.
      // string a = "\t";
   }
}
```

A common cause of this error is using the backslash character in a file name, as the following example shows.

```
string filename = "c:\myFolder\myFile.txt";
```

To resolve this error, use "\\" or the @-quoted string literal, as the following example shows.

```
// CS1009-b.cs
class MyClass
{
   static void Main()
   {
      // The following line causes CS1009.
      string filename = "c:\myFolder\myFile.txt";
      // Try one of the following lines instead.
      // string filename = "c:\\myFolder\\myFile.txt";
      // string filename = @"c:\myFolder\myFile.txt";
   }
}
```

See also

- string

Compiler Error CS1018

5/4/2018 • 2 minutes to read • Edit Online

Keyword 'this' or 'base' expected

The compiler encountered an incomplete constructor declaration.

Example

The following example generates CS1018, and suggests several ways to resolve the error:

```
// CS1018.cs
public class C
{
}

public class a : C
{
    public a(int i)
    {
    }

    public a () :   // CS1018
    // possible resolutions:
    // public a () resolves by removing the colon
    // public a () : base() calls C's default constructor
    // public a () : this(1) calls the assignment constructor of class a
    {
    }

    public static int Main()
    {
        return 1;
    }
}
```

Compiler Error CS1019

1/23/2019 • 2 minutes to read • Edit Online

Overloadable unary operator expected

Something that looks like an overloaded unary operator has been declared, but the operator is missing or is in the wrong location in the signature.

A *unary operator* is an operator that operates on a single operand. For example, `++` is a unary operator. You can overload some unary operators by using the `operator` keyword and specifying a single parameter of the type that the operator operates on. For example, if you want to overload the operator `++` for a user-defined class `Temperature` so that you can write `Temperature++`, you can define it in this way:

```
public static  Temperature operator ++ (Temperature temp)
{
    temp.Degrees++;
    return temp;
}
```

When you receive this error, you have declared something that looks like an overloaded unary operator, except that the operator itself is missing or is in the wrong location in the signature. If you remove the `++` from the signature in the previous example, you will generate CS1019.

The following code generates CS1019:

```
// CS1019.cs
public class ii
{
   int i
   {
      get
      {
         return 0;
      }
   }
}

public class a
{
    public int i;
// Generates CS1019: "ii" is not a unary operator.
   public static a operator ii(a aa)

   // Use the following line instead:
   //public static a operator ++(a aa)
   {
      aa.i++;
      return aa;
   }

   public static void Main()
   {
   }
}
```

See also

- Operators

Compiler Error CS1026

5/15/2018 • 2 minutes to read • Edit Online

) expected

An incomplete statement was found.

A common cause of this error is placing a statement, rather than an expression, within an inline expression in an ASP.NET page. For example, the following is incorrect:

```
<%=new TimeSpan(DateTime.Now.Ticks - new DateTime(2001, 1, 1).Ticks).Days;%>
```

The following is correct:

```
<%=new TimeSpan(DateTime.Now.Ticks - new DateTime(2001, 1, 1).Ticks).Days %>
```

It is interpreted as follows:

```
<% Response.Write(new TimeSpan(DateTime.Now.Ticks - new DateTime(2001, 1, 1).Ticks).Days); %>
```

The following example generates CS1026:

```
// CS1026.cs
#if (a == b   // CS1026, add closing )
#endif

class x
{
   public static void Main()
   {
   }
}
```

Compiler Error CS1029

9/12/2018 • 2 minutes to read • Edit Online

#error: 'text'

Displays the text of an error defined with the #error directive.

The following sample shows how to create a user-defined error:

```
// CS1029.cs
class Sample
{
   static void Main()
   {
      #error Let's give an error here   // CS1029
   }
}
```

Compilation produces the following output:

```
example.cs(9,8): error CS1029: #error: 'Let's give an error here   // CS1029  '
```

Compiler Error CS1061

1/23/2019 • 2 minutes to read • Edit Online

'type' does not contain a definition for 'member' and no extension method 'name' accepting a first argument of type 'type' could be found (are you missing a using directive or an assembly reference?).

This error occurs when you try to call a method or access a class member that does not exist.

Example

The following example generates CS1061 because `TestClass1` does not have a `DisplaySomething` method. It does have a method that is called `WriteSomething`. Perhaps that is what the author of this source code meant to write.

```
// cs1061.cs
public class TestClass1
{
    // TestClass1 has one method, called WriteSomething.
    public void WriteSomething(string s)
    {
        System.Console.WriteLine(s);
    }
}

public class TestClass2
{
    // TestClass2 has one method, called DisplaySomething.
    public void DisplaySomething(string s)
    {
        System.Console.WriteLine(s);
    }
}

public class TestTheClasses
{
    public static void Main()
    {
        TestClass1 tc1 = new TestClass1();
        TestClass2 tc2 = new TestClass2();
        // The following call fails because TestClass1 does not have
        // a method called DisplaySomething.
        tc1.DisplaySomething("Hello");      // CS1061

        // To correct the error, change the method call to either
        // tc1.WriteSomething or tc2.DisplaySomething.
        tc1.WriteSomething("Hello from TestClass1");
        tc2.DisplaySomething("Hello from TestClass2");
    }
}
```

See also

- Classes and Structs
- Extension Methods

Compiler Error CS1112

5/4/2018 • 2 minutes to read • Edit Online

Do not use 'System.Runtime.CompilerServices.ExtensionAttribute'. Use the 'this' keyword instead.

This error is generated when the ExtensionAttribute is used on a non-static class that contains extension methods. If this attribute is used on a static class, another error, such as CS0708: "Cannot declare instance members in a static class," might occur.

In C#, extension methods must be defined in a static class and the first parameter of the method is modified with the `this` keyword. Do not use the attribute at all in the source code. For more information, see Extension Methods.

To correct this error

1. Remove the attribute and apply the `this` modifier to the first parameter of the method.

Example

The following example generates CS1112:

```
// cs1112.cs
[System.Runtime.CompilerServices.ExtensionAttribute] // CS1112
public class Extensions
{
    public bool A(bool b) { return b; }
}

class A { }
```

Compiler Error CS1501

5/4/2018 • 2 minutes to read • Edit Online

No overload for method 'method' takes 'number' arguments

A call was made to a class method, but no definition of the method takes the specified number of arguments.

Example

The following sample generates CS1501.

```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            ExampleClass ec = new ExampleClass();
            ec.ExampleMethod();
            ec.ExampleMethod(10);
            // The following line causes compiler error CS1501 because
            // ExampleClass does not contain an ExampleMethod that takes
            // two arguments.
            ec.ExampleMethod(10, 20);
        }
    }

    // ExampleClass contains two overloads for ExampleMethod. One of them
    // has no parameters and one has a single parameter.
    class ExampleClass
    {
        public void ExampleMethod()
        {
            Console.WriteLine("Zero parameters");
        }

        public void ExampleMethod(int i)
        {
            Console.WriteLine("One integer parameter.");
        }

        //// To fix the error, you must add a method that takes two arguments.
        //public void ExampleMethod (int i, int j)
        //{
        //    Console.WriteLine("Two integer parameters.");
        //}
    }
}
```

Compiler Error CS1502

5/4/2018 • 2 minutes to read • Edit Online

The best overloaded method match for 'name' has some invalid arguments

This error occurs when the argument types being passed to the method do not match the parameter types of that method. If the called method is overloaded, then none of the overloaded versions has a signature that matches the argument types being passed.

To resolve this problem, do one of the following:

- Double-check the types of the arguments being passed. Make sure that they match the arguments of the method being called.
- If appropriate, convert any mismatched parameters using the Convert class.
- If appropriate, cast any mismatched parameters to match the type that the method is expecting.
- If appropriate, define another overloaded version of the method to match the parameter types that are being sent.

The following sample generates CS1502:

```
// CS1502.cs
namespace x
{
   public class a
   {
      public a(char i)
      // try the following constructor instead
      // public a(int i)
      {
      }

      public static void Main()
      {
         a aa = new a(2222);   // CS1502
      }
   }
}
```

Compiler Error CS1519

1/23/2019 • 2 minutes to read • Edit Online

Invalid token 'token' in class, struct, or interface member declaration

This error is generated whenever a token is encountered in a location where it does not belong. A *token* is a keyword; an identifier (the name of a class, struct, method, and so on); a string, character, or numeric literal value such as 108, "Hello", or 'A'; or an operator or punctuator such as `==` or `;`.

Any class, struct, or interface member declaration that contains invalid modifiers before the type will generate this error. To fix the error, remove the invalid modifiers.

The following sample generates CS1519 in five places because tokens are placed in locations where they are not valid:

```
// CS1519.cs
// Generates CS1519 because a class name cannot be a number:
class Test 42
{
// Generates CS1519 because of 'j' following 'I'
// with no comma between them:
    int i j;
// Generates CS1519 because of "checked" on void method:
    checked void f4();

// Generates CS1519 because of "num":
    void f5(int a num){}

// Generates CS1519 because of namespace inside class:
    namespace;

}
```

See also

- Classes
- Structs
- Interfaces
- Methods

Compiler Error CS1540

1/23/2019 • 2 minutes to read • Edit Online

Cannot access protected member 'member' via a qualifier of type 'type1'; the qualifier must be of type 'type2' (or derived from it)

A derived class cannot access protected members of its base class through an instance of the base class. An instance of the base class declared in the derived class might, at run time, be an instance of another type that is derived from the same base but is not otherwise related to the derived class. Because protected members can be accessed only by derived types, any attempts to access protected members that might not be valid at run time are marked by the compiler as not valid.

In the `Employee` class in the following example, `emp2` and `emp3` both have type `Person` at compile time, but `emp2` has type `Manager` at run time. Because `Employee` is not derived from `Manager`, it cannot access the protected members of the base class, `Person`, through an instance of the `Manager` class. The compiler cannot determine what the run-time type of the two `Person` objects will be. Therefore, both the call from `emp2` and the call from `emp3` cause compiler error CS1540.

