

Agenda

1. Introduction to Console based Applications.
2. *Creating a Solution that contains a project.*
3. *Writing a simple program and executing it.*
4. *Understanding Command Line Arguments.*

Table of Contents

CONSOLE BASED APPLICATION	2
STEPS TO CREATE YOUR FIRST CONSOLE APPLICATION	3
COMMAND LINE ARGUMENTS	6
MANAGING MULTIPLE MODULES IN ONE PROJECT	8

Deccansoft

Console Based Application

Introduction

Console based applications run on command prompt instead of Windows environment. These are simple to use and one can put in simple commands for execution.

- VS 2010 consists of a solution that can contain several projects and each of these projects can be of a different language(VB or C#). A solution file has an extension of .sln. Whenever a solution is opened, it's related projects also open along with it.
- A project consists of files including Source Code (.vb/.cs), Resources(.resx), Configuration(.config) files etc...The project file has the extension .vbproj or .csproj
- A project when build (compilation + linking) generates an EXE/DLL as output, which is called as PE(Portable Executable).

Portable Executable(PE):

It is a Microsoft Win32 compatible file for .NET applications which contains the MSIL code and Metadata in binary form. It has the extension .exe or .dll. PE has COFF(Common Object File Format) specification.

Steps to create your first Console Application

1. Goto File -> New → Project → Select Visual Basic on Left and Console Application on right.

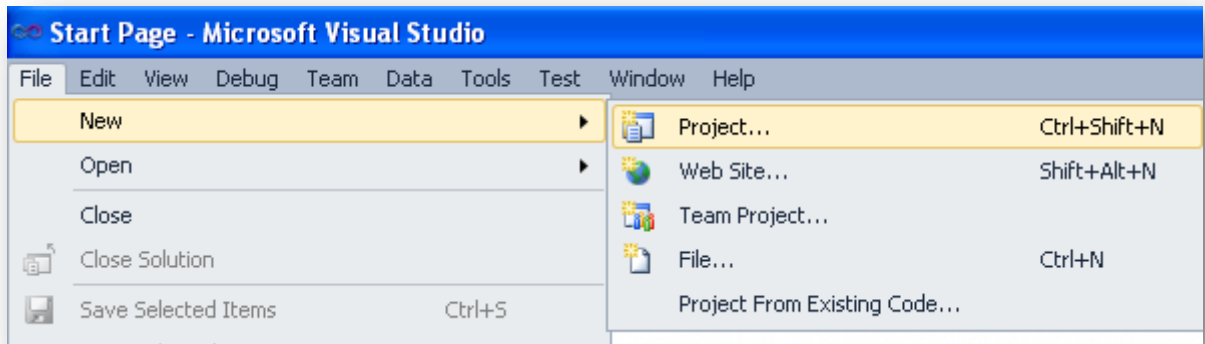


Fig 1.1

2. This will open a “New Project” Dialog Box. Type the name as: **FirstConApp**.
3. Select Visual Basic option from Installed Templates and then choose Console Application.
4. Set a convenient location as: **D:\Demo**
5. Make sure the option of “Create a directory for Solution” is checked.
6. Your screen should look as in **Fig 1.2**

