11

Debug the Application

Debugging, in application development, is a process to identify an issue or a problem using some debugging tool or IDE that provides debugging methods. It involves stepping through the code and analyzing the variables and methods and their values to pin point the exact place of the issue.

If you have been an application developer for some time now, you should have an idea about how important debugging is in the process of application development and even If you are a new developer or just starting out this chapter will help you get started with the terminologies of debugging and how to debug Xamarin applications on Visual Studio and troubleshoot other issues that might arise during the development.

In this chapter, you’ll be learning in deep about the below topics:

* Debugging a Xamarin application on Visual Studio
* Debugging and troubleshooting on Android Emulator
* Debugging Mono’s Class libraries and using debug logs and
* Debugging Git connections

**Terminologies**It’s better to first get an idea of different terminologies used in the process of debugging. These are commonly used terms and are common between all different debugging platforms.

**Bug:**A bug is a defect or a problem that is stopping the program or the application from performing it’s expected functions.

**Debug:**You might have guessed it by now, debug as the name suggests, is removing the bug from the system or program. It usually connotes finding the problem by digging into the program and resolving it after it is identified by correcting the erroneous code.

**Breakpoint**

As the name suggests, breakpoint is a point where you want to break the running application and by break we mean pause. So, it is a point in your application program’s code, where you want to pause the running application and see what’s happened or what’s happening. It very useful and a critical tool in debugging an application.

**Debugging with Xamarin on Visual Studio**

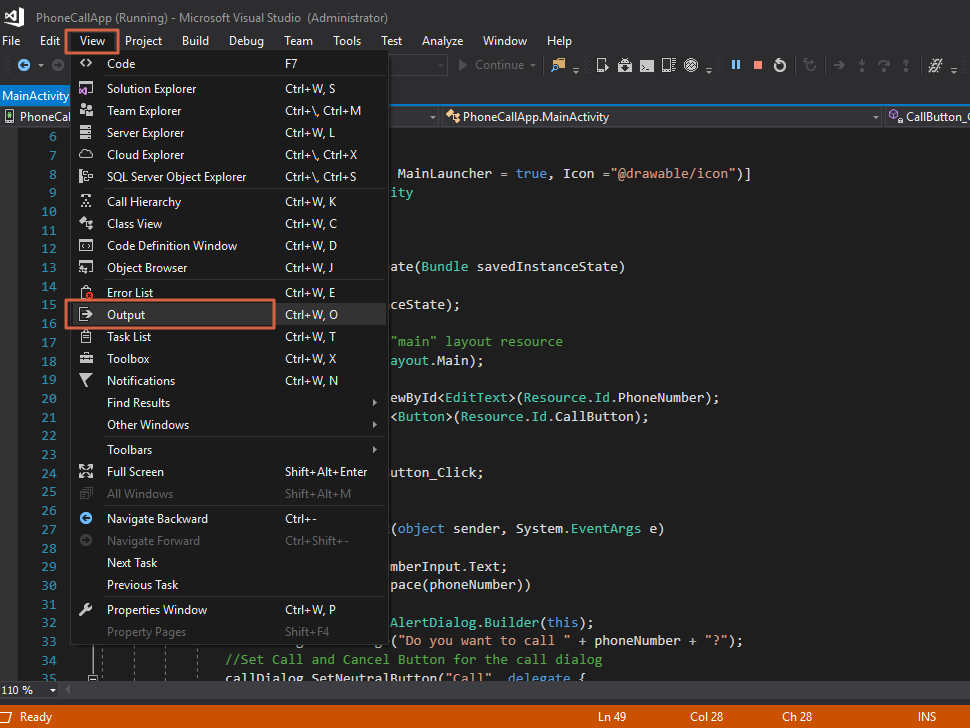
Visual Studio is a great IDE for debugging any application whether it’s a web or mobile or a desktop application. It uses the same debugger that comes with the IDE for all of them and is very easy to follow.

To keep the chapter easy to follow, we’ll be using the same Android application we developed and tested on Xamarin while debugging on Visual Studio.

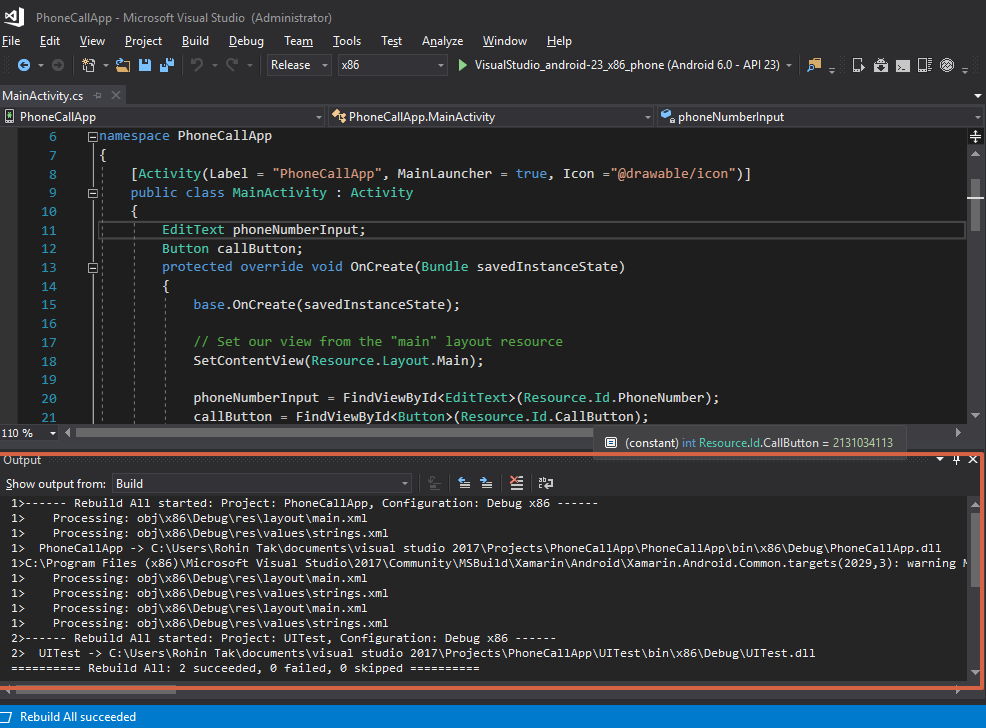
**Using Output Window**

Output window in Visual Studio is a window where you can see the output of what’s happening. To view the output window in Visual Studio:

1. Go to View and click Output



1. This will open a small window in the bottom where you can see current some useful output being written by Visual Studio. For example, this is what is shown in the output windows when we rebuild the application.

