**Property Initializer**

Properties were always a more powerful version of fields. But sometimes you just want them to behave the same and the concise notation was not available. It’s fixed in C# 6.

Let me start with a short history of properties syntax in C#. In early C# times there was only one way to declare them – very explicit:

private string \_name = "Joe"; // backing field

public string Name

{

get { return \_name; }

set { \_name = value; }

}

There was a little improvement introduced in C# 3 – Auto-implemented properties. Basically the need to create backing field was left for compiler.

public string Name { get; set; }

But still it was a bit cumbersome if you wanted to initialize property. You could either use old explicit syntax or do it in constructor.

public string Name { get; set; }

public MyClass()

{

Name = "Joe";

}

Not perfect and not nice, especially if you want to do a simple initialization, like the above. The [C# 6](https://csharp.today/tag/c-6/) release serves well here. You can now initialize property inline. It’s concise and readable.

public string Name { get; set; } = "Joe";

Important to know is that the initializer sets the backing field directly, so the auto-property setter is not used.

Property initializers work similar to field initializers:

* Are executed in declaration order (along with field initializers)
* Can’t reference this  as are executed before object is initialized (before constructor is called)

**Getter-only auto-properties**

It was always possible to create getter-only properties, but the auto-property syntax had not sense for getter-only case. Compiler didn’t know where it should take value from. C# 6 added auto-property initializers and now it makes sense to use it with getter-only properties.

public string Name { get; } = "Joe";

The backing field of getter-only auto-property is declared as read-only. It can be initialized by auto-property initializer or in constructor. In both cases value is assigned directly to backing field.

public string Name { get; } = "Joe";

public MyClass(string name)

{

if (!string.IsNullOrEmpty(name))

{

Name = name;

}

}

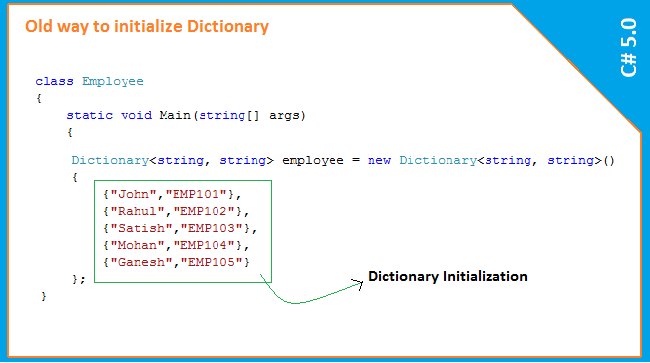
At front auto-properties were shorthand available only if you were willing to make class mutable. Now, getter-only auto-properties are shorthand for immutable approach. The support for mutable and immutable properties have been leveled.

**Dictionary Initializer**

A Dictionary is a generic class of C# that is used to represent a collection of keys and value pairs data. It takes two parameters, the first for keys and the second for values. As we know a Dictionary is a generic class so it takes the parameter as a generic type, it means we can use any type of parameter like int, string, float, class and so on.

* Adding data into a Dictionary collection  
    
  Example:
  1. Dictionary<string, int> student = new Dictionary<string, int>();
  3. student.Add("Ravi Sharma", 20);
  4. student.Add("Mohit Sharma", 22);
  5. student.Add("Varun Sharma", 21);
  7. foreach(KeyValuePair<string,int> stu in student)
  8. {
  9. Console.WriteLine("Name : {0}, Age : {1}",stu.Key,stu.Value);
  10. }

* Initializing a Dictionary in C# 5.0.  
    
  Example:
  1. Dictionary<string, string> employee = new Dictionary<string, string>()
  2. {
  3. {"John","EMP101"},
  4. {"Rahul","EMP102"},
  5. {"Satish","EMP103"},
  6. {"Mohan","EMP104"},
  7. {"Ganesh","EMP105"}
  8. };



Need of Dictionary

Basically a Dictionary is used whenever we need to keep the generic collection of data. As we know the same can be done with a HashTable that also takes the parameter in the keys and values pair but a huge difference between both of them. Let's see the similarities and differences between both of them.

* Similarity:  
    
  Both takes the parameters in key and value pairs  
    
  Example:
  1. //Using Hashtable
  2. Hashtable Student = new Hashtable();
  3. Student.Add("Mohit Sharma",20);
  4. Student.Add("Varun Sharma",22);
  6. //Using Dictionary
  7. Dictionary<string, int> student1 = new Dictionary<string, int>();
  8. student1.Add("Mohit Sharma", 20);
  9. student1.Add("Varun Sharma", 22);
* Differences:

|  |  |
| --- | --- |
| Dictionary | HashTable |
| If we are using Dictionary, we need to use System.Collections.Generic Namespace | If we are using HashTable, we need to use System.Collections Namespace |
| It is faster than HashTable | It is slower than Dictionary because it requires boxing and unboxing |
| It will generate error if we try to use a key that is not exist in the collection | It will return null if we try to use a key that is not exist |
| It is a generic type so we can use any type of data | It's not a generic type |

Let us see the new way to initialize Dictionary data. In the preceding discussions we saw the old way to initialize a Dictionary.

Dictionary Initializers In C# 6.0   
  
In C# 6.0, the way to initialize a Dictionary object became simpler and looks more feasible than the way we previously initialized the older version of C#. Now we can put the key values in square brackets and put the value depending on the type of keys.

Example   
  
If the keys data type is string and the values are also strings. In this example we will use a Dictionary collection that will keep the employee's name (in a string ) and id (in a string also).

1. Dictionary<string, string> employee = new Dictionary<string, string>()
2. {
3. ["John"]="EMP101",
4. ["Rahul"]="EMP102",
5. ["Satish"]="EMP103",
6. ["Mohan"]="EMP104",
7. ["Ganesh"]="EMP105"
9. };

Example

If the keys data type is int and the values are string. In this example we will use a Dictionary collection that will keep the employee's age (in an int ) and name (in a string).

1. Dictionary<int, string> employee = new Dictionary<int, string>()
2. {
3. [25]="John",
4. [28]="Rahul",
5. [25]="Satish",
6. [22]="Mohan",
7. [26]="Ganesh"
9. };

Example

If the keys data type is int and the values are a class type.

1. class Employee
2. {
4. public string Name { get; set; }
5. public string Email { get; set; }
7. Employee empDetails = new Employee();
8. empDetails.Email = "john@mcn.net";
9. empDetails.Name = "JOHN";
11. Dictionary<int, Employee> emp = new Dictionary<int, Employee>();
12. emp.Add(1, empDetails);
13. }