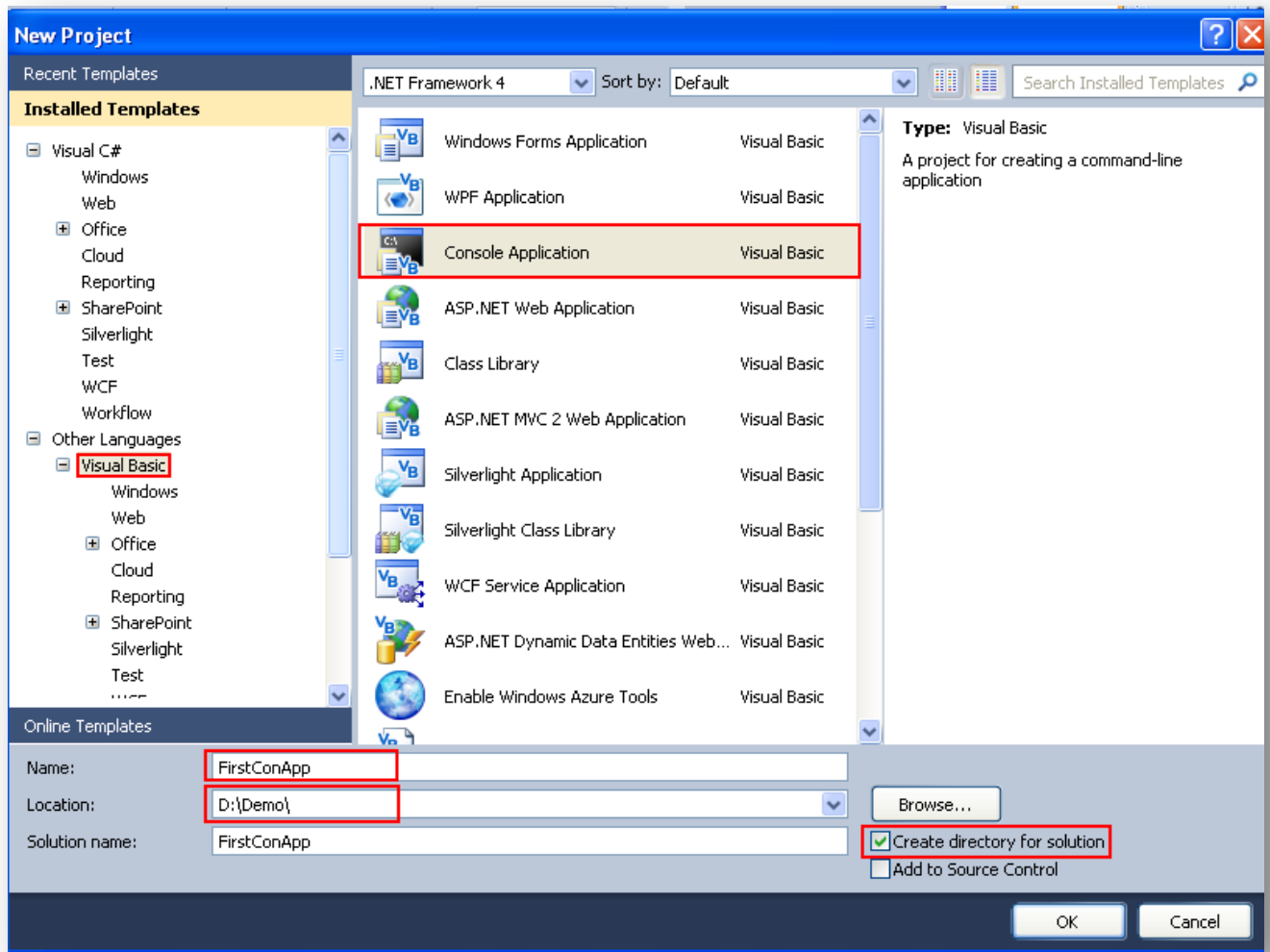


Fig 1.2

7. Then click on Ok.

Visual Studio 2010 will create a default file: **Module1.vb** as shown in **Fig: 1.3**