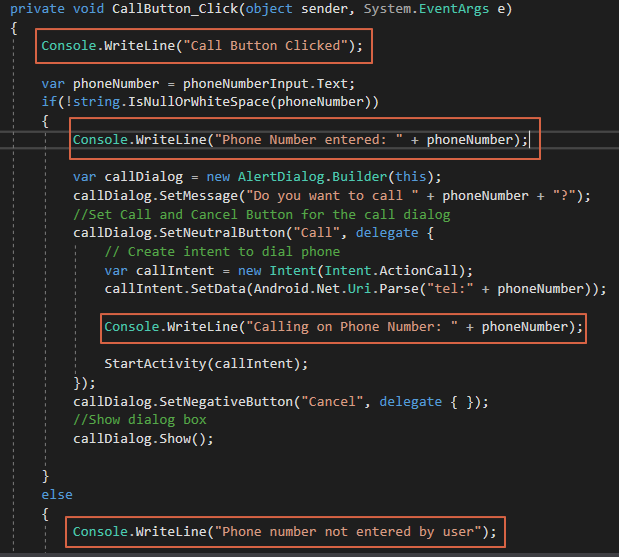


**Using Console Class to show useful output**

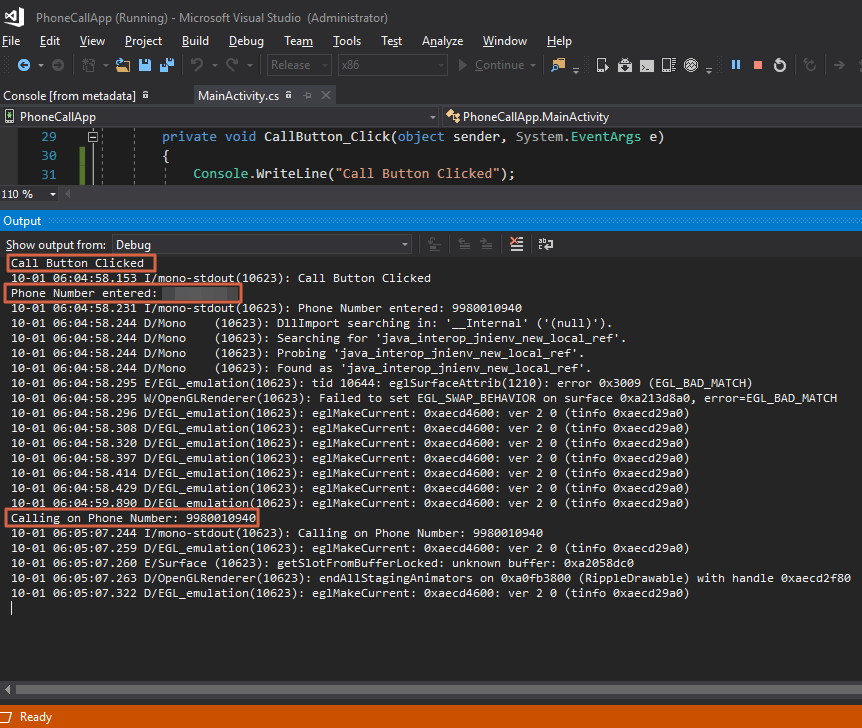
Console class can be used to print some useful step like logs into the output window to get an idea of what steps are being executed. This can help if a method is failing after certain steps, so that will be printed in the output window.

To achieve this, C# has a Console class, which is a static class. This class has methods like Write() and WriteLine() to write anything to the output window. Write() method writes anything on the output window and WriteLine() method writes the same way with a new line at the end.

1. See the below image and analyze how Console.WriteLine() is used to device the method into several steps. (It is the same Click event method that was written while developing PhoneCallApp)



1. Add Console.WriteLine() to your code same as shown in the above image.
2. Now, Run the application, perform the operation and see the output written as per your code.



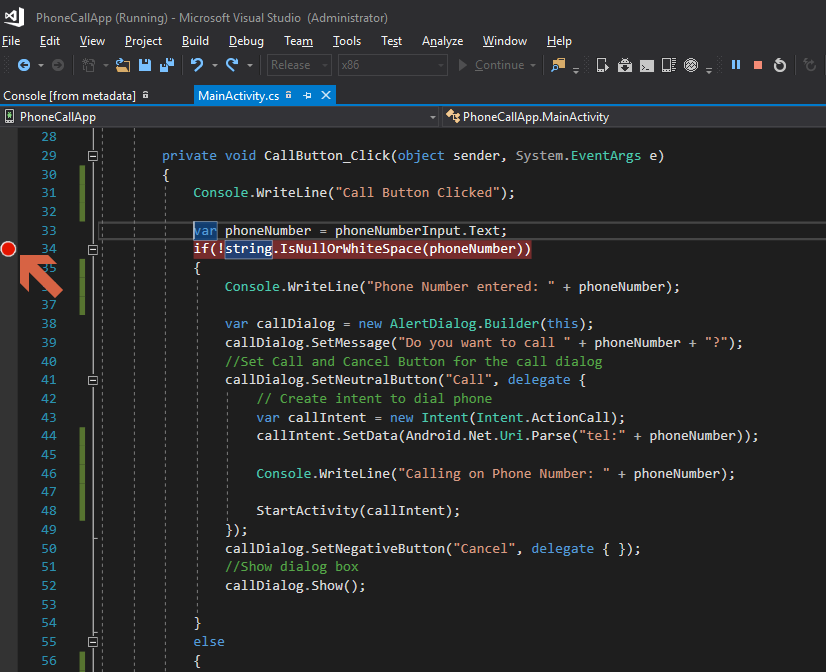
1. This way, Console.WriteLine() can be used to write useful step based outputs / logs on the output window which can be analyzed to identify the issue while debugging.

**Using Breakpoints**

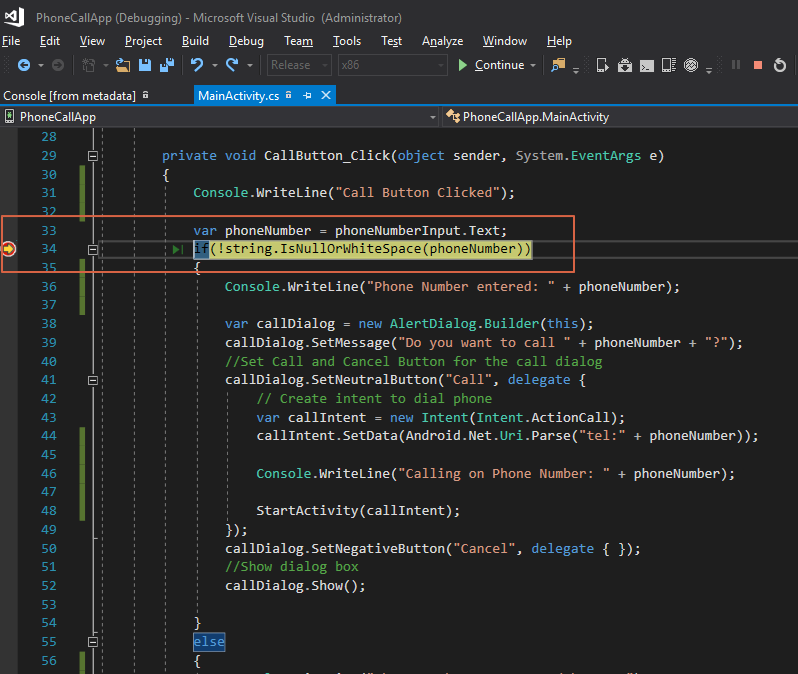
As described earlier, breakpoints are a great way to dig deep into the code without much hassle, it can help checked the variables and their values and the flow at a point or line in the code.