```
using System;
using System.Text;

namespace CS1540
{
    class Program1
    {
        static void Main()
        {
            Employee.PreparePayroll();
        }
    }

    class Person
    {
        protected virtual void CalculatePay()
        {
            Console.WriteLine("CalculatePay in Person class.");
        }
    }

    class Manager : Person
    {
        protected override void CalculatePay()
        {
            Console.WriteLine("CalculatePay in Manager class.");

        }
    }

    class Employee : Person
    {
        public static void PreparePayroll()
        {
            Employee emp1 = new Employee();
            Person emp2 = new Manager();
            Person emp3 = new Employee();
            // The following line calls the method in the Employee base class,
            // Person.
            emp1.CalculatePay();

            // The following lines cause compiler error CS1540. The compiler
            // cannot determine at compile time what the run-time types of
            // emp2 and emp3 will be.
            //emp2.CalculatePay();
            //emp3.CalculatePay();

        }
    }
}
```

See also

- Inheritance
- Polymorphism
- Access Modifiers
- Abstract and Sealed Classes and Class Members

Compiler Error CS1546

5/4/2018 • 2 minutes to read • Edit Online

Property, indexer, or event 'property' is not supported by the language; try directly calling accessor method 'accessor'

Your code is consuming an object that has a default indexed property and tried to use the indexed syntax. To resolve this error, call the property's accessor method. For more information on indexers and properties, see Indexers.

The following sample generates CS1546.

Example

This code sample consists of a .cpp file, which compiles to a .dll, and a .cs file, which uses that .dll. The following code is for the .dll file and defines a property to be accessed by the code in the .cs file.

```
// CPP1546.cpp
// compile with: /clr /LD
using namespace System;
public ref class MCPP
{
public:
    property int Prop [int,int]
    {
        int get( int i, int b )
        {
            return i;
        }
    }
};
```

Example

This is the C# file.

```
// CS1546.cs
// compile with: /r:CPP1546.dll
using System;
public class Test
{
    public static void Main()
    {
        int i = 0;
        MCPP mcpp = new MCPP();
        i = mcpp.Prop(1,1); // CS1546
        // Try the following line instead:
        // i = mcpp.get_Prop(1,1);
    }
}
```

Compiler Error CS1548

1/23/2019 • 2 minutes to read • Edit Online

Cryptographic failure while signing assembly 'assembly' — 'reason'

CS1548 occurs when assembly signing fails. This is usually due to an invalid key file name, an invalid key file path, or a corrupt key file.

To fully sign an assembly, you must provide a valid key file that contains information about the public and private keys. To delay sign an assembly, you must select the **Delay sign only** check box and provide a valid key file that contains information about the public key information. The private key is not necessary when an assembly is delay-signed.

For more information, see How to: Sign an Assembly with a Strong Name, /keyfile (C# Compiler Options) and /delaysign (C# Compiler Options).

When creating an assembly, the C# compiler calls into a utility called al.exe. If there is a failure in the assembly creation, the reason for the failure is reported by al.exe. See Al.exe Tool Errors and Warnings and search that topic for the text reported by the compiler in 'reason'.

See also

- How to: Sign an Assembly with a Strong Name

Compiler Error CS1564

1/23/2019 • 2 minutes to read • Edit Online

Conflicting options specified: Win32 resource file; Win32 manifest.

If you use the **/Win32res** compiler option, you must include the custom Win32 manifest (if it is required) in the resource file. You cannot provide a custom Win32 manifest separately from a Win32 resource file. Use the **/win32manifest** option only if you are not specifying a win32 resource file.

To correct this error

1. Add the win32 manifest to the win32 resource file and remove the **/win32manifest** compiler option.

Example

The following example produces CS1564 if it is compiled with the **/Win32res** option and no manifest is included in the resource file.

```
// cs1564.cs
// Compile with: /Win32res
public class Test
{
    static int Main(string[] args)
    {
        return 1;
    }
}
```

The C# 3.0 compiler adds a default win32Manifest to all binaries.

See also

- /win32manifest (C# Compiler Options)
- /win32res (C# Compiler Options)

Compiler Error CS1567

5/4/2018 • 2 minutes to read • Edit Online

Error generating Win32 resource: 'file'

Your compilation either used the /win32icon compiler option or did not use /win32res, which causes the compiler to generate a file that contains resource information, but the compiler was unable to create the file due to insufficient disk space or some other error.

If you are unable to resolve the file-generation problem, you could use /win32res, which does not generate a file that contains resource information.

Compiler Error CS1579

5/25/2018 • 2 minutes to read • Edit Online

foreach statement cannot operate on variables of type 'type1' because 'type2' does not contain a public definition for 'identifier'

To iterate through a collection using the foreach statement, the collection must meet the following requirements:

- Its type must include a public parameterless `GetEnumerator` method whose return type is either class, struct, or interface type.
- The return type of the `GetEnumerator` method must contain a public property named `Current` and a public parameterless method named `MoveNext` whose return type is Boolean.

Example

The following sample generates CS1579 because the `MyCollection` class doesn't contain the public `GetEnumerator` method:

```
// CS1579.cs
using System;
public class MyCollection
{
   int[] items;
   public MyCollection()
   {
      items = new int[5] {12, 44, 33, 2, 50};
   }

   // Delete the following line to resolve.
   MyEnumerator GetEnumerator()

   // Uncomment the following line to resolve:
   // public MyEnumerator GetEnumerator()
   {
      return new MyEnumerator(this);
   }

   // Declare the enumerator class:
   public class MyEnumerator
   {
      int nIndex;
      MyCollection collection;
      public MyEnumerator(MyCollection coll)
      {
         collection = coll;
         nIndex = -1;
      }

      public bool MoveNext()
      {
         nIndex++;
         return (nIndex < collection.items.Length);
      }

      public int Current => collection.items[nIndex];
   }

   public static void Main()
   {
      MyCollection col = new MyCollection();
      Console.WriteLine("Values in the collection are:");
      foreach (int i in col)   // CS1579
      {
         Console.WriteLine(i);
      }
   }
}
```

Compiler Error CS1612

1/23/2019 • 2 minutes to read • Edit Online

Cannot modify the return value of 'expression' because it is not a variable

An attempt was made to modify a value type that is produced as the result of an intermediate expression but is not stored in a variable. This error can occur when you attempt to directly modify a struct in a generic collection, as shown in the following example:

```
List<myStruct> list = {…};
list[0].Name = "MyStruct42"; //CS1612
```

To modify the struct, first assign it to a local variable, modify the variable, then assign the variable back to the item in the collection.

```
List<myStruct> list = {…};
MyStruct ms = list[0];
ms.Name = "MyStruct42";
list[0] = ms;
```

This error occurs because value types are copied on assignment. When you retrieve a value type from a property or indexer, you are getting a copy of the object, not a reference to the object itself. The copy that is returned is not stored by the property or indexer because they are actually methods, not storage locations (variables). You must store the copy into a variable that you declare before you can modify it.

The error does not occur with reference types because a property or indexer in that case returns a reference to an existing object, which is a storage location.

If you are defining the class or struct, you can resolve this error by modifying your property declaration to provide access to the members of a struct. If you are writing client code, you can resolve the error by creating your own instance of the struct, modifying its fields, and then assigning the entire struct back to the property. As a third alternative, you can change your struct to a class.

Example

CS1612 also occurs when you attempt to access the member of a struct through a property on an enclosing class that is returning the entire struct, as shown in the following example:

```
// CS1612.cs
using System;

public struct MyStruct
{
    public int Width;
}

public class ListView
{
    MyStruct ms;
    public MyStruct Size
    {
        get { return ms; }
        set { ms = value; }
    }
}

public class MyClass
{
    public MyClass()
    {
        ListView lvi;
        lvi = new ListView();
        lvi.Size.Width = 5; // CS1612

        // You can use the following lines instead.
        // MyStruct ms;
        // ms.Width = 5;
        // lvi.Size = ms;  // CS1612
    }

    public static void Main()
    {
        MyClass mc = new MyClass();
        // Keep the console open in debug mode.
        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
}
```

See also

- Structs
- Value Types
- Reference Types

Compiler Error CS1614

8/21/2018 • 2 minutes to read • Edit Online

'name' is ambiguous between 'name' and 'nameAttribute'; use either '@name' or 'nameAttribute'.

The compiler has encountered an ambiguous attribute specification.

For convenience, the C# compiler allows you to specify **ExampleAttribute** as just `[Example]`. However, ambiguity arises if an attribute class named `Example` exists along with **ExampleAttribute**, because the compiler cannot tell if `[Example]` refers to the `Example` attribute or the **ExampleAttribute** attribute. To clarify, use `[@Example]` for the `Example` attribute and `[ExampleAttribute]` for **ExampleAttribute**.

The following sample generates CS1614:

```
// CS1614.cs
using System;

// Both of the following classes are valid attributes with valid
// names (MySpecial and MySpecialAttribute). However, because the lookup
// rules for attributes involves auto-appending the 'Attribute' suffix
// to the identifier, these two attributes become ambiguous; that is,
// if you specify MySpecial, the compiler can't tell if you want
// MySpecial or MySpecialAttribute.

public class MySpecial : Attribute {
   public MySpecial() {}
}

public class MySpecialAttribute : Attribute {
   public MySpecialAttribute() {}
}

class MakeAWarning {
   [MySpecial()] // CS1614
                 // Ambiguous: MySpecial or MySpecialAttribute?
   public static void Main() {
   }

   [@MySpecial()] // This isn't ambiguous, it binds to the first attribute above.
   public static void NoWarning() {
   }

   [MySpecialAttribute()] // This isn't ambiguous, it binds to the second attribute above.
   public static void NoWarning2() {
   }

   [@MySpecialAttribute()] // This is also legal.
   public static void NoWarning3() {
   }
}
```

Compiler Error CS1640

5/4/2018 • 2 minutes to read • Edit Online

foreach statement cannot operate on variables of type 'type' because it implements multiple instantiations of 'interface', try casting to a specific interface instantiation

The type inherits from two or more instances of IEnumerator<T>, which means there is not a unique enumeration of the type that `foreach` could use. Specify the type of IEnumerator<T> or use another looping construct.

Example

The following sample generates CS1640:

```
// CS1640.cs

using System;
using System.Collections;
using System.Collections.Generic;

public class C : IEnumerable, IEnumerable<int>, IEnumerable<string>
{
    IEnumerator<int> IEnumerable<int>.GetEnumerator()
    {
        yield break;
    }

    IEnumerator<string> IEnumerable<string>.GetEnumerator()
    {
        yield break;
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return (IEnumerator)((IEnumerable<string>)this).GetEnumerator();
    }
}

public class Test
{
    public static int Main()
    {
        foreach (int i in new C()){}    // CS1640

        // Try specifying the type of IEnumerable<T>
        // foreach (int i in (IEnumerable<int>)new C()){}
        return 1;
    }
}
```

Compiler Error CS1644

5/4/2018 • 2 minutes to read • Edit Online

Feature 'feature' is not part of the standardized ISO C# language specification, and may not be accepted by other compilers

This error occurs if you specified the /langversion option ISO-1 and the code you are compiling uses features that are not part of the ISO 1.0 standard. To resolve this error, do not use any of the C# 2.0 compiler features such as generics or anonymous methods with the ISO-1 compatibility option.

The following sample generates CS1644:

```
// CS1644.cs
// compile with: /langversion:ISO-1 /target:library
class C<T> {}   // CS1644
```

Compiler Error CS1656

5/4/2018 • 2 minutes to read • Edit Online

Cannot assign to 'variable' because it is a 'read-only variable type'

This error occurs when an assignment to variable occurs in a read-only context. Read-only contexts include foreach iteration variables, using variables, and fixed variables. To resolve this error, avoid assignments to a statement variable in `using` blocks, `foreach` statements, and `fixed` statements.

Example

The following example generates error CS1656 because it tries to replace complete elements of a collection inside a `foreach` loop. One way to work around the error is to change the `foreach` loop to a for loop. Another way, not shown here, is to modify the members of the existing element; this is possible with classes, but not with structs.

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Text;

namespace CS1656_2
{

    class Book
    {
        public string Title;
        public string Author;
        public double Price;
        public Book(string t, string a, double p)
        {
            Title=t;
            Author=a;
            Price=p;

        }
    }

    class Program
    {
        private List<Book> list;
        static void Main(string[] args)
        {
            Program prog = new Program();
            prog.list = new List<Book>();
            prog.list.Add(new Book ("The C# Programming Language",
                                    "Hejlsberg, Wiltamuth, Golde",
                                     29.95));
            prog.list.Add(new Book ("The C++ Programming Language",
                                    "Stroustrup",
                                     29.95));
            prog.list.Add(new Book ("The C Programming Language",
                                    "Kernighan, Ritchie",
                                    29.95));
            foreach(Book b in prog.list)
            {
                // Cannot modify an entire element in a foreach loop
                // even with reference types.
                // Use a for or while loop instead
                if (b.Title == "The C Programming Language")
                    // Cannot assign to 'b' because it is a 'foreach
                    // iteration variable'
                    b = new Book("Programming Windows, 5th Ed.", "Petzold", 29.95); //CS1656
            }

            //With a for loop you can modify elements
            //for(int x = 0; x < prog.list.Count; x++)
            //{
            //    if(prog.list[x].Title== "The C Programming Language")
            //        prog.list[x] = new Book("Programming Windows, 5th Ed.", "Petzold", 29.95);
            //}
            //foreach(Book b in prog.list)
            //    Console.WriteLine(b.Title);

        }
    }
}
```

Example

The following sample demonstrates how CS1656 can be generated in other contexts besides a `foreach` loop:

```
// CS1656.cs
// compile with: /unsafe
using System;

class C : IDisposable
{
    public void Dispose() { }
}

class CMain
{
    unsafe public static void Main()
    {
        using (C c = new C())
        {
            // Cannot assign to 'c' because it is a 'using variable'
            c = new C(); // CS1656
        }

        int[] ary = new int[] { 1, 2, 3, 4 };
        fixed (int* p = ary)
        {
            // Cannot assign to 'p' because it is a 'fixed variable'
            p = null; // CS1656
        }
    }
}
```

Compiler Error CS1674

5/4/2018 • 2 minutes to read • Edit Online

'T': type used in a using statement must be implicitly convertible to 'System.IDisposable'

The using Statement is intended to be used to ensure the disposal of an object at the end of the `using` block, thus, only types which are disposable may be used in such a statement. For example, value types are not disposable, and type parameters which are not constrained to be classes may not be assumed to be disposable.

Example

The following sample generates CS1674.

```
// CS1674.cs
class C
{
   public static void Main()
   {
      int a = 0;
      a++;

      using (a) {}   // CS1674
   }
}
```

Example

The following sample generates CS1674.

```
// CS1674_b.cs
using System;
class C {
   public void Test() {
      using (C c = new C()) {}   // CS1674
   }
}

// OK
class D : IDisposable {
   void IDisposable.Dispose() {}
   public void Dispose() {}

   public static void Main() {
      using (D d = new D()) {}
   }
}
```

Example

The following case illustrates the need for a class type constraint to guarantee that an unknown type parameter is disposable. The following sample generates CS1674.

```
// CS1674_c.cs
// compile with: /target:library
using System;
public class C<T>
// Add a class type constraint that specifies a disposable class.
// Uncomment the following line to resolve.
// public class C<T> where T : IDisposable
{
   public void F(T t)
   {
      using (t) {}   // CS1674
   }
}
```

Compiler Error CS1703

5/4/2018 • 2 minutes to read • Edit Online

An assembly with the same simple name 'name' has already been imported. Try removing one of the references or sign them to enable side-by-side.

The compiler removes references that have the same path and file name, but it is possible that the same file exists in two places, or that you forgot to change the version number. This error points out that two references have the same assembly identity and thus the compiler has no way of distinguishing between them in metadata. Either remove one of the redundant references, or make the references unique somehow, such as by incrementing the assembly version number.

The following code generates error CS1703.

Example

This code creates assembly A in the .\bin1 directory.

Save this example in a file named CS1703a1.cs, and compile it with the following flags:

```
/t:library /out:.\bin1\cs1703.dll /keyfile:key.snk
```

```
using System;
public class A { }
```

Example

This code creates a copy of assembly A in the .\bin2 directory.

Save this example in a file named CS1703a2.cs, and compile it with the following flags:

```
/t:library /out:.\bin2\cs1703.dll /keyfile:key.snk
```

```
using System;
public class A { }
```

Example

This code references the assembly A in the two prior modules.

Save this example in a file named CS1703ref.cs, and compile it with the following flags:

```
/t:library /r:A2=.\bin2\cs1703.dll /r:A1=.\bin1\cs1703.dll
```

```
extern alias A1;
extern alias A2;
```

Compiler Error CS1704

5/4/2018 • 2 minutes to read • Edit Online

An assembly with the same simple name 'Assembly Name' has already been imported. Try removing one of the references or sign them to enable side-by-side.

This error points out that two references have the same assembly identity because the assemblies in question lack strong names, they were not signed, and thus the compiler has no way of distinguishing between them in metadata. Thus, the run time ignores the version and culture assembly name properties. The user should remove the redundant reference, rename one of the references, or provide a strong name for them.

Example

This sample creates an assembly and saves it to the root directory.

```
// CS1704_a.cs
// compile with: /target:library /out:c:\\cs1704.dll
public class A {}
```

Example

This sample creates an assembly with the same name as the previous sample, but saves it to a different location.

```
// CS1704_b.cs
// compile with: /target:library /out:cs1704.dll
public class A {}
```

Example

This sample attempts to reference both assemblies. The following sample generates CS1704.

```
// CS1704_c.cs
// compile with: /target:library /r:A2=cs1704.dll /r:A1=c:\\cs1704.dll
// CS1704 expected
extern alias A1;
extern alias A2;
```

Compiler Error CS1705

1/23/2019 • 4 minutes to read • Edit Online

Assembly 'AssemblyName1' uses 'TypeName' which has a higher version than referenced assembly 'AssemblyName2'

You are accessing a type that has a higher version number than the version number in a referenced assembly. Typically, this error is caused by the accidental use of two versions of the same assembly.

For example, suppose that you have two assemblies, Asmb1 and Asmb2. Assembly Asmb1 references version 1.0 of assembly Asmb2. Assembly Asmb1 also uses a class `MyClass` that was added to assembly Asmb2 in version 2.0. The compiler has unification rules for binding references, and a reference to `MyClass` in version 2.0 cannot be satisfied by version 1.0.

Example

The following more detailed example consists of four code modules:

- Two DLLs that are identical except for a version attribute.
- A third DLL that references the first two.
- A client that references only version 1.0 of the identical DLLs, but accesses a class from version 2.0.

