New features added to C# 5.0

Author : Shailendra Chauhan
Posted On : 17 Nov 2013
Updated On : 17 Nov 2013

C# most recent version 5.0 was released on August 15, 2012 with .NET Framework 4.5 and Visual Studio 2012. There are two main features in C# 5.0 - Async Programming and Caller Information. Let's understand both these features in details as given below.

Async Feature (Asynchronous Methods)

C# 5.0 Async feature introduces two keywords async and await which allows you to write asynchronous code more easily and intuitively like as synchronous code. Before C# 5.0, for writing an asynchronous code, you need to define callbacks (also known as continuations) to capture what happens after an asynchronous process finishes. This makes your code and other routine task such exception handling complicated.

Both the keywords are used in a combination of each other. Hence, an await operator is applied to a one or more than one expressions of an async method. An async method returns a Task or Task<TResult> that represents the ongoing work of the method. The task contains information that the caller of the asynchronous method can use, such as the status of the task, its unique ID, and the method's result.

```
public async Task<IEnumerable<Product>> GetProductList()
{
 HttpClient client = new HttpClient();
 Uri address = new Uri("http://dotnet-tricks.com/");
 client.BaseAddress = address;

 HttpResponseMessage response = await client.GetAsync("myservice/product/ProductList");

 if (response.IsSuccessStatusCode)
 {
 var list = await response.Content.ReadAsAsync<IEnumerable<Product>>();
 return list;
 }
 else
 {
 return null;
 }
}
```

Caller Information (Caller info attributes)

Caller Information can help you in tracing, debugging and creating diagnose tools. It will help you to avoid duplicate codes which are generally invoked in many methods for same purpose, such as logging and tracing.

You could get the following information of caller method:

01. CallerFilePathAttribute

Full path of the source file that contains the caller. This is the file path at compile time.

02. CallerLineNumberAttribute

Line number in the source file at which the method is called.

03. CallerMemberNameAttribute

Method or property name of the caller.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

class Example
{
 static void Main(string[] args)
 {
 Console.WriteLine("Main method Start");
 InsertLog("Main");
 MyMethodB();
 MyMethodA();
 Console.WriteLine("Main method End!");
 Console.ReadLine(); // hold on result
 }

 static void MyMethodA()
 {
 InsertLog("MyMethodA");
 MyMethodB();
 }

 static void MyMethodB()
 {
 // some code here.
 }

 static void InsertLog(string method)
 {
 Console.WriteLine("{0} called MyMethodB at {1}", method,
 DateTime.Now);
 }
}
/* Output:
 Main method Start
 Main called MyMethodB at 11/17/2013 11:12:24 PM
 MyMethodA called MyMethodB at 11/17/2013 11:12:24 PM
 Main method End!
*/
```

In both Main and MyMethodA, method InsertLog is invoked for logging. Now we can change the above code as follows.

```
using System;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using System.Text;
using System.Threading.Tasks;

class Example
{
 static void Main(string[] args)
 {
 Console.WriteLine("Main method Start");
 MyMethodB();
 MyMethodA();
 Console.WriteLine("Main method End!");
 Console.ReadLine();
 }

 static void MyMethodA()
 {
 MyMethodB();
 }

 static void MyMethodB([CallerMemberName] string memberName = "", [CallerFilePath] string sourceFilePath = "", [CallerLineNumber] int sourceLineNumber =
 0)
 {
 InsertLog(memberName);
 }

 static void InsertLog(string method)
 {
 Console.WriteLine("{0} called MyMethodB at {1}", method, DateTime.Now);
 }
}

/*Output:
 Main method Start
 Main called MyMethodB at 11/17/2013 10:30:11 PM
 MyMethodA called MyMethodB at 11/17/2013 10:30:11 PM
 Main method End!
 */
```