.NET Framework 4.6 and C# 7.0

Lesson 04 : Using Microsoft Visual Studio

Capgemini

Lesson Objectives

- After completing this module you will understand:
  - Debugging techniques in .NET
  - Creating multiple projects in one solution

4.1: Debugging in Visual Studio

Overview

- Debugging is about running your code line by line to ensure that the execution path and data are both correct
- Debugging is trickier and difficult as compared to writing a code
- Around 30 percent of the developer's time is spent in debugging
- Visual Studio offers many debugging features

A developer normally has to spend 30 percent of their time debugging a code. Also we know that debugging is more tricky and difficult as compared to writing the program code. In order to successfully debug a code, you should be familiar with the debugging tools available to you and should be able to effectively use them.

Visual Studio offers us many debugging features. We will be exploring through them in this section. Visual Studio IDE always has an admirable support for warning of potential errors at design time. Any syntax errors that is likely to cause problems during compilation process are underlined(squiggled) or highlighted. An error notification pops up when an exception occurs during debugging session and recommends a course of action that would prevents the exception. These all features has been part of the Visual Studio IDE.

Having all these features incorporated into the IDE itself helps in writing an almost bug free code. Why almost?? Because it is near to impossible to write a Bug free code.

The editor that we user to write C# code shows squiggles and tooltips for many errors before compilation. Also Visual Studio now has greatly improved the XML editor with enhancements like full XML1.0 syntax checking, Support for DTD and XSD validation.

Not only the XML files but even the HTML or ASPX code is checked by the editor to show any potential compilation errors.

4.2: Debugging Tools in Visual Studio

Debugging Tools

- Debugging Tools
  - Integrated Debugger
  - Visual Debugger

Visual Studio.NET provides you with debugging tools like:

Integrated debugger: Integrated with the Visual Studio itself. There is no need to run a separate tool to debug your application.

Visual debugger: This is a separate tool that you can find at "\Microsoft Visual Studio 5\SDK\v2.0\GuiDebug\DbgCLR.exe" (locate this at your own Visual Studio folder). Using this tool you can debug complex bugs that you can not fix using the aforementioned integrated debugger.

4.3: What does Debugging Offer?

Features

- Debugging Offers:
  - Code execution examination. Step line by line
  - Viewing variable values at each step
  - Changing the value of a certain variable
  - Moving the execution point and running application to another point
- These features can be categorized as:
  - Breakpoints
  - Stepping
  - Data Viewing

ASP.NET provides you with two classes: 'Debug' and 'Trace'. These are diagnostic classes available in System.Diagnostics namespace, to help you generate logs during development and debugging.

During debugging, you can see how exactly your code is executed by stepping line by line, viewing the values of your variables during each step and showing how these values are changed by each step. You can change the value of a certain variable, You can move the execution point and run application to another point.

Visual Studio integrated debugger has many features you can use to debug your applications.
These features includes the following: Breakpoints, Stepping, and Data Viewing.

4.4: Breakpoints

Overview

- Places in code, where the debugger stops application execution
- To set the breakpoint, click the gray margin to the left of the line
- Point is visible as a red circle in the left gray margin
  - To clear the breakpoint, click the red circle or press F9.

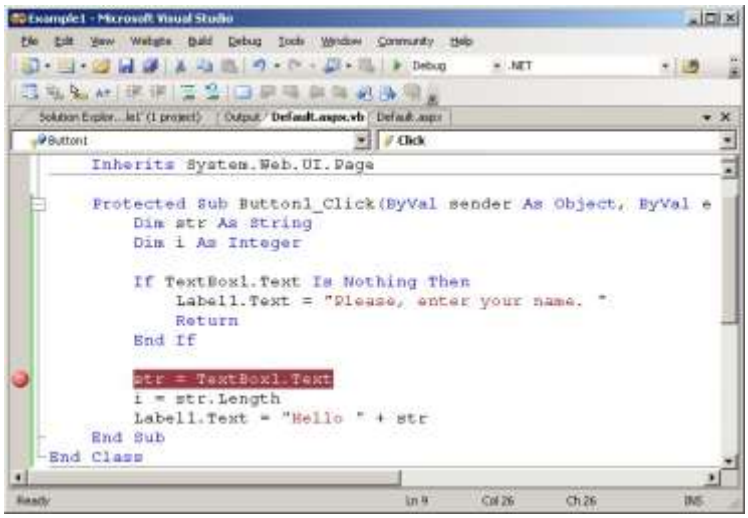
Breakpoints are places in the code, where the debugger will stop the execution of the application. After stopping, you can watch the state of the application's variables, and then you can step through your code.

