Previous Next

C# - Anonymous Type

Anonymous type, as the name suggests, is a type that doesn't have any name. C# allows you to create an object with the *new* keyword without defining its class. The implicitly typed variable- var is used to hold the reference of anonymous types.

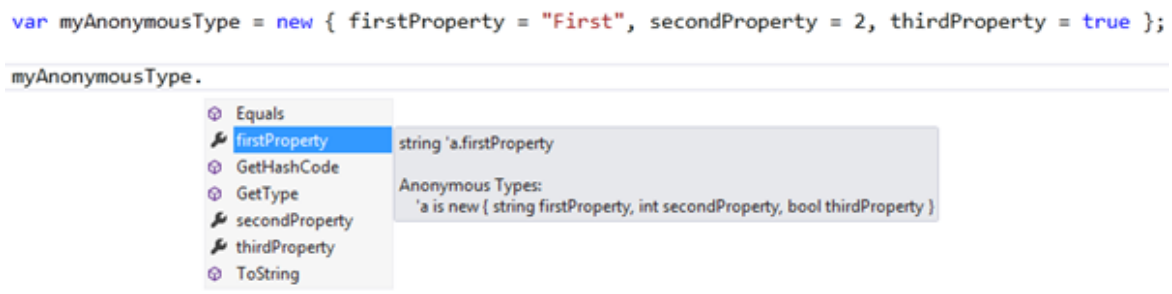
Example: Anonymous Type

```
var myAnonymousType = new { firstProperty = "First",
    secondProperty = 2,
    thirdProperty = true
};
```

In the above example, myAnonymousType is an object of anonymous type created using the new keyword and object initializer syntax It includes three properties of different data types.

An anonymous type is a temporary data type that is inferred based on the data that you include in an object initializer. Properties of anonymous types will be **read-only** properties so you cannot change their values.

Anonymous types have intellisense support in Visual Studio:



Intellisense support for anonymous type

Notice that the compiler applies the appropriate type to each property based on the value expression. For example, firstProperty is a string type, secondProperty is an int type and thirdProperty is a bool.

Internally, the compiler automatically generates the new type for anonymous types. You can check the type of an anonymous type as shown below.

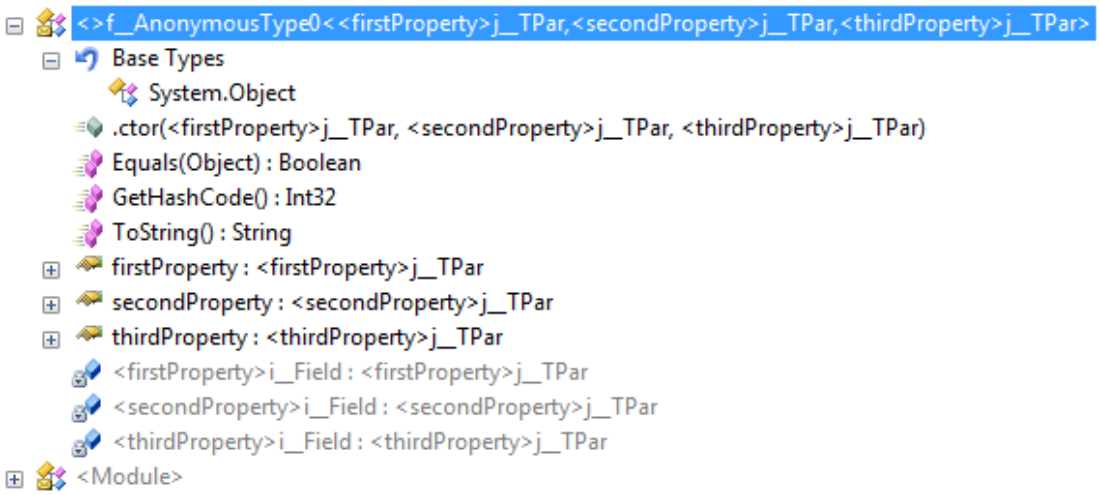
```
static void Main(string[] args)
{
    var myAnonymousType = new { firstProperty = "First",
                                secondProperty = 2,
                                thirdProperty = true
                              };

    Console.WriteLine(myAnonymousType.GetType().ToString());
}
```