The following code creates the first of the identical DLLs. For information about how to generate a key file, see /keyfile (C# Compiler Options).

```
// CS1705a.cs

// Compile by using the following command:
//      csc /target:library /out:C:\\CS1705.dll /keyfile:mykey.snk CS1705a.cs

// The DLL is created in the C:\ directory.

// The AssemblyVersion attribute specifies version 1.0 for this DLL.

[assembly:System.Reflection.AssemblyVersion("1.0")]
public class Class1
{
   public void Method1() {}
}
```

Example

The following code defines version 2.0 of the assembly, as specified by the AssemblyVersionAttribute attribute.

```
// CS1705b.cs

// Compile by using the following command:
//     csc /target:library /out:CS1705.dll /keyfile:mykey.snk CS1705b.cs

// The DLL is created in the current directory.

// The AssemblyVersion attribute specifies version 2.0 for this DLL.

[assembly:System.Reflection.AssemblyVersion("2.0")]
public class Class1
{
    public void Method1() { }
}
```

Example

The following code references the two DLL versions that are defined in the preceding code. `AssemblyA` refers to the DLL created by CS1705a.cs (version 1.0). `AssemblyB` refers to the DLL created by CS1705b.cs (version 2.0). In `ClassC`, two methods are defined. The first, `Return1A`, returns an object of type `Class1A`, which is an alias for `Class1` from version 1.0 of the DLL. The second, `Return1B`, returns an object of type `Class1B`, which is an alias for `Class1` from version 2.0 of the DLL. The definition of `Return1A` creates a dependency on version 1.0; the definition of `Return1B` creates a dependency on version 2.0.

```
// CS1705c.cs

// Compile by using the following command. AssemblyA refers to the DLL created by
// CS1705a.cs (version 1.0). AssemblyB refers to the DLL created by CS1705b.cs
// (version 2.0).
//    csc /target:library /r:AssemblyA=C:\\CS1705.dll /r:AssemblyB=CS1705.dll CS1705c.cs

extern alias AssemblyA;
extern alias AssemblyB;

// Class1A is an alias for type Class1 from VS1705a.cs, which is in version 1.0
// of the assembly. Class1B is an alias for type Class1 from CS1705b.cs, which
// is in version 2.0 of the assembly.

using Class1A = AssemblyA::Class1;
using Class1B = AssemblyB::Class1;

// Method Return1A in ClassC returns an object of type Class1A, which is
// Class1 from version 1.0 of the DLL. Method Return1B returns an object
// of type Class1B, which is Class1 from version 2.0 of the DLL.

public class ClassC
{
    // The following line creates a dependency on version 1.0 of CS1705.dll.
    // This is not the source of the problem when ClassC is accessed from
    // CS1705d.cs because CS1705d.cs references version 1.0 of the DLL.
    // Therefore, type Class1A and the assembly have the same version.
    public static Class1A Return1A() { return new Class1A(); }

    // The following line creates a dependency on version 2.0 of CS1705.dll.
    // This causes compiler error CS1705 when ClassC is accessed from
    // CS1705d.cs, because CS1705d.cs does not reference version 2.0 of the
    // DLL. Class1B is the alias for Class1 in version 2.0, and CS1705d.cs
    // references version 1.0.
    public static Class1B Return1B() { return new Class1B(); }
}
```

Example

The following code generates compiler error CS1705. It references the DLL created by CS1705a.cs (version 1.0). However, in the `Main` method, the code accesses `ClassC` from CS1705c.cs. `ClassC` uses a type that is defined in CS1705b.cs (version 2.0). This causes compiler error CS1705 because the type has a version number for CS1705.dll that is higher than the referenced version of CS1705.dll.

```
// CS1705d.cs

// Compile by using the following command:
//     csc /reference:C:\\CS1705.dll /reference:CS1705c.dll CS1705d.cs

// C:\\CS1705.dll is version 1.0 of the assembly.

class Tester
{
    static void Main()
    {
        // Return1A returns a type defined in version 1.0.
        ClassC.Return1A().Method1();
        // Return1B returns a type defined in version 2.0.
        ClassC.Return1B().Method1();
    }
}
```

You can resolve the error in one of the following ways:

- Update the code so that all files use the same version of the DLL.
- Add a reference to version 2.0 of the DLL to CS1705d.cs by using the following command to compile:

  ```
  csc /reference:C:\\CS1705.dll /reference:CS1705.dll /reference:CS1705c.dll CS1705d.cs
  ```

  Although the program compiles when you use this command, it still does not run. To enable the program to run, you can provide an application configuration file that includes a <dependentAssembly> element that uses <assemblyIdentity> and <codeBase> child elements to specify the location of version 1.0 of the DLL. For more information about configuration files, see Configuring Apps.

See also

- extern alias
- :: Operator
- Command-line Building With csc.exe

Compiler Error CS1708

5/4/2018 • 2 minutes to read • Edit Online

Fixed size buffers can only be accessed through locals or fields

A new feature in C# 2.0 is the ability to define an in-line array inside of a `struct`. Such arrays can only be accessed via local variables or fields, and may not be referenced as intermediate values on the left-hand side of an expression. Also, the arrays cannot be accessed by fields that are `static` or `readonly`.

To resolve this error, define an array variable, and assign the in-line array to the variable. Or, remove the `static` or `readonly` modifier from the field representing the in-line array.

Example

The following sample generates CS1708.

```
// CS1708.cs
// compile with: /unsafe
using System;

unsafe public struct S
{
    public fixed char name[10];
}

public unsafe class C
{
    public S UnsafeMethod()
    {
        S myS = new S();
        return myS;
    }

    static void Main()
    {
        C myC = new C();
        myC.UnsafeMethod().name[3] = 'a';  // CS1708
        // Uncomment the following 2 lines to resolve:
        // S myS = myC.UnsafeMethod();
        // myS.name[3] = 'a';

        // The field cannot be static.
        C._s1.name[3] = 'a';  // CS1708

        // The field cannot be readonly.
        myC._s2.name[3] = 'a';  // CS1708
    }

    static readonly S _s1;
    public readonly S _s2;
}
```

Compiler Error CS1716

5/4/2018 • 2 minutes to read • Edit Online

Do not use 'System.Runtime.CompilerServices.FixedBuffer' attribute. Use the 'fixed' field modifier instead.

This error arises in an unsafe code section that contains a fixed-size array declaration similar to a field declaration. Do not use this attribute. Instead, use the keyword `fixed`.

Example

The following example generates CS1716.

```
// CS1716.cs
// compile with: /unsafe
using System;
using System.Runtime.CompilerServices;

public struct UnsafeStruct
{
    [FixedBuffer(typeof(int), 4)]  // CS1716
    unsafe public int aField;
    // Use this single line instead of the above two lines.
    // unsafe public fixed int aField[4];
}

public class TestUnsafe
{
    static int Main()
    {
        UnsafeStruct us = new UnsafeStruct();
        unsafe
        {
            if (us.aField[0] == 0)
                return us.aField[1];
            else
                return us.aField[2];
        }
    }
}
```

Compiler Error CS1721

1/23/2019 • 2 minutes to read • Edit Online

Class 'class' cannot have multiple base classes: 'class_1' and 'class_2'

The most common cause of this error message is attempting to use multiple inheritance. A class in C# may only inherit directly from one class. However, a class can implement any number of interfaces.

Example

The following example shows one way in which CS1721 is generated, and then shows two possible ways to avoid the error.

```
// CS1721.cs
public class A {}
public class B {}
public class MyClass : A, B {}   // CS1721

// One possible fix is to use the following approach instead:
public class A {}
public class B : A {}
public class C : B {}

// Another possible fix is to use interfaces instead of base classes:
public class A {}
public interface B {}
public class C : A, B {}
```

See also

- Polymorphism
- Interfaces

Compiler Error CS1726

1/23/2019 • 2 minutes to read • Edit Online

Friend assembly reference 'reference' is invalid. Strong-name signed assemblies must specify a public key in their InternalsVisibleTo declarations.

A strong name signed assembly can only grant friend assembly access, made with the InternalsVisibleToAttribute, to other strongly signed assemblies.

To resolve CS1726, either sign (give a strong name to) the assembly to which you want to grant friend access, or don't grant friend access.

For more information, see Friend Assemblies.

Example

The following sample generates CS1726.

```
// Save this code as CS1726.cs

// Run the following command to create CS1726.key:
//      sn -k CS1726.key

// Then compile by using the following command:
//      csc /keyfile:CS1726.key /target:library CS1726.cs

using System.Runtime.CompilerServices;

// The following line causes compiler error CS1726.
[assembly: InternalsVisibleTo("UnsignedAssembly")]

// To get rid of the error, try the following line instead.
//[assembly: InternalsVisibleTo("SignedAssembly,
PublicKey=0024000004800000940000000602000000240000525341310004000001000100031d7b6f3abc16c7de526fd67ec2926fe68ed
2f9901afbc5f1b6b428bf6cd9086021a0b38b76bc340dc6ab27b65e4a593fa0e60689ac98dd71a12248ca025751d135df7b98c5f9d09172
f7b62dabdd302b2a1ae688731ff3fc7a6ab9e8cf39fb73c60667e1b071ef7da5838dc009ae0119a9cbff2c581fc0f2d966b77114b2c4")]


class A { }
```

See also

- How to: Create Signed Friend Assemblies

Compiler Error CS1729

1/23/2019 • 2 minutes to read • Edit Online

'type' does not contain a constructor that takes 'number' arguments.

This error occurs when you either directly or indirectly invoke the constructor of a class but the compiler cannot find any constructors with the same number of parameters. In the following example, the `test` class has no constructors that take any arguments. It therefore has only a default constructor that takes zero arguments. Because in the second line in which the error is generated, the derived class declares no constructors of its own, the compiler provides a default constructor. That constructor invokes a parameterless constructor in the base class. Because the base class has no such constructor, CS1729 is generated.

To correct this error

1. Adjust the number of parameters in the call to the constructor.
2. Modify the class to provide a constructor with the parameters you must call.
3. Provide a parameterless constructor in the base class.

Example

The following example generates CS1729:

```
// cs1729.cs
class Test
{
    static int Main()
    {
        // Class Test has only a default constructor, which takes no arguments.
        Test test1 = new Test(2); // CS1729
        // The following line resolves the error.
        Test test2 = new Test();

        // Class Parent has only one constructor, which takes two int parameters.
        Parent exampleParent1 = new Parent(10); // CS1729
        // The following line resolves the error.
        Parent exampleParent2 = new Parent(10, 1);

        return 1;
    }
}

public class Parent
{
    // The only constructor for this class has two parameters.
    public Parent(int i, int j) { }
}

// The following declaration causes a compiler error because class Parent
// does not have a constructor that takes no arguments. The declaration of
// class Child2 shows how to resolve this error.
public class Child : Parent { } // CS1729

public class Child2 : Parent
{
    // The constructor for Child2 has only one parameter. To access the
    // constructor in Parent, and prevent this compiler error, you must provide
    // a value for the second parameter of Parent. The following example provides 0.
    public Child2(int k)
        : base(k, 0)
    {
        // Add the body of the constructor here.
    }
}
```

See also

- Inheritance
- Constructors

Compiler Error CS1919

1/23/2019 • 2 minutes to read • Edit Online

Unsafe type 'type name' cannot be used in object creation.