To set a breakpoint, simply click once at the gray margin next (left) to the line where you want to set the breakpoint. A red circle in the left gray
margin, and a red shading over this line of code will appear, indicating that, this line is a breakpoint line.

To clear the breakpoint, just click the red circle again and it will disappear. You can also use F9 or select, on the Debug menu, click Toggle Breakpoint to set a breakpoint or to clear it.

You can think of a breakpoint as a signal to the debugger. It tells the integrated debugger to pause the execution of your application and puts it at the break mode. While in break mode, your application's data is still in memory, and you still have the ability to resume the execution at any time, but the difference here is that you can control the execution of your code step by step, line by line, watching every piece of code closely looking for violations or bugs.

Overview (Contd…)



The above figure shows a red circle indicating it is a breakpoint line

Advantages of Breakpoints

- No code required to be added to your program
- Set or disable breakpoints without changing source codes
- Pause the execution of a program at any point
- Breakpoint management is simple

Advantages of Breakpoints:
Breakpoints are not actual source code you have to add to your program. You do not type a breakpoint statement into your source window.
Because are not statements, breakpoints produce no extra code, and never changes the size of your code file.
You can set or disable a breakpoint without changing your source code.
A breakpoint gives you the ability to pause the execution of your program at a certain point you specify, and this is very important in large applications. The alternative is that you run the program line by line from the first line till the line you want to stop at, which is undesirable at all.
You can manage all your breakpoints from the 'Breakpoints' window. In this window you can enable, disable, or even delete a breakpoint without need to search for it in your source code.
To display the 'Breakpoints' window, on the Debug menu, on the Windows sub-menu, click Breakpoints.

Important Features of Breakpoints

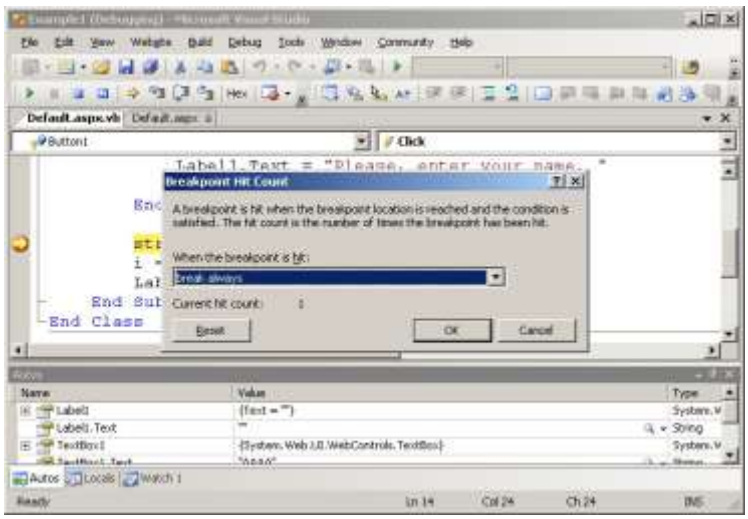
- Hit Count:
  - Specify how many times you wish to hit your breakpoint before the application breaks.
  - Useful when one has to deal with loops.
- Condition:
  - Set a condition.
    - Breakpoint is hit when this condition evaluates to TRUE.

Breakpoints have many magnificent features and properties. Using these features, you can control and modify the behavior of a breakpoint.
The two most used features are Hit Count and Condition.

Hit Count
The Hit Count property gives you the ability to specify how many times you want to hit your breakpoint before breaking the application. You can set a hit counter to stop execution after hitting the breakpoint 2 times, may be 10 times, or any number you choose. This is very useful when you are dealing with loops. Some times a bug does not appear the first time you execute a loop, access a variable, or call a function.

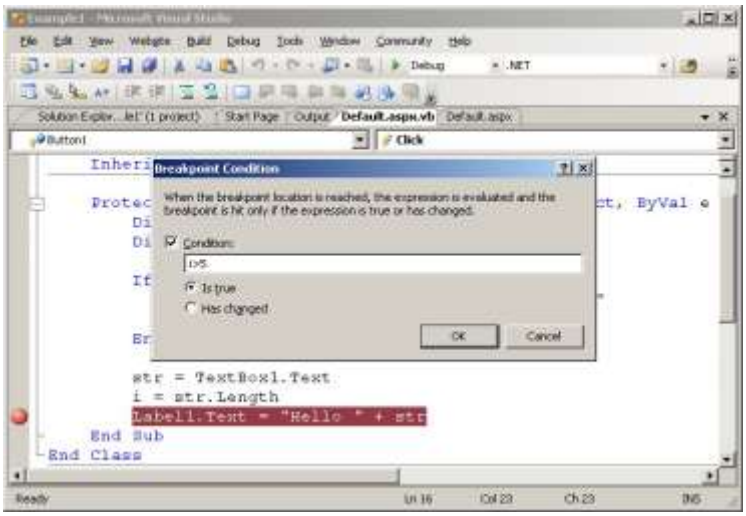
Condition Property
The Condition property gives you the ability to set a condition, that the debugger will always evaluate. When the condition evaluates to true, the breakpoint will be hit, or otherwise it will be skipped as if it does not exist at all.