Using breakpoint is very simple:

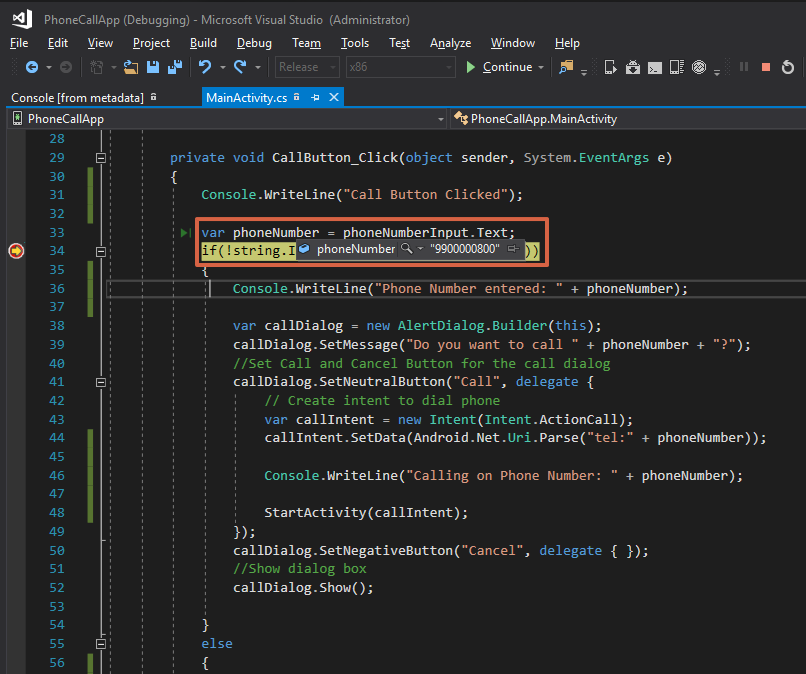
1. The simplest way to add a breakpoint on a line is to click on the margin are on the left side in front of the line or click on the line and hit F9 key.



1. You’ll see a red dot on the margin area where you clicked when the breakpoint is set as shown in the above image.
2. Now run the application and perform call button click on it, the flow should stop at the breakpoint and the line would turn yellow when it does.



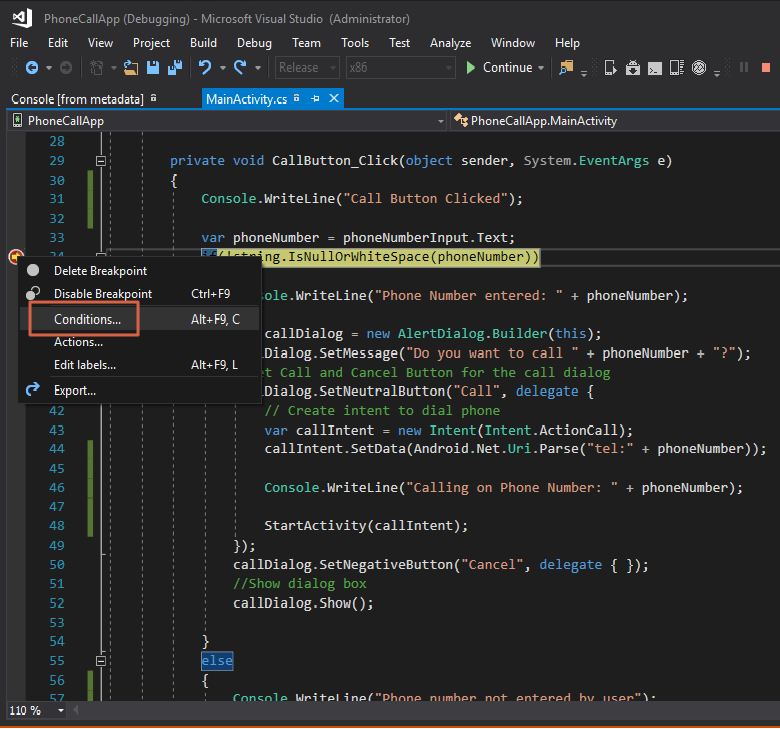
1. At this point, you can inspect the values of variable before the breakpoint line by hovering over them.



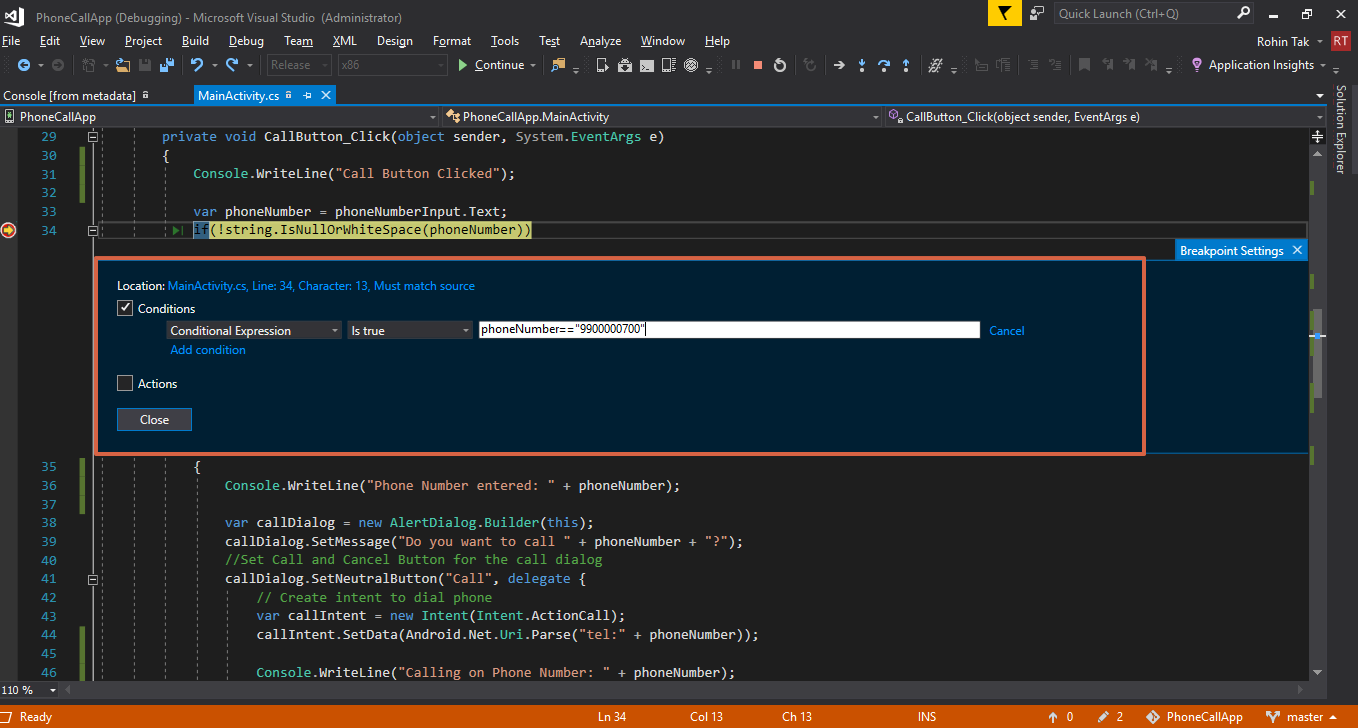
**Setting a Conditional Breakpoint**

You can also set a conditional breakpoint in the code, that is basically telling Visual Studio to pause the flow only when a certain condition is met.