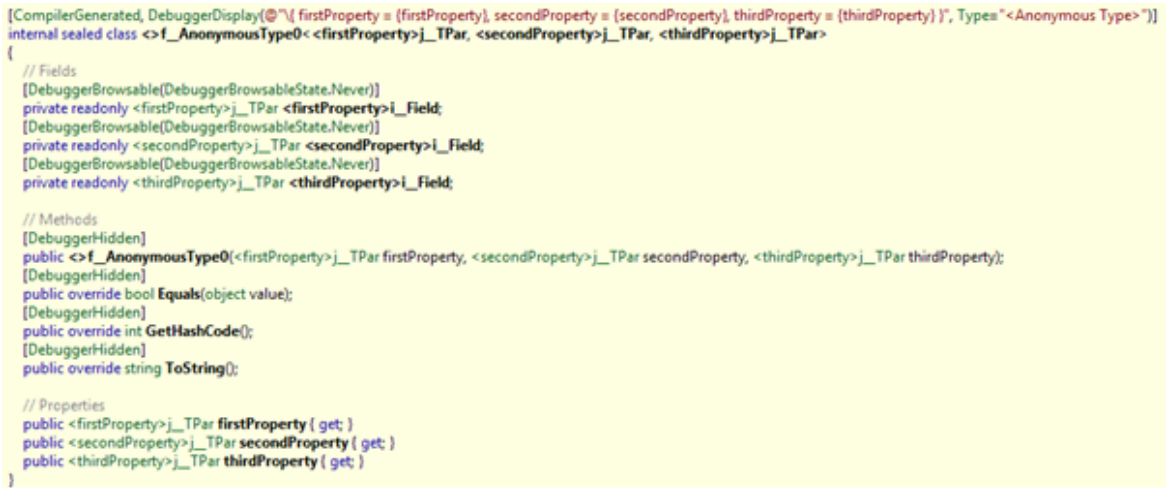
Output:

```
<>f__AnonymousType0'3[System.String,System.Int32,System.Boolean]
```

As you can see in the above output, the compiler generates a type with some cryptic name for an anonymous type. If you see the above anonymous type in reflector, it looks like below.



Disassembled Anonymous Type



```
[CompilerGenerated, DebuggerDisplay("{ firstProperty = {firstProperty}, secondProperty = {secondProperty}, thirdProperty = {thirdProperty} }", Type="<Anonymous Type>")]
internal sealed class <>f__AnonymousType0<<firstProperty>j__TPar, <secondProperty>j__TPar, <thirdProperty>j__TPar>
{
  // Fields
  [DebuggerBrowsable(DebuggerBrowsableState.Never)]
  private readonly <firstProperty>j__TPar <firstProperty>i__Field;
  [DebuggerBrowsable(DebuggerBrowsableState.Never)]
  private readonly <secondProperty>j__TPar <secondProperty>i__Field;
  [DebuggerBrowsable(DebuggerBrowsableState.Never)]
  private readonly <thirdProperty>j__TPar <thirdProperty>i__Field;

  // Methods
  [DebuggerHidden]
  public <>f__AnonymousType0(<firstProperty>j__TPar firstProperty, <secondProperty>j__TPar secondProperty, <thirdProperty>j__TPar thirdProperty);
  [DebuggerHidden]
  public override bool Equals(object value);
  [DebuggerHidden]
  public override int GetHashCode();
  [DebuggerHidden]
  public override string ToString();

  // Properties
  public <firstProperty>j__TPar firstProperty { get; }
  public <secondProperty>j__TPar secondProperty { get; }
  public <thirdProperty>j__TPar thirdProperty { get; }
}
```

Disassembled Anonymous Type

Notice that it is derived from the System.Object class. Also, it is a sealed class and all the properties are created as read only properties.

