Visual Studio 2010 - Visual C#

**Object and Collection Initializers (C# Programming Guide)**

Updated: October 2010

Object initializers let you assign values to any accessible fields or properties of an object at creation time without having to explicitly invoke a constructor. The following example shows how to use an object initializer with a named type, Cat. Note the use of auto-implemented properties in the Cat class. For more information, see [Auto-Implemented Properties (C# Programming Guide)](http://msdn.microsoft.com/en-us/library/bb384054.aspx).

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl02_code');" \o "Copy Code)

private class Cat

{

// Auto-implemented properties.

public int Age { get; set; }

public string Name { get; set; }

}

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl03_code');" \o "Copy Code)

Cat cat = new Cat { Age = 10, Name = "Fluffy" };

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifObject Initializers with anonymous types

Although object initializers can be used in any context, they are especially useful in LINQ query expressions. Query expressions make frequent use of [anonymous types](http://msdn.microsoft.com/en-us/library/bb397696.aspx), which can only be initialized by using an object initializer, as shown in the following declaration.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl27_ctl00_ctl01_code');" \o "Copy Code)

var pet = new { Age = 10, Name = "Fluffy" };

Anonymous types enable the **select** clause in a LINQ query expression to transform objects of the original sequence into objects whose value and shape may differ from the original. This is useful if you want to store only a part of the information from each object in a sequence. In the following example, assume that a product object (p) contains many fields and methods, and that you are only interested in creating a sequence of objects that contain the product name and the unit price.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl27_ctl00_ctl02_code');" \o "Copy Code)

var productInfos =

from p in products

select new { p.ProductName, p.UnitPrice };

When this query is executed, the productInfos variable will contain a sequence of objects that can be accessed in a **foreach** statement as shown in this example:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl27_ctl00_ctl03_code');" \o "Copy Code)

foreach(var p in productInfos){...}

Each object in the new anonymous type has two public properties which receive the same names as the properties or fields in the original object. You can also rename a field when you are creating an anonymous type; the following example renames the UnitPrice field to Price.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl27_ctl00_ctl04_code');" \o "Copy Code)

select new {p.ProductName, Price = p.UnitPrice};

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifObject initializers with nullable types

It is a compile-time error to use a collection initializer with a nullable struct.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifCollection Initializers

Collection initializers let you specify one or more element intializers when you initialize a collection class that implements [IEnumerable](http://msdn.microsoft.com/en-us/library/system.collections.ienumerable.aspx). The element initializers can be a simple value, an expression or an object initializer. By using a collection initializer you do not have to specify multiple calls to the **Add** method of the class in your source code; the compiler adds the calls.

The following examples shows two simple collection initializers:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl29_ctl00_ctl01_code');" \o "Copy Code)

List<int> digits = new List<int> { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

List<int> digits2 = new List<int> { 0 + 1, 12 % 3, MakeInt() };

The following collection initializer uses object initializers to initialize objects of the Cat class defined in a previous example. Note that the individual object initializers are enclosed in braces and separated by commas.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl29_ctl00_ctl02_code');" \o "Copy Code)

List<Cat> cats = new List<Cat>

{

new Cat(){ Name = "Sylvester", Age=8 },

new Cat(){ Name = "Whiskers", Age=2 },

new Cat(){ Name = "Sasha", Age=14 }

};

You can specify [null](http://msdn.microsoft.com/en-us/library/edakx9da.aspx) as an element in a collection initializer if the collection's **Add** method allows it.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl29_ctl00_ctl04_code');" \o "Copy Code)

List<Cat> moreCats = new List<Cat>

{

new Cat(){ Name = "Furrytail", Age=5 },

new Cat(){ Name = "Peaches", Age=4 },

null

};

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifExample

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl30_ctl00_ctl00_code');" \o "Copy Code)

// The following code consolidates examples from the topic.

class ObjInitializers

{

private class Cat

{

// Auto-implemented properties.

public int Age { get; set; }

public string Name { get; set; }

}

static void Main()

{

Cat cat = new Cat { Age = 10, Name = "Fluffy" };

List<Cat> cats = new List<Cat>

{

new Cat(){ Name = "Sylvester", Age=8 },

new Cat(){ Name = "Whiskers", Age=2 },

new Cat(){ Name = "Sasha", Age=14 }

};

List<Cat> moreCats = new List<Cat>

{

new Cat(){ Name = "Furrytail", Age=5 },

new Cat(){ Name = "Peaches", Age=4 },

null

};

// Display results.

System.Console.WriteLine(cat.Name);

foreach (Cat c in cats)

System.Console.WriteLine(c.Name);

foreach (Cat c in moreCats)

if (c != null)

System.Console.WriteLine(c.Name);

else

System.Console.WriteLine("List element has null value.");

}

// Output:

//Fluffy

//Sylvester

//Whiskers

//Sasha

//Furrytail

//Peaches

//List element has null value.

}