1. Right click on the set breakpoint in the previous steps, and click conditions

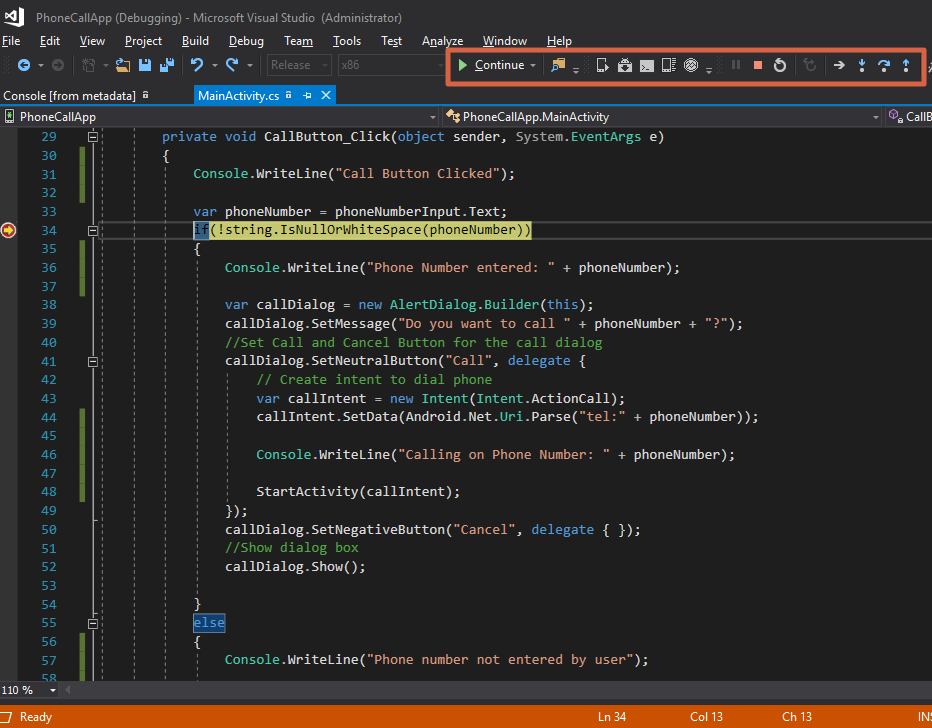


1. This will open a small window over the code to set a condition for the breakpoint. For example, In the below image, condition is set to when phoneNumber == “9900000700”.  
   So the breakpoint will only be hit when this condition is met, otherwise it’ll not be hit.



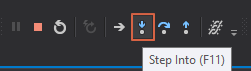
**Stepping Through the Code**

When a breakpoint has been reached, the Debug tools enable you to get control over the program’s execution flow. You’ll see some buttons in the toolbar, allowing you to run and step through the code.



You can hover over these buttons to see their respective names.

1. **Step Over (F11)** – This will execute the next line of code. If the next line is a function call, step over will execute the function, and will stop at the next line of code *after* the function.



1. **Step Into (F10)** – This will also execute the next line of code. If the next line is a function call, Step Into will stop at the first line of the function, allowing you to continue line-by-line debugging of the function. If the next line is not a function, it will behave the same as Step Over.

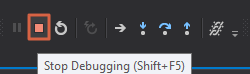


1. **Step Out (Shift + F11)** – This will return to the line where the current function was called.



* **Continue** – This will continue the execution and run until next breakpoint is reached.



* **Stop** – This will stop the debugging process.  
    
  

**Using Watch**

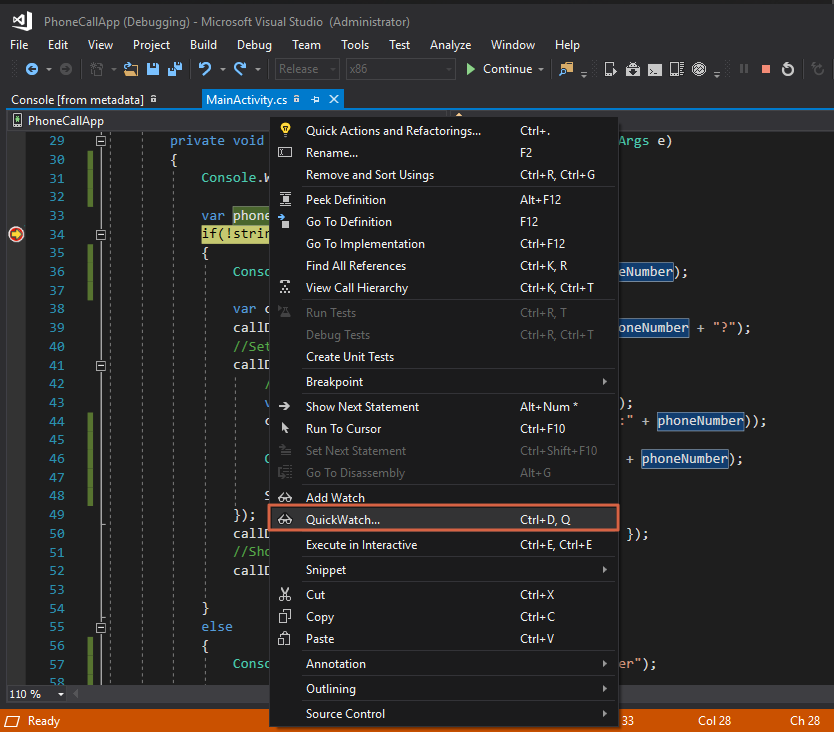
Watch is a very useful function in the debugging, it allows us to see the values, types and other details related to variables and evaluate them in a better way then hovering over the variables.