The `new` operator creates objects only on the managed heap. However, you can create objects in unmanaged memory indirectly by using the interoperability capabilities of the language to call native methods that return pointers.

To correct this error

1. Use a safe type in the new object creation expression. For example, use `char` or `int` instead of `char*` or `int*`.
2. If you must create objects in unmanaged memory, use a Win32 or COM method or else write your own function in C or C++ and call it from C#.

Example

The following example generates CS1919 because a pointer type is unsafe:

```
// cs1919.cs
// Compile with: /unsafe
unsafe public class C
{
    public static int Main()
    {
        var col1 = new int* { }; // CS1919
        var col2 = new char* { }; // CS1919
        return 1;
    }
}
```

See also

- Interoperability

Compiler Error CS1921

1/23/2019 • 2 minutes to read • Edit Online

The best overloaded method match for 'method' has wrong signature for the initializer element. The initializable Add must be an accessible instance method.

This error is generated when you try to use a collection initializer with a class that has no public non-static `Add` method. If the `Add` method is not accessible because of its protection level (`private` , `protected` , `internal`) then you will get CS0122, so that this error probably means that the method is defined as `static` .

Example

The following example generates CS1921:

```
// cs1921.cs
using System.Collections;
public class C : CollectionBase
{
    public static void Add(int i)
    {
    }
}
public class Test
{
    public static void Main()
    {
        var collection = new C { 1, 2, 3 }; // CS1921
    }
}
```

See also

- Object and Collection Initializers

Compiler Error CS1926

1/23/2019 • 2 minutes to read • Edit Online

Error reading Win32 manifest file 'filename' -- 'error'.

This error is generated when the following conditions are true:

1. The **/win32manifest** option is specified either on the command line or by right-clicking the **Project** icon in **Solution Explorer**, pointing to **Add**, clicking **New Item**, and then clicking **Application Manifest File**.

2. The file is either corrupted or missing.

To correct this error

1. Remove the option.

2. Replace, repair, or regenerate the file.

Example

The following example generates CS1926 when it is compiled with a corrupted for missing win32 manifest file:

```
// cs1926.cs
// Compile with: /win32manifest: ../../app.manifest
// CS1926
class Test
{
    public static int Main()
    {
        return 1;
    }
}
```

See also

- /win32manifest (C# Compiler Options)

Compiler Error CS1933

1/23/2019 • 2 minutes to read • Edit Online

Expression cannot contain query expressions

Some variables cannot be initialized with a query expression. Constants cannot be initialized with query expressions because constants may only be initialized with some combination of literals, named constants, and mathematical operators.

To correct this error

1. Remove the modifier from the query variable.

Example

The following example generates CS1933:

```
// cs1933.cs
using System.Linq;
using System.Collections;

class P
{
    const IEnumerable e = from x in new[] { 1, 2, 3 } select x; // CS1933
    static int Main()
    {
        return 1;
    }
}
```

See also

- LINQ Query Expressions

Compiler Error CS1936

1/23/2019 • 2 minutes to read • Edit Online

Could not find an implementation of the query pattern for source type 'type'. 'method' not found.

In order to query a source type, that type must implement the standard query operator methods that you are invoking in the query. The implementation can be either in the form of class members or extension methods that are brought into scope with the appropriate `using` directive.

To correct this error

- Make sure that you are querying a collection of objects, not an individual object.
- Make sure that you have specified the necessary `using` directives.

Example

The following example produces CS1936:

```
// cs1936.cs
using System.Collections;
using System.Linq;
class Test
{
    static int Main()
    {
        object obj;
        IEnumerable e = from x in obj // CS1936
                        select x;
        return 0;
    }
}
```

This error typically occurs when you accidentally try to query an object of some type instead of a collection of those objects.

See also

- Standard Query Operators Overview

Compiler Error CS1941

1/23/2019 • 2 minutes to read • Edit Online

The type of one of the expressions in the 'clause' clause is incorrect. Type inference failed in the call to 'method'.

Type inference in query expressions flows from the type of the elements in the data source(s).

To correct this error

1. If it is not immediately obvious why the error is occurring, examine the query carefully and trace the type of the result of each clause from the data source to the point where the error is occurring.

Example

The following code generates CS1941 because the `equals` operator is being asked to compare an `int` to a `string`.

```
// cs1941.cs
using System.Collections;
using System.Linq;
class Test
{
    static int Main()
    {
        var nums = new[] { 1, 2, 3, 4, 5, 6 };
        var words = new string[] { "lake", "mountain", "sky" };
        IEnumerable e = from n in nums
                        join w in words on n equals w // CS1941
                        select w;
        return 0;
    }
}
```

The method in which type inference fails is the method that the query clause is translated to at compile time.

See also

- LINQ Query Expressions
- Type Relationships in LINQ Query Operations

Compiler Error CS1942

1/23/2019 • 2 minutes to read • Edit Online

The type of the expression in the 'clause' clause is incorrect. Type inference failed in the call to 'method'.

This error is typically generated when the range variable has been given an incorrect explicit type.

To correct this error

1. If the range variable is explicitly typed, make sure that the type is either the same as, or implicitly convertible from, the type of the elements in the collection it iterates. If the range variable is preceded with the `var` keyword, remove `var`.

Example

The following code generates CS1942:

```
// cs1942.cs
class Program
    {
        static void Main(string[] args)
        {
            var x = from var i in Enumerable.Range(1, 100) // CS1949
                    select i; //CS1942
        }
    }
```

CS1942 is related to CS1949 because the use of `var` with a range variable causes the underlying `Cast<T>` operation to fail because `var` is not a type.

See also

- var
- LINQ Query Expressions

Compiler Error CS1943

5/4/2018 • 2 minutes to read • Edit Online

An expression of type 'type' is not allowed in a subsequent from clause in a query expression with source type 'type'. Type inference failed in the call to 'method'.

All range variables must represent queryable types.

To correct this error

1. Make sure that the type is a queryable type that implements `IEnumerable`, `IEnumerable<T>` or a derived interface, or any other type which has a query pattern defined for it.
2. If the type is a non-generic `IEnumerable`, provide an explicit type on the range variable.

Example

The following code generates CS1943:

```
// cs1943.cs
using System.Linq;
class Test
{
    class TestClass
    { }
    static void Main()
    {
        int[] nums = { 0, 1, 2, 3, 4, 5 };
        TestClass tc = new TestClass();

        var x = from n in nums
                from s in tc // CS1943
                select n + s;
    }
}
```

Compiler Error CS1946

1/23/2019 • 2 minutes to read • Edit Online

An anonymous method expression cannot be converted to an expression tree.

An anonymous method represents a set of statements but an expression tree must not contain a statement. Therefore an anonymous method cannot be represented by an expression tree.

To correct this error

1. Change the anonymous method to a lambda expression.

Example

The following example generates CS1946:

```
// cs1946.cs
using System;
    using System.Linq.Expressions;

    public delegate void D();

    class Test
    {
        static void Main()
        {
            Expression<D> tree = delegate() { }; //CS1946
            // Try using a lambda expression instead.
            // Expression<D> tree = (x) => x + 1;
        }
    }
```

See also

- Anonymous Methods
- Expression Trees

Compiler Error CS2032

5/4/2018 • 2 minutes to read • Edit Online

Character 'character' is not allowed on the command-line or in response files

Response files and the command line options for csc.exe cannot contain ASCII control characters in the range 0-31 or the pipe (`|`) character.

Compiler error CS2032 is difficult to demonstrate from the command line because the command line processor and the integrated development environment (IDE) filter out characters that are not valid. The following procedure generates the error by using a response file.

To generate this error

1. In the **My Documents** folder, create a text file that is named CS2032.rsp, and then enter the following compiler options in it:

   ```
   /target:exe /out:cs|2032.exe cs2032.cs
   ```

2. In the **My Documents** folder, create a text file that is named cs2032.cs and that contains whatever you want.
3. Open the **Start** menu. Locate and select the Developer Command Prompt for Visual Studio.
4. Change the current directory to `C:\Users\\<YourUsername>\Documents`.
5. Run the following command from the command prompt: `csc @cs2032.rsp`
6. The CS2032 error message appears because the response file, CS2032.rsp, contains a pipe character.

Compiler Error CS7003

1/23/2019 • 2 minutes to read • Edit Online

Unexpected use of an unbound generic name

This error occurs if you use a generic type needing one parameter type without passing any generic parameter type name between the angle brackets. This use may be a variable declaration, or an object instanciation.

To correct this error

Provide one parameter type name in angle brackets when using a generic type.

Example

The following example generates CS7003:

```
// CS7003.cs
    class Program
    {
        static void Main(string[] args)
        {
            var myVar1 = new MyGenericClass<>();  //CS7003

            MyGenericClass<> var2;                //CS7003
        }
    }

    public class MyGenericClass<T> { }
```

See also

- Generics

Compiler Warning (level 1) CS0420

5/4/2018 • 2 minutes to read • Edit Online

'identifier': a reference to a volatile field will not be treated as volatile

A volatile field should not normally be passed using a `ref` or **out** parameter, since it will not be treated as volatile within the scope of the function. There are exceptions to this, such as when calling an interlocked API. As with any warning, you may use the #pragma warning to disable this warning in those rare cases where you are intentionally using a volatile field as a reference parameter.

The following sample generates CS0420:

```
// CS0420.cs
// compile with: /W:1
using System;

class TestClass
{
   private volatile int i;

   public void TestVolatile(ref int ii)
   {
   }

   public static void Main()
   {
      TestClass x = new TestClass();
      x.TestVolatile(ref x.i);   // CS0420
   }
}
```

Compiler Warning (level 1) CS0465

5/4/2018 • 2 minutes to read • Edit Online

Introducing a 'Finalize' method can interfere with destructor invocation. Did you intend to declare a destructor?

This warning occurs when you create a class with a method whose signature is `public virtual void Finalize`.

If such a class is used as a base class and if the deriving class defines a destructor, the destructor will override the base class `Finalize` method, not Finalize.