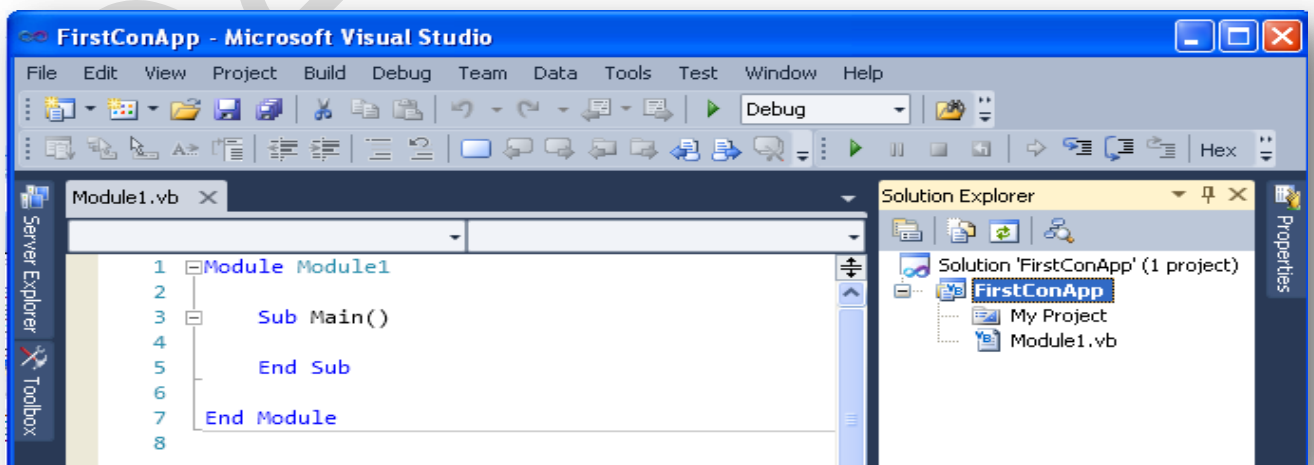


Fig 1.3

Now add the code as follows:

```
First Console Application
Module Module1
    Sub Main()
        Console.WriteLine("Hello Module1")
    End Sub
End Module
Code: 1.1 VB
```

8. To Build the Solution:

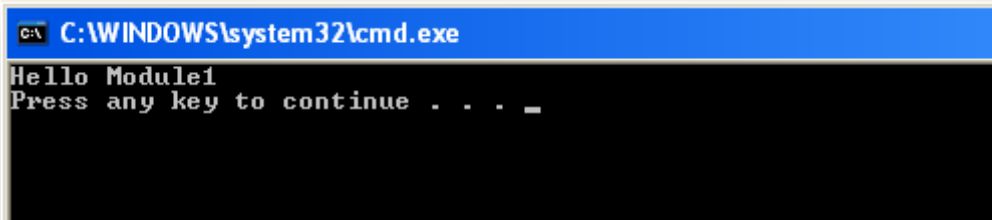
Goto Menu: Build → Build Solution (Ctrl + Shift + B)

9. To Run the Application, select one of the following options:

Option 1: Debug → Start Debugging (F5) – Here we can use break points and debug the application.

Option 2: Debug → Start Without Debugging (Ctrl + F5) - We can see the output in console window.

Here is the output of executing the application:



```
C:\WINDOWS\system32\cmd.exe
Hello Module1
Press any key to continue . . . _
```

Fig 1.4

When the project is build the following output file is generated:

Drive:\...\<ProjectName>\bin\debug\<ProjectName>.Exe

Note:

In VS.NET 2010, at the time of creating a project we can specify the Framework Version to be used which can be either 2.0, 3.0, 3.5 or 4.0.

Command Line Arguments

Mostly, the console based applications are developed in situations where the application should either run as a top level process or as child process (invoked from another parent process).

Command line Arguments: This is the best way of providing input to the console based application.

Console Application with Arguments	
<pre>Module Module1 Sub Main(ByVal args As String()) 'args is an array of commandline arguments. Console.WriteLine("Hello " & args(0)) 'args(0) is the first command line argument. End Sub End Module</pre>	
Code: 1.2	VB

To give Commandline arguments in Visual Studio: View → Solution Explorer → Select Project → Right Click → Properties → Debug Tab → Command Line Arguments → Provide string separated by space (Args1 Args2 Args3)