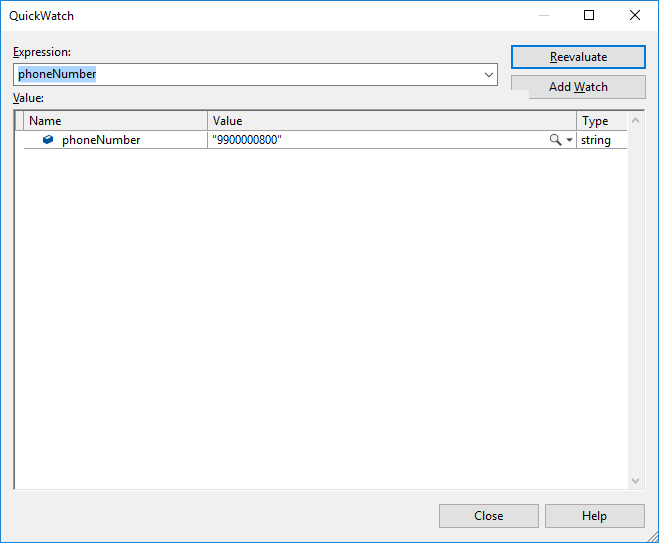
There are 2 types of watch tools available in Visual Studio:

**Quick Watch**

Quick watch is similar to watch but as the name suggests it allows us to evaluate the values at the time.  
To use Quick Watch in Visual Studio:

1. Right click on the variable you want to analyze and click Quick watch.



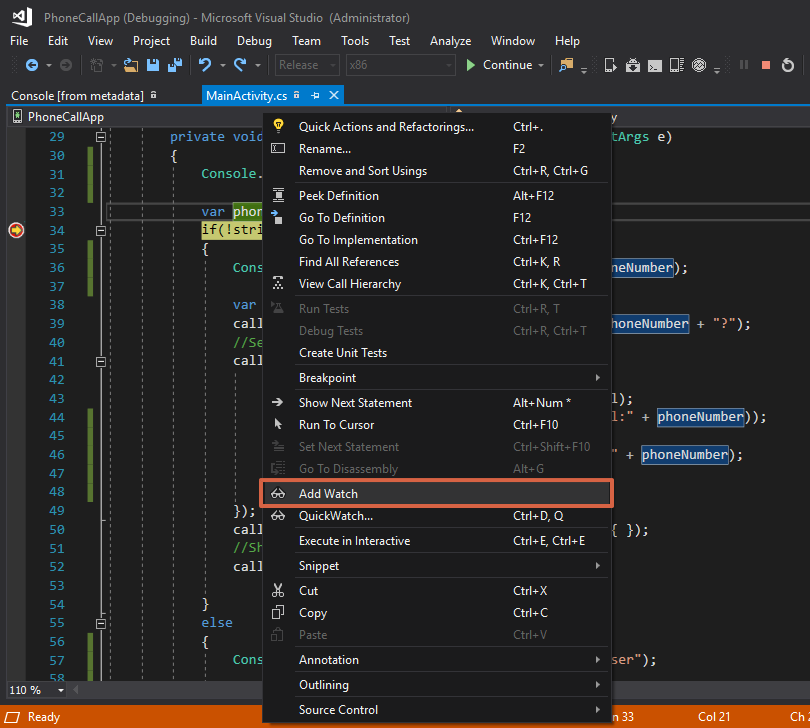
1. This will open a new window where you can see the type, value and other details related to the variable.  
     
   
2. This is very useful when a variable has a long value or string that cannot be read and evaluated properly with just hovering over the variable.

**Add Watch**

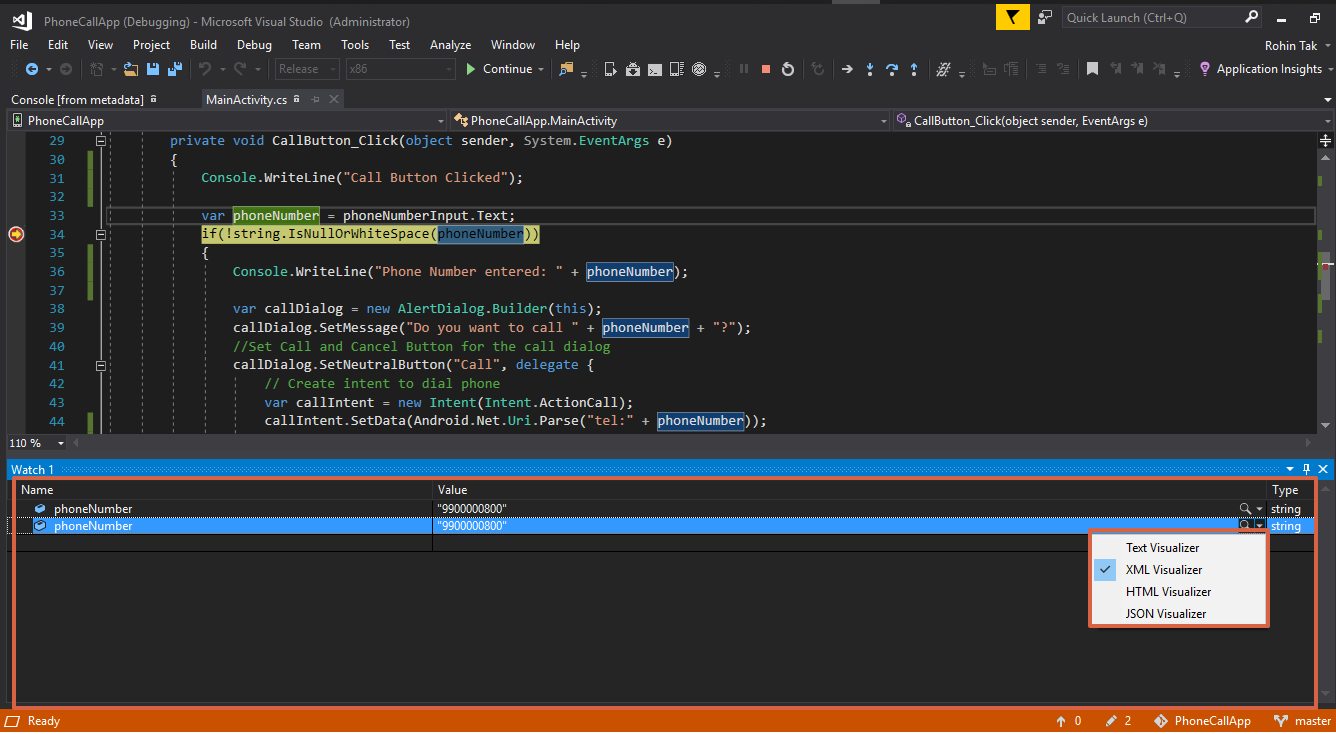
Add watch is similar to quick watch but it is more useful when you have multiple variables to analyze and looking each variable’s value can take a lot of time.

To add watch on variables:

1. Right click on the variable and click add watch.



1. This will add the variable to the watch and show you it’s values always, and reflect anytime it changes at runtime.
2. You can also see these variable values in a particular format for different data types, like you can have an XML value shown in XML format of a Json object value shown in Json format.