Example

The following sample generates CS0465.

```
// CS0465.cs
// compile with: /target:library
class A
{
   public virtual void Finalize() {}   // CS0465
}

// OK
class B
{
   ~B() {}
}
```

Compiler Warning (level 1) CS1058

5/4/2018 • 2 minutes to read • Edit Online

A previous catch clause already catches all exceptions. All exceptions thrown will be wrapped in a System.Runtime.CompilerServices.RuntimeWrappedException

This attribute causes CS1058 if a `catch()` block has no specified exception type after a `catch (System.Exception e)` block. The warning advises that the `catch()` block will not catch any exceptions.

A `catch()` block after a `catch (System.Exception e)` block can catch non-CLS exceptions if the `RuntimeCompatibilityAttribute` is set to false in the AssemblyInfo.cs file: `[assembly: RuntimeCompatibilityAttribute(WrapNonExceptionThrows = false)]`. If this attribute is not set explicitly to false, all thrown non-CLS exceptions are wrapped as Exceptions and the `catch (System.Exception e)` block catches them. For more information, see How to: Catch a non-CLS Exception.

Example

The following example generates CS1058.

```
// CS1058.cs
// CS1058 expected
using System.Runtime.CompilerServices;

// the following attribute is set to true by default in the C# compiler
// set to false in your source code to resolve CS1058
[assembly: RuntimeCompatibilityAttribute(WrapNonExceptionThrows = true)]

class TestClass
{
   static void Main()
   {
      try {}

      catch (System.Exception e) {
         System. Console.WriteLine("Caught exception {0}", e);
      }

      catch {}   // CS1058. This line will never be reached.
   }
}
```

Compiler Warning (level 1) CS1060

1/23/2019 • 2 minutes to read • Edit Online

Use of possibly unassigned field 'name'. Struct instance variables are initially unassigned if struct is unassigned.

Struct members are initialized to their default value if you do not explicitly initialize them. The default value for class types (and other reference types) is null. If the class is not initialized before any attempt to access it, a `NullReferenceException` will be thrown at runtime. The compiler cannot determine definitively whether the class member will be initialized or not, and so CS1060 is a warning and not an error.

To correct this error

1. Provide a constructor for the `struct` that initializes all its members.

Example

The following code generates CS1060 because the class type `U` is a member of the `struct S` but is never initialized.

```
// cs1060.cs
namespace CS1060
{
    public class U
    {
        public int i;
    }

    public struct S
    {
        public U u;
        // Add constructor to correct the error.
        //public S(int val)
        //{
        //    u = new U() { i = val };
        //}
    }
    public class Test
    {
        static void Main()
        {
            S s;
            s.u.i = 5;  // CS1060

            //Try these lines instead, and uncomment the constructor in S
            // S s = new S(0);
            // s.u.i = 5;

        }
    }
}
```

See also

- Structs

Compiler Warning (level 1) CS1598

5/4/2018 • 2 minutes to read • Edit Online

XML parser could not be loaded for the following reason: 'reason'. The XML documentation file 'file' will not be generated.

The /doc option was specified, but the compiler could not find and load msxml3.dll. Make sure that the file msxml3.dll is installed and registered.

Compiler Warning (level 1) CS1607

5/4/2018 • 2 minutes to read • Edit Online

Assembly generation -- reason

A warning was generated from the assembly-creation phase of the compilation.

If you are building a 64-bit application on a 32-bit operating system, you must ensure that 64-bit versions of all referenced assemblies are installed on the target operating system.

All x86-specific common language runtime (CLR) assemblies have 64-bit counterparts (every CLR assembly will exist on all operating systems). Therefore, you can safely ignore CS1607 for CLR assemblies.

You can ignore this warning if you encounter it when you create a AssemblyInformationalVersionAttribute. The informational version is a string that attaches additional version information to an assembly; this information is not used at run time. Although you can specify any text, a warning message appears on compilation if the string is not in the format that is used by the assembly version number, or if it is in that format but contains wildcard characters. This warning is harmless.

For more information, see Al.exe Tool Errors and Warnings.

Compiler Warning (level 1) CS1616

5/4/2018 • 2 minutes to read • Edit Online

Option 'option' overrides attribute 'attribute' given in a source file or added module

This warning occurs if the assembly attributes AssemblyKeyFileAttribute or AssemblyKeyNameAttribute found in source conflict with the /keyfile or /keycontainer command line option or key file name or key container specified in the Project Properties.

For the example below, assume you have a key file named `cs1616.snk`. Generate this file with the command line:

```
sn -k CS1616.snk
```

The following sample generates CS1616:

```
// CS1616.cs
// compile with: /keyfile:cs1616.snk
using System.Reflection;

// To fix the error, remove the next line
[assembly: AssemblyKeyFile("cs1616b.snk")]  // CS1616

class C
{
  public static void Main()
  {
  }
}
```

Compiler Warning (level 1) CS1658

5/4/2018 • 2 minutes to read • Edit Online

'warning text'. See also error 'error code'

The compiler emits this warning when it overrides an error with a warning. For information about the problem, refer to the error mentioned. To find the appropriate error from within the Visual Studio IDE, use the index. For example, if the text above reads "See also error 'CS1037'," look for CS1037 in the index.

Example

The following example generates CS1658.

```
// CS1658.cs
// compile with: /doc:x.xml
// CS1584 expected
/// <summary>
/// </summary>
public class C
{
    /// <see cref="C.F(params object[])"/>  // CS1658
    public static void M()
    {
    }

    /// <summary>
    /// </summary>
    public void F(params object[] o)
    {
    }

    static void Main()
    {
    }
}
```

Compiler Warning (level 1) CS1683

5/4/2018 • 2 minutes to read • Edit Online

Reference to type 'Type Name' claims it is defined in this assembly, but it is not defined in source or any added modules

This error can occur when you are importing an assembly that contains a reference back to the assembly you are currently compiling, but the assembly being compiled contains nothing matching the reference. One way to get to this situation is to compile your assembly, which initially does contain the member that the assembly being imported is referencing. Then you update your assembly, mistakenly removing the members that the imported assembly is referencing.

Compiler Warning (level 1) CS1685

5/4/2018 • 2 minutes to read • Edit Online

The predefined type 'System.type name' is defined in multiple assemblies in the global alias; using definition from 'File Name'

This error occurs when a predefined system type such as System.int32 is found in two assemblies. One way this can happen is if you are referencing mscorlib from two different places, such as trying to run the.Net Framework versions 1.0 and 1.1 side-by-side.

The compiler will use the definition from only one of the assemblies. The compiler searches only global aliases, does not search libraries defined **/reference**. If you have specified **/nostdlib**, the compiler will search for Object, and in the future start all searches for predefined types in the file where it found Object.

Compiler Warning (level 1) CS1690

5/4/2018 • 2 minutes to read • Edit Online

Accessing a member on 'member' may cause a runtime exception because it is a field of a marshal-by-reference class

This warning occurs when you try to call a method, property, or indexer on a member of a class that derives from MarshalByRefObject, and the member is a value type. Objects that inherit from `MarshalByRefObject` are typically intended to be marshaled by reference across an application domain. If any code ever attempts to directly access the value-type member of such an object across an application domain, a runtime exception will occur. To resolve the warning, first copy the member into a local variable and call the method on that variable.

The following sample generates CS1690:

```
// CS1690.cs
using System;

class WarningCS1690: MarshalByRefObject
{
   int i = 5;

   public static void Main()
   {
     WarningCS1690 e = new WarningCS1690();
     e.i.ToString();   // CS1690

     // OK
     int i = e.i;
     i.ToString();
     e.i = i;
   }
}
```

Compiler Warning (level 1) CS1691

5/4/2018 • 2 minutes to read • Edit Online

'number' is not a valid warning number

A number that was passed to the #pragma warning preprocessor directive was not a valid warning number. Verify that the number represents a warning, not an error or another sequence of characters.

Example

The following example generates CS1691.

```
// CS1691.cs
public class C
{
    int i = 1;
    public static void Main()
    {
        C myC = new C();
#pragma warning disable 151  // CS1691
// Try the following line instead:
// #pragma warning disable 1645
        myC.i++;
#pragma warning restore 151  // CS1691
// Try the following line instead:
// #pragma warning restore 1645
    }
}
```

Compiler Warning (level 1) CS1699

1/23/2019 • 2 minutes to read • Edit Online

Use command line option "compiler_option" or appropriate project settings instead of "attribute_name"

In order to sign an assembly, it is necessary to specify a key file. Prior to Microsoft Visual C# 2005, you specified the key file using CLR attributes in source code. These attributes are now deprecated.

Beginning in Microsoft Visual C# 2005, you should use the **Signing Page** of the **Project Designer** or the Assembly Linker to specify the key file.

The **Signing Page** of the **Project Designer** is the preferred method; for more information, see Signing Page, Project Designer and Managing Assembly and Manifest Signing.

The How to: Sign an Assembly with a Strong Name uses the following compiler options:

- /keyfile (C# Compiler Options) instead of the AssemblyKeyFileAttribute attribute.
- /keycontainer (C# Compiler Options) instead of AssemblyKeyNameAttribute.
- /delaysign (C# Compiler Options) instead of AssemblyDelaySignAttribute.

These attributes have been deprecated for the following reasons:

- There were security issues due to the attributes being embedded in the binary files produced by the compiler. Everyone who had your binary also had the keys stored in it.
- There were usability issues due to the fact that the path specified in the attributes was relative to the current working directory, which could change in the integrated development environment (IDE), or to the output directory. Thus, most times the key file is likely to be ..\\..\\mykey.snk. Attributes also make it more difficult for the project system to properly sign satellite assemblies. When you use the compiler options instead of these attributes, you can use a fully qualified path and file name for the key without anything being embedded in the output file; the project system and source code control system can properly manipulate that full path when projects are moved around; the project system can maintain a project-relative path to the key file, and still pass a full path to the compiler; other build programs can more easily sign outputs by passing the proper path directly to the compiler instead of generating a source file with the correct attributes.
- Using attributes with friend assemblies can hamper compiler efficiency. When you use attributes, the compiler does not know what the key is when it has to decide whether or not to grant friendship and so it has to guess. At the end of compilation, the compiler is able to verify the guess once it finally knows the key. When the key file is specified with a compiler option, the compiler can immediately decide whether to grant friendship.