Nested Anonymous Type

An anonymous type can have another anonymous type as a property.

Example: Nested Anonymous Type

```
var myAnonymousType = new
                      {
                          firstProperty = "First",
                          secondProperty = 2,
                          thirdProperty = true,
                                 anotherAnonymousType = new { nestedProperty =
"Nested"}
                      };
```

Nested anonymous types also have intellisense support.

Scope of Anonymous Type

An anonymous type will always be local to the method where it is defined. Usually, you cannot pass an anonymus type to another method; however, you can pass it to a method that accepts a parameter of dynamic type. Please note that Passing anonymous types using dynamic is not recommended.

Example: Passing Anonymous Type

```
static void Main(string[] args)
```

```
{

    var myAnonymousType = new
                          {
                              firstProperty = "First Property",
                              secondProperty = 2,
                              thirdProperty = true
                          };

    DoSomethig(myAnonymousType);
}

static void DoSomethig(dynamic param)
{
    Console.WriteLine(param.firstProperty);
}
```

Output:

```
First Property
```

Anonymous Types with a LINQ Query

Linq Select clause creates an anonymous type as a result of a query to include various properties which is not defined in any class. Consider the following example.

Example: Return Anonymous Type from LINQ Query

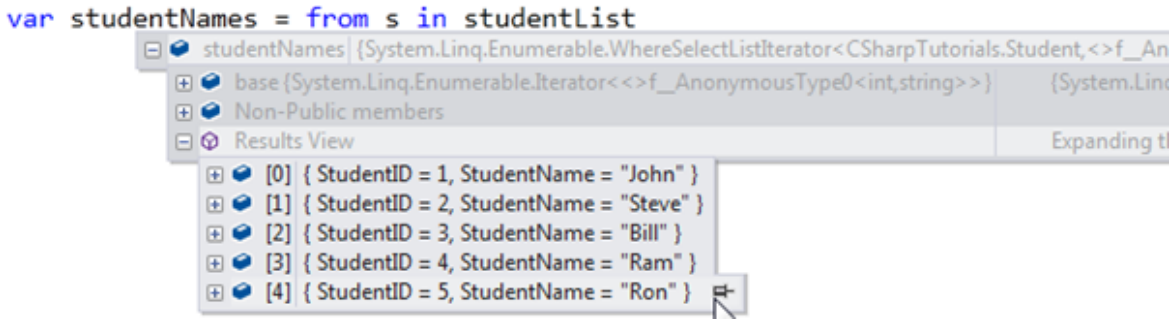
```
public class Student
{
    public int StudentID { get; set; }
    public string StudentName { get; set; }
    public int age { get; set; }
}

class Program
{
    static void Main(string[] args)
    {
        IList<Student> studentList = new List<Student>() {
                        new Student() { StudentID = 1, StudentName = "John", age
= 18 } ,
                          new Student() { StudentID = 2, StudentName = "Steve",
age = 21 } ,
                           new Student() { StudentID = 3, StudentName = "Bill",
```

```
        age = 18 } ,
                            new Student() { StudentID = 4, StudentName = "Ram" , age
        = 20  } ,
                            new Student() { StudentID = 5, StudentName = "Ron" , age
        = 21 }
                        };

            var studentNames = from s in studentList
                               select new { StudentID = s.StudentID,
                                            StudentName = s.StudentName
                                          };
        }
    }
```

In the above example, Student class includes various properties. In the Main() method, Linq select clause creates an anonymous type to include only StudentId and StudentName instead of including all the properties in a result. Thus, it is useful in saving memory and unnecessary code. The query result collection includes only StudentID and StudentName properties as shown in the following debug view.



Anonymous Type in debug view

Points to Remember :

1) Anonymous type can be defined using the new keyword and object initializer syntax.

2) The implicitly typed variable- **var**, is used to hold an anonymous type.

3) Anonymous type is a **reference type** and all the properties are read-only.

4) The scope of an anonymous type is local to the method where it is defined.

Previous Next