Setting a Hit Count



To set a hit counter, position your mouse pointer over a breakpoint line. Right-click the right-mouse button and click Breakpoint Hit Count as shown in the above figure. You need to select a proper option from the combo box to be able to set the Hit Count.

Setting a Breakpoint Condition



To set a Breakpoint Condition, position the mouse pointer over the new breakpoint. Right-click and click Breakpoint Condition. Specify the condition in the textbox.

Demo: BreakPoints

- Adding Breakpoints
- Setting Hit Count
- Setting Breakpoints Condition
- Viewing and Changing Data

4.5: Stepping

Overview

- Following are options to step through code:
  - Step Into
  - Step Over
  - Step Out

Once you stopped your application at a given point, you can step through your code line by line, that is, execute your code line by line. The integrated debugger provides you with a number of features to help you step through your code.

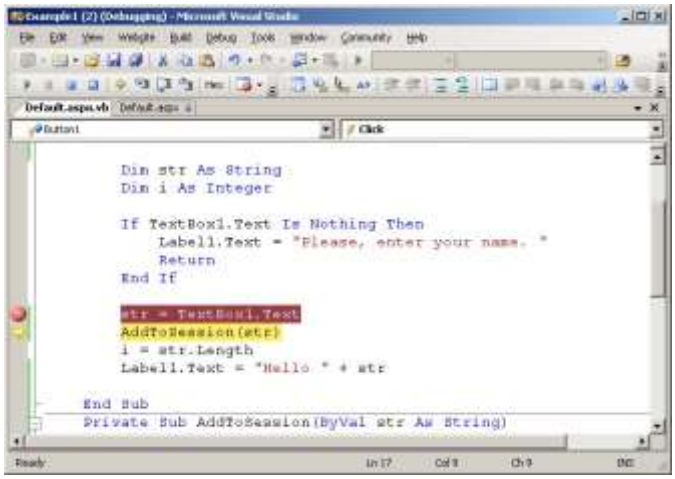
There are three commands you can use for stepping through code. These commands are:

Step Into
Step Over
Step Out

4.5: Stepping

Step Into

- Step through the code line by line
  - Press F11 or on the Debug menu, click Step Into.



Step Into
This command is to provide the ability to walk through your code while you are in the break mode. After stopping at a breakpoint line, you certainly need to go to the next line, and the line next to the next line and so on. Press the F11 key (or on the Debug menu click Step Into) that executes the step into command one can do so.

4.5: Stepping

Step Over

- Step Over differs from Step Into in the way it handles a function call
- When we use Step Over, the function is executed as a whole. No need to step through it line by line
- Press F10 or on the Debug menu, click Step Over

Step Over

You can step over line by pressing F10, or on the Debug menu click Step Over. Step Over is just like step into, but it differs in only one aspect, which is the way it handles a function call.

When you have a function call, step into will move you inside the function code line by line, which means executing the function call line by line. When you use step over, the function is executed as a whole without the need to step through its lines line by line. The highlighting indicator then moves to the next line.

Step Out

- Use Step Out when inside a function and you wish to go directly to the point from where the function is called without stepping to the end of the function
- If the control is at the top level of a program, choosing the Step Out option causes the program to resume running as normal.
- Press shifted+F11 or on the Debug menu, click Step Out