Example

The following sample generates CS1699. To resolve the error, remove the attribute and compile with **/delaysign**.

```
// CS1699.cs
// compile with: /target:library
[assembly:System.Reflection.AssemblyDelaySign(true)]   // CS1699
```

See also

- Signing Page, Project Designer
- Managing Assembly and Manifest Signing
- How to: Sign an Assembly with a Strong Name

Compiler Warning (level 1) CS1762

1/23/2019 • 2 minutes to read • Edit Online

A reference was created to embedded interop assembly '<assembly1>' because of an indirect reference to that assembly from assembly '<assembly2>'. Consider changing the 'Embed Interop Types' property on either assembly.

You have added a reference to an assembly (assembly1) that has the `Embed Interop Types` property set to `True`. This instructs the compiler to embed interop type information from that assembly. However, the compiler cannot embed interop type information from that assembly because another assembly that you have referenced (assembly2) also references that assembly (assembly1) and has the `Embed Interop Types` property set to `False`.

> **NOTE**
> Setting the `Embed Interop Types` property on an assembly reference to `True` is equivalent to referencing the assembly by using the `/link` option for the command-line compiler.

To address this warning

- To embed interop type information for both assemblies, set the `Embed Interop Types` property on all references to assembly1 to `True`. For more information about how to set that property, see Walkthrough: Embedding Types from Managed Assemblies.
- To remove the warning, you can set the `Embed Interop Types` property of assembly1 to `False`. In this case, a primary interop assembly (PIA) provides interop type information.

See also

- /link (C# Compiler Options)
- Interoperating with Unmanaged Code

Compiler Warning (level 1) CS1956

5/4/2018 • 2 minutes to read • Edit Online

Member 'name' implements interface member 'name' in type 'type'. There are multiple matches for the interface member at run-time. It is implementation dependent which method will be called.

This warning can be generated when two interface methods are differentiated only by whether a particular parameter is marked with `ref` or with `out`. It is best to change your code to avoid this warning because it is not obvious or guaranteed which method is called at runtime.

Although C# distinguishes between `out` and `ref`, the CLR sees them as the same. When deciding which method implements the interface, the CLR just picks one.

To avoid this warning

1. Give the compiler some way to differentiate the methods. For example, you can give them different names or provide an additional parameter on one of them.

Example

The following code generates CS1956 because the two `Test` methods in `Base` differ only by the `ref` / `out` modifier on the first parameter.

```
// cs1956.cs
class Base<T, S>
{
    // This is the method that should be called.
    public virtual int Test(out T x) // CS1956
    {
        x = default(T);
        return 0;
    }

    // This is the "last" method and is the one that ends up being called
    public virtual int Test(ref S x)
    {
        return 1;
    }
}

interface IFace
{
    int Test(out int x);
}

class Derived : Base<int, int>, IFace
{
    static int Main()
    {
        IFace x = new Derived();
        int y;
        return x.Test(out y);
    }
}
```

Compiler Warning (level 1) CS3003

5/4/2018 • 2 minutes to read • Edit Online

Type of 'variable' is not CLS-compliant

A public, protected, or `protected internal` variable must be of a type that is compliant with the Common Language Specification (CLS). For more information on CLS Compliance, see Language Independence and Language-Independent Components.

Example

The following example generates CS3003:

```
// CS3003.cs

[assembly:System.CLSCompliant(true)]
public class a
{
    public ushort a1;   // CS3003, public variable
    public static void Main()
    {
    }
}
```

Compiler Warning (level 1) CS3007

5/4/2018 • 2 minutes to read • Edit Online

Overloaded method 'method' differing only by unnamed array types is not CLS-compliant

This error occurs if you have an overloaded method that takes a jagged array and the only difference between the method signatures is the element type of the array. To avoid this error, consider using a rectangular array rather than a jagged array; use an additional parameter to disambiguate the function call; rename one or more of the overloaded methods; or, if CLS Compliance is not needed, remove the CLSCompliantAttribute attribute. For more information on CLS Compliance, see Language Independence and Language-Independent Components.

Example

The following example generates CS3007:

```
// CS3007.cs
[assembly: System.CLSCompliant(true)]
public struct S
{
    public void F(int[][] array) { }
    public void F(byte[][] array) { }  // CS3007
    // Try this instead:
    // public void F1(int[][] array) {}
    // public void F2(byte[][] array) {}
    // or
    // public void F(int[,] array) {}
    // public void F(byte[,] array) {}
}
```

Compiler Warning (level 1) CS3009

5/4/2018 • 2 minutes to read • Edit Online

'type': base type 'type' is not CLS-compliant

A base type was marked as not having to be compliant with the Common Language Specification (CLS) in an assembly that was marked as being CLS compliant. Either remove the attribute that specifies the assembly is CLS compliant or remove the attribute that indicates the type is not CLS compliant. For more information on CLS Compliance, see CLS compliance rules and Language Independence and Language-Independent Components.

Example

The following example generates CS3009:

```
// CS3009.cs

using System;

[assembly:CLSCompliant(true)]
[CLSCompliant(false)]
public class B
{
}

public class C : B   // CS3009
{
    public static void Main () {}
}
```

Compiler Warning (level 1) CS4014

1/23/2019 • 6 minutes to read • Edit Online

Because this call is not awaited, execution of the current method continues before the call is completed. Consider applying the 'await' operator to the result of the call.

The current method calls an async method that returns a Task or a Task<TResult> and doesn't apply the await operator to the result. The call to the async method starts an asynchronous task. However, because no `await` operator is applied, the program continues without waiting for the task to complete. In most cases, that behavior isn't what you expect. Usually other aspects of the calling method depend on the results of the call or, minimally, the called method is expected to complete before you return from the method that contains the call.

An equally important issue is what happens to exceptions that are raised in the called async method. An exception that's raised in a method that returns a Task or Task<TResult> is stored in the returned task. If you don't await the task or explicitly check for exceptions, the exception is lost. If you await the task, its exception is rethrown.

As a best practice, you should always await the call.

You should consider suppressing the warning only if you're sure that you don't want to wait for the asynchronous call to complete and that the called method won't raise any exceptions. In that case, you can suppress the warning by assigning the task result of the call to a variable.

The following example shows how to cause the warning, how to suppress it, and how to await the call.

```
async Task CallingMethodAsync()
{
    resultsTextBox.Text += "\r\n  Entering calling method.";
    // Variable delay is used to slow down the called method so that you can
    // distinguish between awaiting and not awaiting in the program's output.
    // You can adjust the value to produce the output that this topic shows
    // after the code.
    var delay = 5000;

    // Call #1.
    // Call an async method. Because you don't await it, its completion
    // isn't coordinated with the current method, CallingMethodAsync.
    // The following line causes warning CS4014.
    CalledMethodAsync(delay);

    // Call #2.
    // To suppress the warning without awaiting, you can assign the
    // returned task to a variable. The assignment doesn't change how
    // the program runs. However, recommended practice is always to
    // await a call to an async method.

    // Replace Call #1 with the following line.
    //Task delayTask = CalledMethodAsync(delay);

    // Call #3
    // To contrast with an awaited call, replace the unawaited call
    // (Call #1 or Call #2) with the following awaited call. Best
    // practice is to await the call.

    //await CalledMethodAsync(delay);

    // If the call to CalledMethodAsync isn't awaited, CallingMethodAsync
    // continues to run and, in this example, finishes its work and returns
    // to its caller.
    resultsTextBox.Text += "\r\n  Returning from calling method.";
}

async Task CalledMethodAsync(int howLong)
{
    resultsTextBox.Text +=
        "\r\n    Entering called method, starting and awaiting Task.Delay.";

    // Slow the process down a little so that you can distinguish between
    // awaiting and not awaiting in the program's output. Adjust the value
    // for howLong if necessary.
    await Task.Delay(howLong);
    resultsTextBox.Text +=
        "\r\n    Task.Delay is finished--returning from called method.";
}
```

In the example, if you choose Call #1 or Call #2, the unawaited async method (`CalledMethodAsync`) finishes after both its caller (`CallingMethodAsync`) and the caller's caller (`startButton_Click`) are complete. The last line in the following output shows you when the called method finishes. Entry to and exit from the event handler that calls `CallingMethodAsync` in the full example are marked in the output.

```
Entering the Click event handler.
  Entering calling method.
    Entering called method, starting and awaiting Task.Delay.
  Returning from calling method.
Exiting the Click event handler.
    Task.Delay is finished--returning from called method.
```

You can also suppress compiler warnings by using #pragma warning directives.

Example

The following Windows Presentation Foundation (WPF) application contains the methods from the previous example. The following steps set up the application.

1. Create a WPF application, and name it `AsyncWarning`.

2. In the Visual Studio Code Editor, choose the **MainWindow.xaml** tab.

   If the tab isn't visible, open the shortcut menu for MainWindow.xaml in **Solution Explorer**, and then choose **View Code**.

3. Replace the code in the **XAML** view of MainWindow.xaml with the following code.

```
<Window x:Class="AsyncWarning.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="350" Width="525">
    <Grid>
        <Button x:Name="startButton" Content="Start" HorizontalAlignment="Left" Margin="214,28,0,0"
VerticalAlignment="Top" Width="75" HorizontalContentAlignment="Center" FontWeight="Bold"
FontFamily="Aharoni" Click="startButton_Click" />
        <TextBox x:Name="resultsTextBox" Margin="0,80,0,0" TextWrapping="Wrap" FontFamily="Lucida
Console"/>
    </Grid>
</Window>
```

   A simple window that contains a button and a text box appears in the **Design** view of MainWindow.xaml.

   For more information about the XAML Designer, see Creating a UI by using XAML Designer. For information about how to build your own simple UI, see the "To create a WPF application" and "To design a simple WPF MainWindow" sections of Walkthrough: Accessing the Web by Using Async and Await.