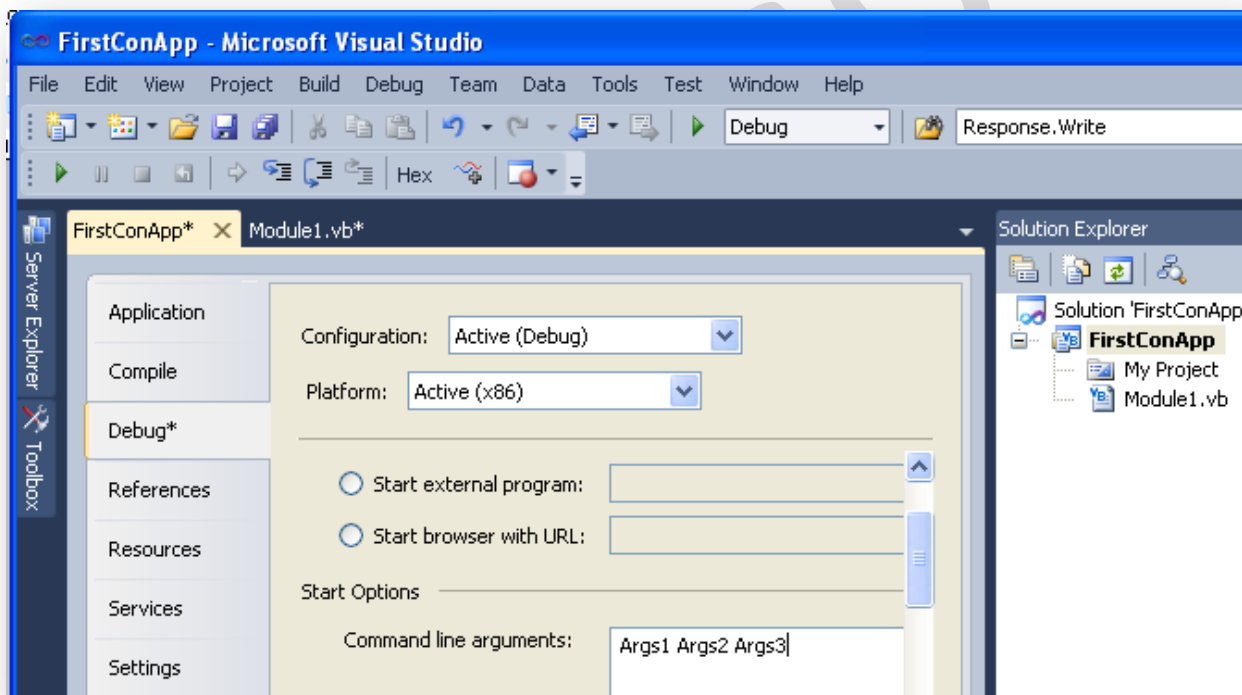


Fig 1.4

Or give at command prompt in console window

Drive:\. . . \ ProjectName\ in\debug**<ProjectName>**.exe Args1 Args2 Args3

Return value of Main:

```
Console application returning a value
Module Module1
    Function Main(ByVal args() As String) As Integer
        Console.WriteLine("Hello " & args(0))
        Return 0
    End Function
End Module
```

Code: 1.3	VB
-----------	----

- When the application is not used as child process, the return value of main is not useful because whatever may return the value, main ends the application ends and OS will clean all the memory allocated to it.
- The return value of Main is used by the current application to communicate its state to the parent process when the application terminated. The Integer return value can be predefined with some meaning and the same will be used by the parent process to know the state of child process which terminated. The return value of an application is called as EXIT CODE of the application.

Managing Multiple Modules in One Project

One Project can have multiple modules but only one of many modules with valid Main can be treated as Startup or Entry Point of the application.

To Add another File with Module to Project:

Goto Solution Explorer → Right Click on Project → Add → Module

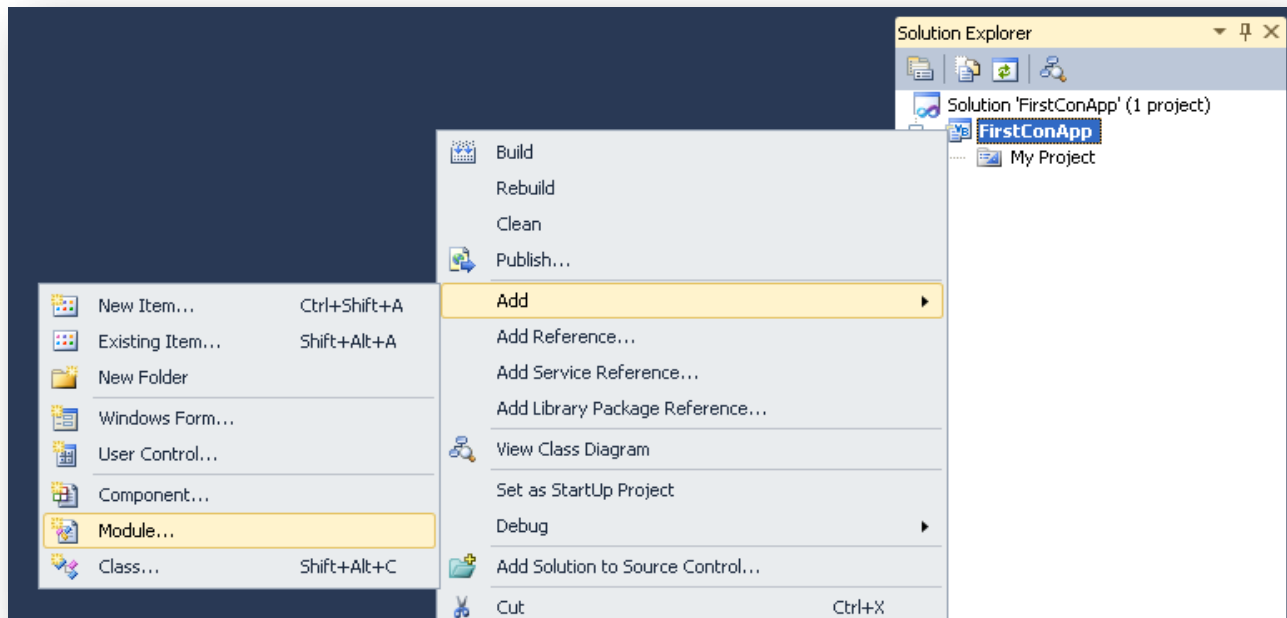


Fig 1.5

Add the code as follows:

```
Console application returning a value
Module Module2
  Sub Main()
    Console.WriteLine("Module2")
  End Sub
End Module
```

Code: 1.4	VB
-----------	----

To Set Startup Object:

View → Solution Explorer → Project → Right Click → Properties → Application Tab → Startup Object → Select Module whose Main should be used as entry point.