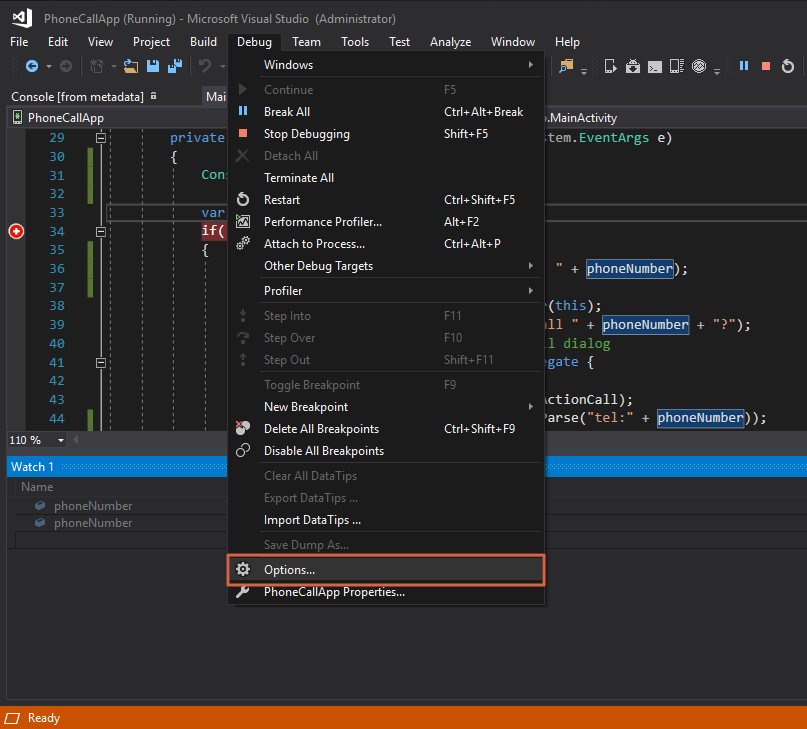


1. It is a life saver when you want to evaluate variable’s value in each step of the code and see how it changes with every line.

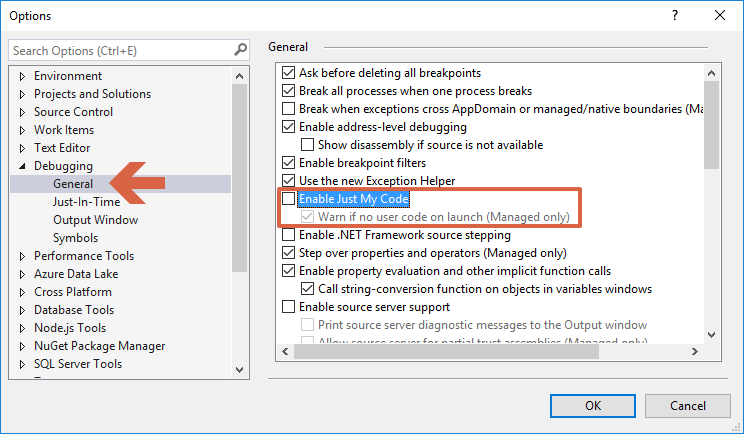
**Debugging Mono’s Class Libraries**

Xamarin ships with the source code for Mono's class libraries, you can use this to debug the Xamarin (formerly known as Mono) source code.

1. To be able to use this option go to Debug -> Options



1. Then go to General - > and uncheck the option “Enable Just My Code” and click OK.



1. Once this is disabled, we can “step into” Mono’s class libraries and debug them.

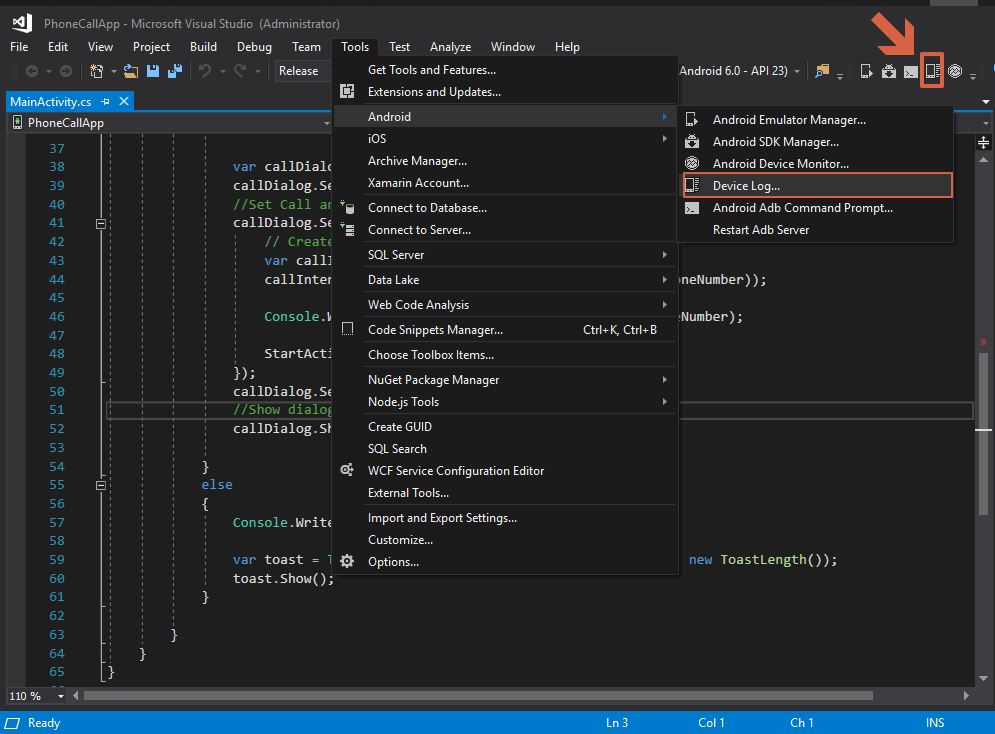
**Android Debug Log**

As mentioned in previous sections in this chapter, we have seen how to use Console.WriteLine() method to write some output steps while debugging in Visual Studio.

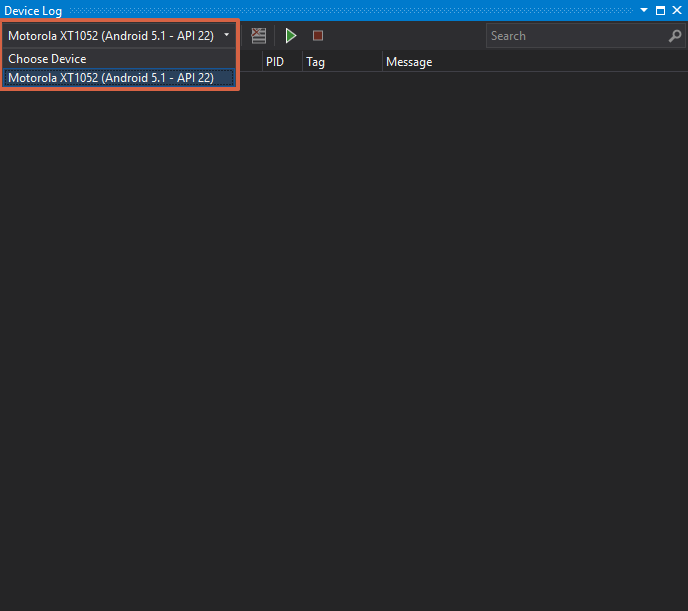
However, on a mobile platform like Android there is no console and it is only available for us during debugging in Visual Studio. Android devices provide a log that you can utilize while coding Android apps. This is also known as “logcat” due to the command used to retrieve this log.

