**Partial and Static Class**

It is possible to split the definition of a [class](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/class) or a [struct](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/struct), an [interface](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/interface) or a method over two or more source files. Each source file contains a section of the type or method definition, and all parts are combined when the application is compiled.

**Partial Classes**

There are several situations when splitting a class definition is desirable:

* When working on large projects, spreading a class over separate files enables multiple programmers to work on it at the same time.
* When working with automatically generated source, code can be added to the class without having to recreate the source file. Visual Studio uses this approach when it creates Windows Forms, Web service wrapper code, and so on. You can create code that uses these classes without having to modify the file created by Visual Studio.

 Partial method declarations must begin with the contextual keyword [partial](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/partial-type) and the method must return [void](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/void).

 Partial methods can have [ref](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/ref) but not [out](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/out) parameters.

 Partial methods are implicitly [private](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/private), and therefore they cannot be [virtual](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/virtual).

 Partial methods cannot be [extern](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/extern), because the presence of the body determines whether they are defining or implementing.

 Partial methods can have [static](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/static) and [unsafe](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/unsafe) modifiers.

 Partial methods can be generic. Constraints are put on the defining partial method declaration, and may optionally be repeated on the implementing one. Parameter and type parameter names do not have to be the same in the implementing declaration as in the defining one.

 You can make a [delegate](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/delegate) to a partial method that has been defined and implemented, but not to a partial method that has only been defined.

**Static Class**

A [static](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/static) class is basically the same as a non-static class, but there is one difference: a static class cannot be instantiated. In other words, you cannot use the [new](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/new) keyword to create a variable of the class type. Because there is no instance variable, you access the members of a static class by using the class name itself.

A static class can be used as a convenient container for sets of methods that just operate on input parameters and do not have to get or set any internal instance fields. For example, in the .NET Framework Class Library, the static [System.Math](https://docs.microsoft.com/en-us/dotnet/api/system.math) class contains methods that perform mathematical operations, without any requirement to store or retrieve data that is unique to a particular instance of the [Math](https://docs.microsoft.com/en-us/dotnet/api/system.math) class.

The following list provides the main features of a static class:

* Contains only static members.
* Cannot be instantiated.
* Is sealed.
* Cannot contain Instance Constructors.