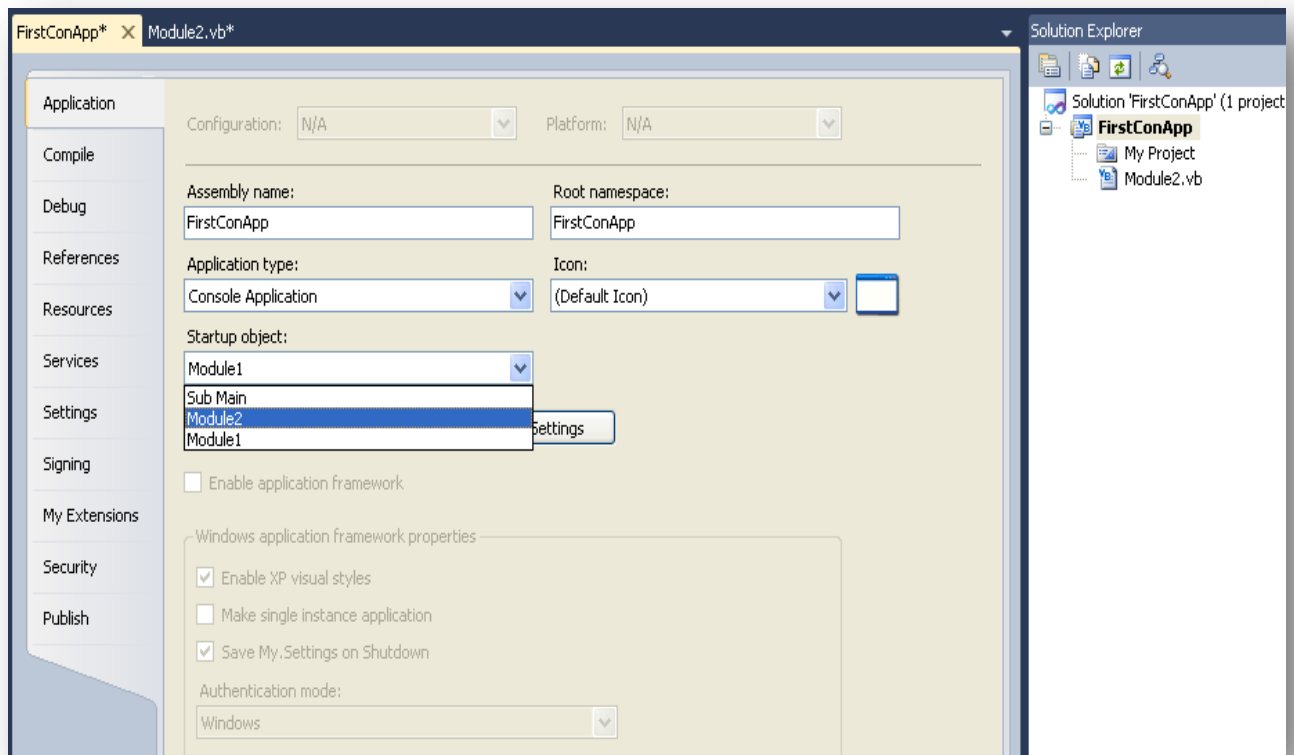


Fig 1.6

Note:

The module declared as startup object cannot have more than one form of Main which is valid for entry point.

Here is the output of Code 1.4:

```

C:\WINDOWS\system32\cmd.exe
Module2
Press any key to continue . . . _
  
```

Fig 1.7

Every Module in the project can be placed in same file or in different files. In either case the compiler is going to compile them together. i.e. for a dotnet language compiler files cannot be used as boundaries and all the files are compiled together as one unit.

Following are the possible forms of Main which can be used in a module so that the module can be marked as startup object.

Form 1:

```
Sub Main()  
    Console.WriteLine("Module1")  
End Sub
```

Form 2:

```
Function Main() As Integer  
    Console.WriteLine("Hello")  
    Return 0  
End Function
```

Form 3:

```
Sub Main(ByVal args As String())  
    Console.WriteLine("Hello " & args(0))  
End Sub
```

Form 4:

```
Function Main(ByVal args() As String) as Integer.  
    Console.WriteLine("Hello " & args(0))  
    Return 0  
End Function
```

If Main is written in Class it must be declared as **Shared**.

 **Tips:****An instruction to Object Oriented Programmers:**

As an Object Oriented programmer it's always better to avoid usage of Module in a project. This is because every public member of a module is by treated as Global Member and it has be accessed in any part of the application without explicitly using module name and this against object orientated principles.

Thus all the above four forms of entry point Main can be written in Class but then they should be declared as **Shared** in VB.NET / **Static** in C#.

Here we have set the values of arguments: **Args1 Args2 Args3**

Program to display command line argument values

```
Class Program  
    Function Main(ByVal args() As String) As Integer  
        Console.WriteLine("Hello " & args(0))  
        Return 0  
    End Function  
End Class
```

Code: 1.5

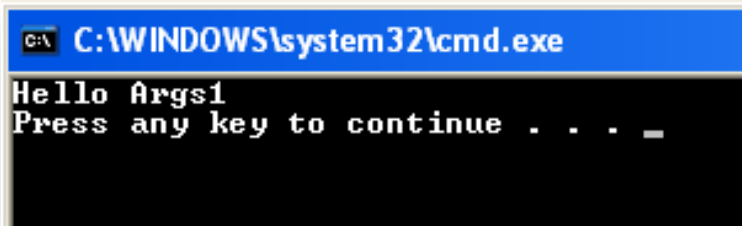
VB

Program to display command line argument values

```
using System;
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello " + args[0]);
    }
}
```

Code: 1.5

C#

Note: C# doesn't support Module.**Output of executing the Code 1.5:***Fig 1.8***To Open a closed Solution / Project:**

File → Open → Project / Solution → Give the Path of .sln file.

To develop an application without using VS.NET (compiling at command prompt):

1. Open Notepad
2. Write the code as given above and Save as C:\Demo.CS
3. Goto Start → Programs → Microsoft Visual Studio 2008 → Visual Studio Tools → VS 2008 Command Prompt (so that the directory of MS.NET Framework is set in "PATH")
4. CSC.EXE C:\Demo.CS => Generates Demo.exe
5. Run Demo.exe

Framework Folder:

C:\Windows\Microsoft.Net\Framwork\V3.5.XXXX\

Summary:

In this section, we have studied about Console Applications and seen how to create a Console based application. In addition to this, one can also pass arguments from command prompt or through settings in Visual Studio.