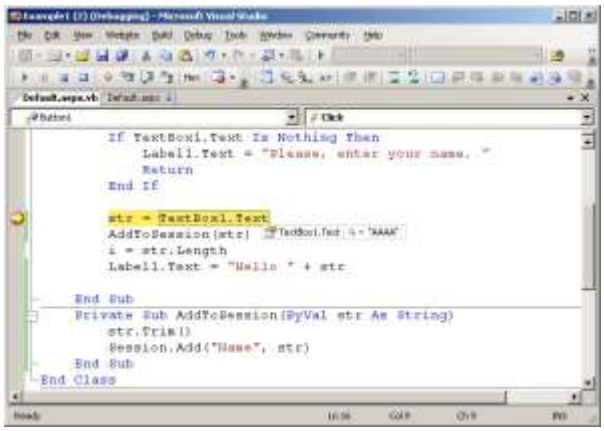
Demo: Stepping

- Demonstrating Step Into, Step Over and Step Out

4.6: Data Viewing

Overview

- View and Track data while debugging
- Edit values in break mode



The integrated debugger provides you with many different tools that you can use to view and track data while debugging. It also gives you the ability to edit your data values while you are in break mode. You can then complete the debugging process depending on the new modified values. Most of these tools function properly only in break mode.

You can watch the current value of a variable by simply placing your mouse pointer over it. An example for this behavior can be seen in the above figure. You can also change its value by right clicking the data tip. You need to choose Edit Value from the shortcut menu to achieve this. Type the new value then press enter.

4.7: Variables Window

Types of Windows

- Integrated Developer provides following Windows to display and edit application's variables and expressions:
  - Locals Window:
    - Watch variables inside your current scope
    - On the Debug menu, navigate click Windows from the Locals menu to view this window
  - Autos Window:
    - View variables in the current line of code and above it.
    - On the Debug menu, navigate click Windows from the Autos menu to view this window
  - Watch Window:
    - Watch certain variables or expressions

The integrated debugger also provides a number of windows to display and edit your application's variables and expressions. These windows are:

Locals Window
Autos Window
Watch Window

Each window has three columns: 'Name', 'Value', and 'Type'.

In the 'Locals' window you can watch the variables inside your current scope. For example, when you are inside a function or a sub. The entries inside this window are created automatically by the debugger. To display this window on the Debug menu, on the Windows sub-menu, click Locals.

The entries inside the 'Autos' window is also created automatically by the debugger. Use this window to view the variables inside the current line of code, and the line above it. To display this window on the Debug menu, on the Windows sub-menu, click Autos.

The 'Watch' window is the window you can use to watch certain variables or expressions. You can add the variable or expressions you want to watch by positioning your mouse pointer over it, then click the right mouse button and choosing 'Add Watch' from the pop up menu. You can add more than one variable. To display this window on the Debug menu, on the Windows sub-menu, click Watch 1.

Demo: Variables Window

- Trying to view values in Variables Window

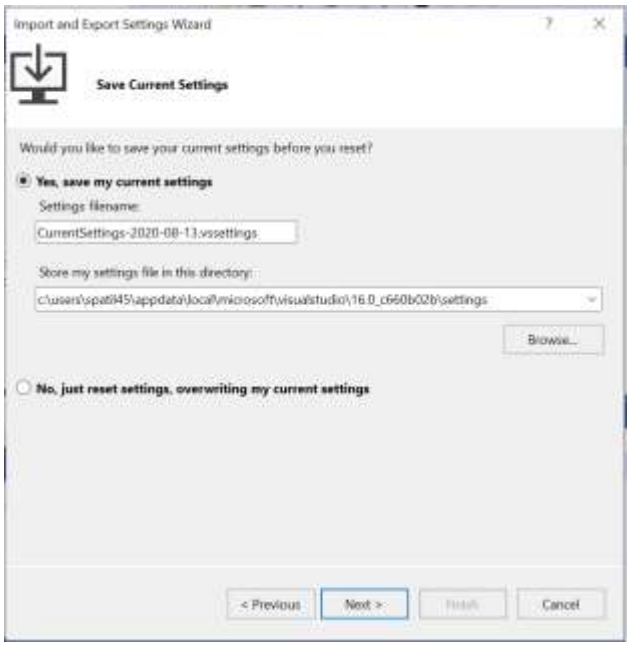
Customizing Visual Studio – Environment Settings

- When you open Visual Studio for the first time, you can optimize the development environment for the type of development that you do the most by choosing a collection of settings.
- Each collection optimizes elements such as keyboard shortcuts, window layouts, project and item templates, and command visibility.
- The following settings collections are available:
  - General
  - JavaScript
  - Visual Basic
  - Visual C#
  - Visual C++
  - Web Development
  - Web Development (Code Only)