4. Replace the code in MainWindow.xaml.cs with the following code.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace AsyncWarning
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private async void startButton_Click(object sender, RoutedEventArgs e)
        {
            resultsTextBox.Text += "\r\nEntering the Click event handler.";
            await CallingMethodAsync();
            resultsTextBox.Text += "\r\nExiting the Click event handler.";
```

```
    }

    async Task CallingMethodAsync()
    {
        resultsTextBox.Text += "\r\n  Entering calling method.";
        // Variable delay is used to slow down the called method so that you can
        // distinguish between awaiting and not awaiting in the program's output.
        // You can adjust the value to produce the output that this topic shows
        // after the code.
        var delay = 5000;

        // Call #1.
        // Call an async method. Because you don't await it, its completion
        // isn't coordinated with the current method, CallingMethodAsync.
        // The following line causes warning CS4014.
        CalledMethodAsync(delay);

        // Call #2.
        // To suppress the warning without awaiting, you can assign the
        // returned task to a variable. The assignment doesn't change how
        // the program runs. However, recommended practice is always to
        // await a call to an async method.

        // Replace Call #1 with the following line.
        //Task delayTask = CalledMethodAsync(delay);

        // Call #3
        // To contrast with an awaited call, replace the unawaited call
        // (Call #1 or Call #2) with the following awaited call. Best
        // practice is to await the call.

        //await CalledMethodAsync(delay);

        // If the call to CalledMethodAsync isn't awaited, CallingMethodAsync
        // continues to run and, in this example, finishes its work and returns
        // to its caller.
        resultsTextBox.Text += "\r\n  Returning from calling method.";
    }

    async Task CalledMethodAsync(int howLong)
    {
        resultsTextBox.Text +=
            "\r\n    Entering called method, starting and awaiting Task.Delay.";

        // Slow the process down a little so that you can distinguish between
        // awaiting and not awaiting in the program's output. Adjust the value
        // for howLong if necessary.
        await Task.Delay(howLong);
        resultsTextBox.Text +=
            "\r\n    Task.Delay is finished--returning from called method.";
    }
}

// Output with Call #1 or Call #2. (Wait for the last line to appear.)

// Entering the Click event handler.
//   Entering calling method.
//     Entering called method, starting and awaiting Task.Delay.
//   Returning from calling method.
// Exiting the Click event handler.
//     Task.Delay is finished--returning from called method.

// Output with Call #3, which awaits the call to CalledMethodAsync.

// Entering the Click event handler.
//   Entering calling method.
//     Entering called method, starting and awaiting Task.Delay.
//     Task.Delay is finished--returning from called method.
//   Returning from calling method.
```

```
    // Exiting the Click event handler.
}
```

5. Choose the F5 key to run the program, and then choose the **Start** button.

The expected output appears at the end of the code.

See also

- await
- Asynchronous Programming with async and await

Compiler Warning (level 2) CS0108

1/23/2019 • 2 minutes to read • Edit Online

'member1' hides inherited member 'member2'. Use the new keyword if hiding was intended.

A variable was declared with the same name as a variable in a base class. However, the new keyword was not used. This warning informs you that you should use **new**; the variable is declared as if **new** had been used in the declaration.

The following sample generates CS0108:

```
// CS0108.cs
// compile with: /W:2
using System;

namespace x
{
   public class clx
   {
      public int i = 1;
   }

   public class cly : clx
   {
      public static int i = 2;   // CS0108, use the new keyword
      // Use the following line instead:
      // public static new int i = 2;

      public static void Main()
      {
         Console.WriteLine(i);
      }
   }
}
```

See also

- new Modifier
- new

Compiler Warning (level 2) CS0467

10/16/2018 • 2 minutes to read • Edit Online

Ambiguity between method 'method' and non-method 'non-method'. Using method group.

Inherited members from different interfaces that have the same signature cause an ambiguity error.

Example

The following example generates CS0467.

```
// CS0467.cs
interface IList
{
    int Count { get; set; }
}

interface ICounter
{
    void Count(int i);
}

interface IListCounter : IList, ICounter {}

class Driver
{
    void Test(IListCounter x)
    {
        // The following line causes the warning. The assignment also
        // causes an error because you can't assign a value to a method.
        x.Count = 1;
        x.Count(3);
        // To resolve the warning, you can change the name of the method or
        // the property.

        // You also can disambiguate by specifying IList or ICounter.
        ((IList)x).Count = 1;
        ((ICounter)x).Count(3);
    }

    static void Main()
    {
    }
}
```

Compiler Warning (level 2) CS0618

5/4/2018 • 2 minutes to read • Edit Online

'member' is obsolete: 'text'

A class member was marked with the `Obsolete` attribute, such that a warning will be issued when the class member is referenced. For more information, see Common Attributes.

The following sample generates CS0618:

```
// CS0618.cs
// compile with: /W:2
using System;

public class C
{
   [Obsolete("Use newMethod instead", false)]   // warn if referenced
   public static void m2()
   {
   }

   public static void newMethod()
   {
   }
}

class MyClass
{
   public static void Main()
   {
      C.m2();  // CS0618
   }
}
```

Compiler Warning (level 2) CS1701

5/4/2018 • 2 minutes to read • Edit Online

Assuming assembly reference "Assembly Name #1" matches "Assembly Name #2", you may need to supply runtime policy

The two assemblies differ in release and/or version number. For unification to occur, you must specify directives in the application's .config file, and you must provide the correct strong name of an assembly, as demonstrated in the following example code.

Example

The following multifile sample references an assembly using two different external aliases. This first sample builds the older version of the code that creates assembly CS1701_d.

```
// CS1701_a.cs
// compile with: /target:library /out:cs1701_d.dll /keyfile:mykey.snk
using System.Reflection;
[assembly:AssemblyVersion("1.0")]
public class A {
   public void M1() {}
}

public class C1 {}
```

Example

This is the code that creates the newer version of assembly CS1701_d. Note that it compiles into a different directory than the older version, necessary since the output files have the same names.

```
// CS1701_b.cs
// compile with: /target:library /out:c:\\cs1701_d.dll /keyfile:mykey.snk
using System.Reflection;
[assembly:AssemblyVersion("2.0")]
public class A {
   public void M2() {}
   public void M1() {}
}

public class C2 {}
public class C1 {}
```

Example

This sample sets up the external aliases A1 and A2.

```
// CS1701_c.cs
// compile with: /target:library /reference:A2=c:\\cs1701_d.dll /reference:A1=cs1701_d.dll

extern alias A1;
extern alias A2;
// using System;
using a1 = A1::A;
using a2 = A2::A;

public class Ref {
   public static a1 A1() { return new a1(); }
   public static a2 A2() { return new a2(); }

   public static A1::C1 M1() { return new A1::C1(); }
   public static A2::C2 M2() { return new A2::C2(); }
}
```

Example

This sample calls methods using two different aliases of A. The following sample generates C1701.

```
// CS1701_d.cs
// compile with: /reference:c:\\CS1701_d.dll /reference:CS1701_c.dll
// CS1701 expected
class Tester {
   public static void Main() {
      Ref.A1().M1();
      Ref.A2().M2();
   }
}
```

Compiler Warning (level 3) CS0675

5/4/2018 • 2 minutes to read • Edit Online

Bitwise-or operator used on a sign-extended operand; consider casting to a smaller unsigned type first

The compiler implicitly widened and sign-extended a variable, and then used the resulting value in a bitwise OR operation. This can result in unexpected behavior.

The following sample generates CS0675:

```
// CS0675.cs
// compile with: /W:3
using System;

public class sign
{
   public static void Main()
   {
      int hi = 1;
      int lo = 1;
      long value = (((long)hi) << 32) | lo;              // CS0675
      // try the following line instead
      // long value = (((long)hi) << 32) | ((uint)lo);   // correct
   }
}
```

Compiler Warning (level 3) CS1700

5/4/2018 • 2 minutes to read • Edit Online

Assembly reference Assembly Name is invalid and cannot be resolved

This warning indicates that an attribute, such as InternalsVisibleToAttribute, was not specified correctly.

For more information, see Friend Assemblies.

Example

The following sample generates CS1700.

```
// CS1700.cs
// compile with: /target:library
using System.Runtime.CompilerServices;
[assembly:InternalsVisibleTo("app2, Retargetable=f")]   // CS1700
[assembly:InternalsVisibleTo("app2")]   // OK
```

Compiler Warning (level 4) CS0429

5/4/2018 • 2 minutes to read • Edit Online

Unreachable expression code detected

This error occurs whenever part of an expression in your code is unreachable. In the following example, the condition `false && myTest()` meets this criteria because the `myTest()` method will never get evaluated due to the fact that the left side of the `&&` operation is always false. As soon as the `&&` operator evaluates the `false` statement as false, it stops the evaluation, and will never evaluate the right side.

Example

The following code generates CS0429.

```
// CS0429.cs
public class cs0429
{
    public static void Main()
    {
        if (false && myTest())  // CS0429
        // Try the following line instead:
        // if (true && myTest())
        {
        }
        else
        {
            int i = 0;
            i++;
        }
    }

    static bool myTest() { return true; }
}
```

Compiler Warning (level 4) CS1591

5/4/2018 • 2 minutes to read • Edit Online

Missing XML comment for publicly visible type or member 'Type_or_Member'

The /doc compiler option was specified, but one or more constructs did not have comments.

The following sample generates CS1591:

```
// CS1591.cs
// compile with: /W:4 /doc:x.xml

/// text
public class Test
{
   // /// text
   public static void Main()   // CS1591, remove "//" from previous line
   {
   }
}
```

Compiler Warning (level 4) CS1610

5/4/2018 • 2 minutes to read • Edit Online

Unable to delete temporary file 'file' used for default Win32 resource -- resource

When using the /win32res compiler option and when your **%TEMP%** directory does not have DELETE permission, this warning indicates that the compiler could not delete a temporary file that it created.

Make sure that you have read/write/delete permissions for the **%TEMP%** directory.

If necessary, you can manually delete these files and there will be no harm to C# or any of your programs.