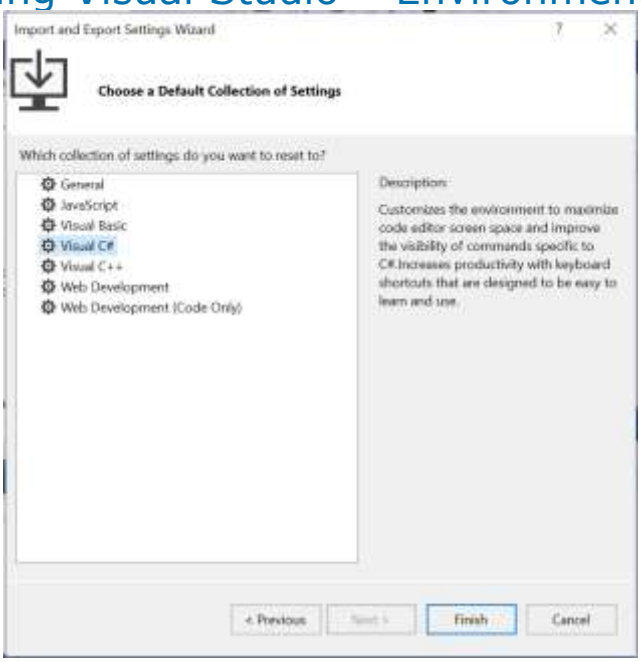
Customizing Visual Studio – Environment Settings

- To change your development settings after you open Visual Studio for the first time, follow these steps:
  - Select Tools > Import and Export Settings from the menu bar to open the Import and Export Settings Wizard.
  - In the Import and Export Settings Wizard, select Reset all settings, and then select Next.
  - On the Save Current Settings page, select either Yes or No, and then select Next.
  - On the Choose a Default Collection of Settings page, choose a collection, and then select Finish.

Customizing Visual Studio – Environment Settings



Customizing Visual Studio – Environment Settings

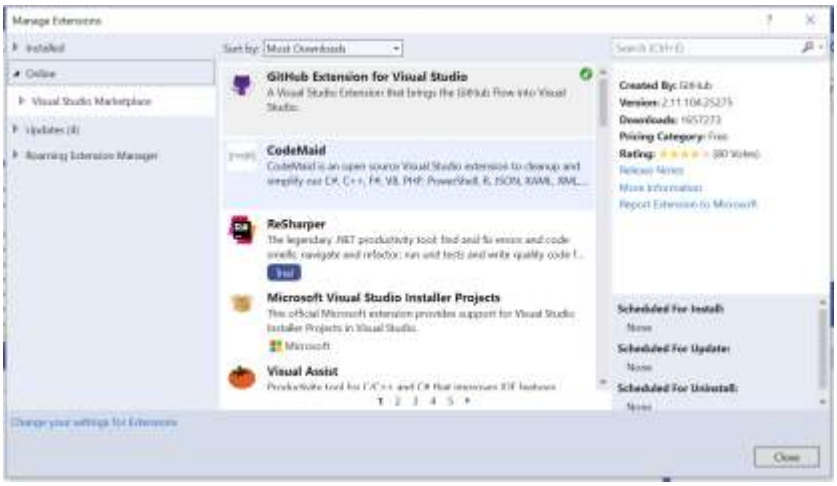


Manage Extensions for Visual Studio

- Extensions are code packages that run inside Visual Studio and provide new or improved features.
- Extensions may be controls, samples, templates, tools, or other components that add functionality to Visual Studio, for example, Live Share or Visual Studio IntelliCode
- Use the Manage Extensions dialog box to install and manage Visual Studio extensions

Manage Extensions for Visual Studio

- To open the Manage Extensions dialog, choose Extensions > Manage Extensions. Or, type Extensions in the search box and choose Manage Extensions.



The pane on the left categorizes extensions by those that are installed, those available on Visual Studio Marketplace (Online), and those that have updates available. Roaming Extension Manager keeps a list of all the Visual Studio extensions you've installed on any machine or instance of Visual Studio. It's designed to let you find your favorite extensions more easily.
Find and install extensions
You can install extensions from Visual Studio Marketplace or the Manage Extensions dialog box in Visual Studio.

To install extensions from within Visual Studio:

1. From Extensions > Manage Extensions, find the extension you want to install. (If you know the name or part of the name of the extension, you can search in the Search window.)
2. Select Download.

The extension is scheduled for install. Your extension will be installed after all instances of Visual Studio have been closed.

If you try to install an extension that has dependencies, the installer verifies whether they're already installed. If they aren't installed, the Manage Extensions dialog box lists the dependencies that must be installed before you can install the extension.

Summary

- In this lesson you have learnt:
  - Debugging techniques

Review Question

- Question 1: _____ window shows the variables in the active statement and the previous code line
  - Autos
  - Locals
  - Watch
  - Immediate