To access this from Visual Studio, follow below steps:

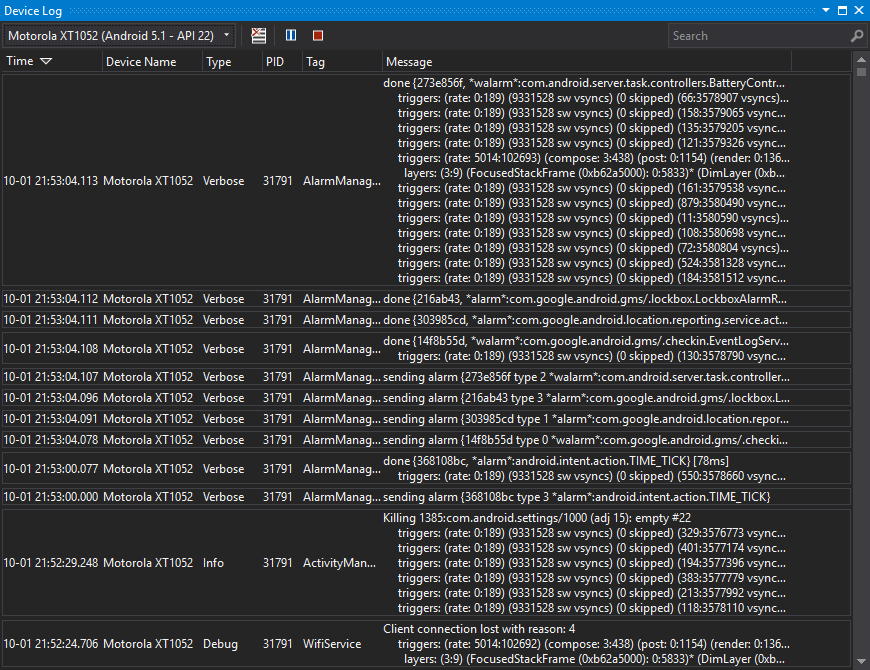
1. Either you can directly click on the Device Log (logcat) icon from the Android Tools in the toolbar or you can go to Tools -> Android -> Device Log.



1. This will open a new window where you can choose your device application is running on. The application needs to be running on a physical device to be clear, since it is debugging when app is running on the device and log is provided by Android devices.



1. Select the device form the dropdown on which application is running.
2. When the device is selected, it automatically starts to add log entries from a running app in the table. Switching between devices will stop and start the device logging.

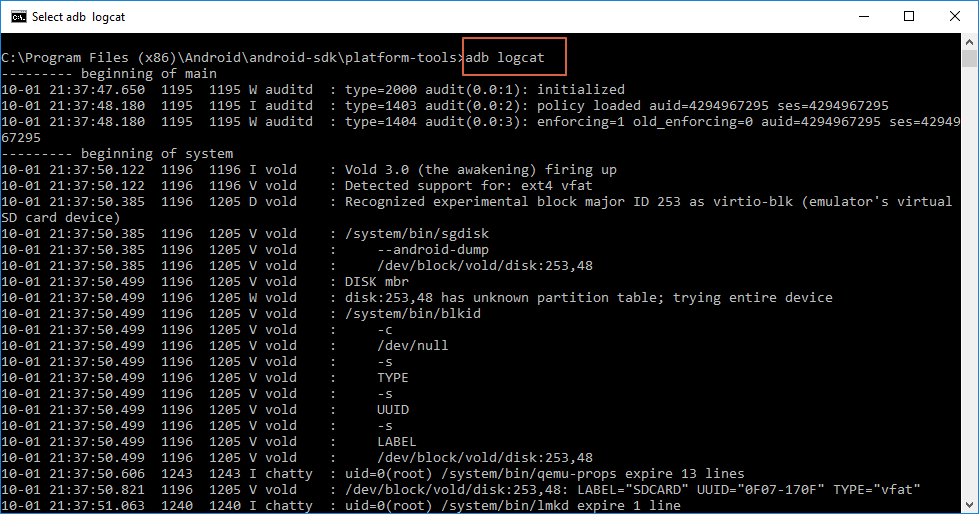


**Accessing logcat from the Command Line**

Another option to view the debug log is via the command line.

* 1. Open a console window and navigate to the Android SDK platform-tools folder (like C:\Program Files (x86)\Android\android-sdk\platform-tools).
  2. If only one device is attached, the log can be viewed with:

*$ adb logcat*



* 1. If more than one device is attached, then the device must be identified.  
     For example *adb -d logcat* shows the log of the only physical device connected, while adb -e logcat shows the log of the only emulator running.

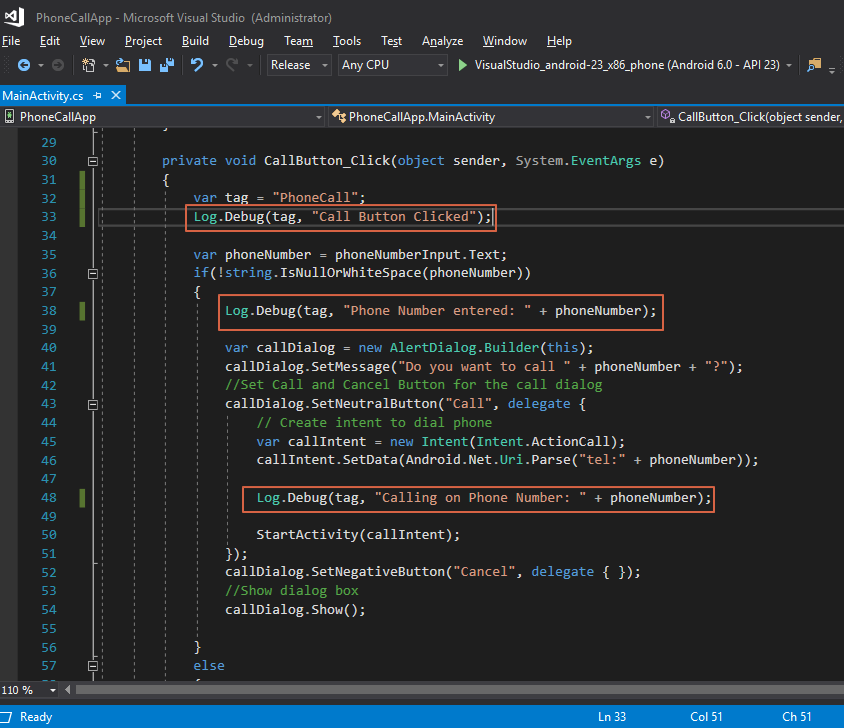
**Writing to the Debug Log**

You can log messages to Debug log using **Android.Util.Log** class. It has different level of logging:

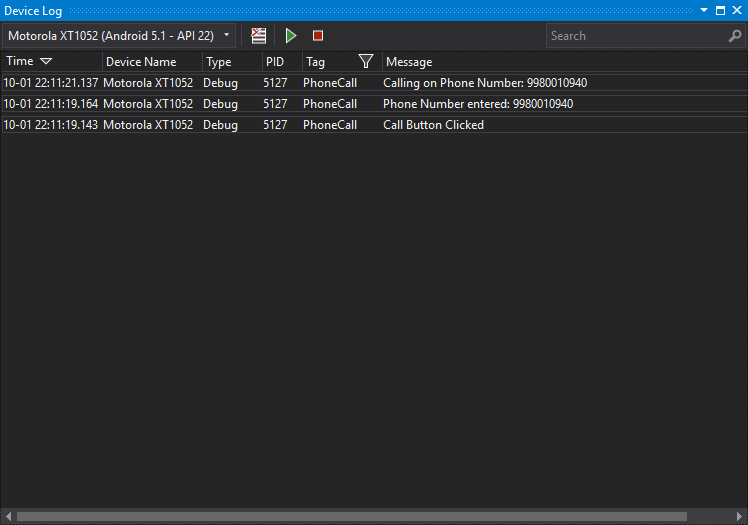
* **Info**
* **Debug**
* **Warning**
* **Error**

All these levels are self explainatory.

1. Let’s replace all the Console.WriteLine() written in previous section with Log.Debug() to write the logs into logcat.



1. Go to the logcat (Device Log) window and filter tags with the tag given in the code to see only the logs we have written in the code. In this case the tag would be “PhoneCall”

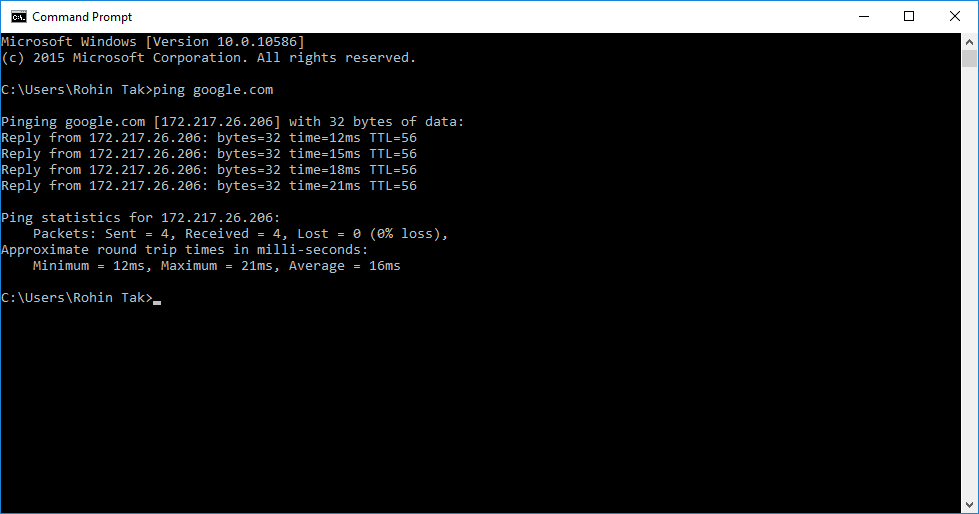


1. This is a very simple and straight forward way of debugging and monitoring an application running on the physical device.

**Debugging GIT connections**

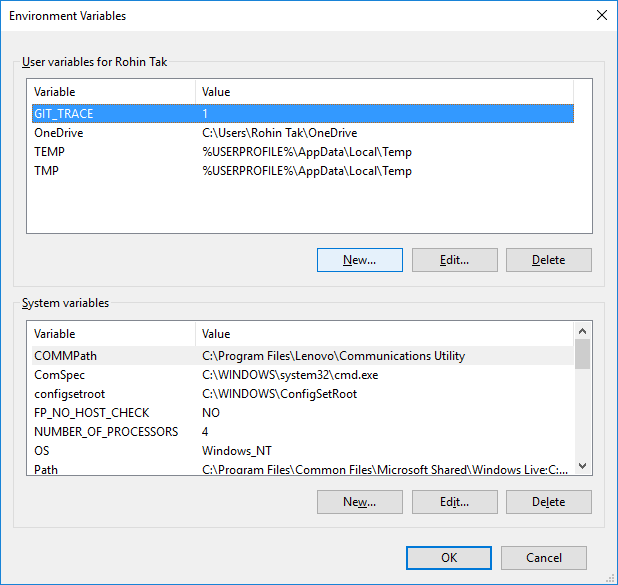
Git is essential for saving code into repositories and there can sometimes when it is not working as expected, just like our application. To debug Git when you are not able to fetch or clone code from the repo, try below steps:

1. **Check your connectivity**This would be the first thing to check when you are facing any issues with git. It might be possible that your connection is not as you think it is. Ping any public domain site such google.com to check your connectivity.

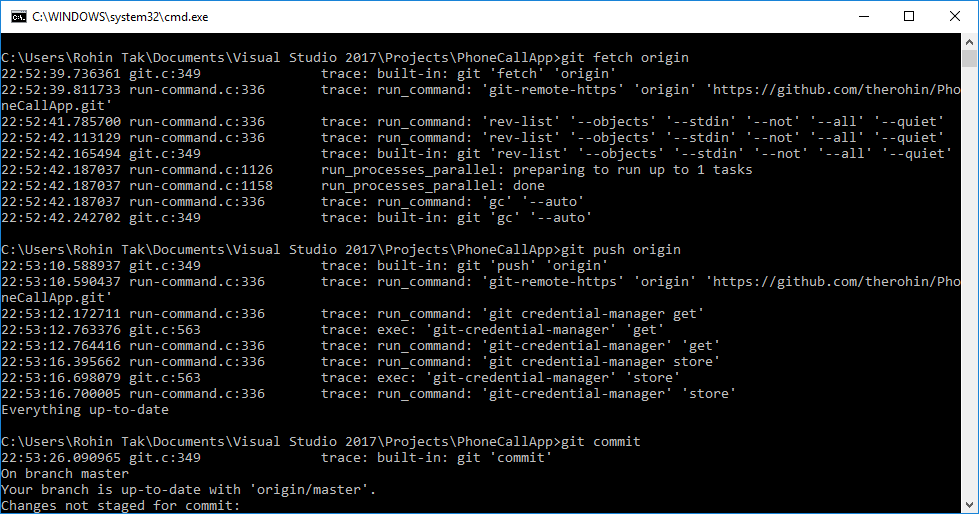


If you get a response like above and are able to ping successfully that means your connection is totally fine.

1. **GIT\_TRACE:**This configuration option gives us a more verbose trace to the git network connections and all the internal commands it goes through.  
   Type your git command with GIT\_TRACE = 1 and it should give you a good detailed verbose trace for the same.
2. Add a new environment variable named GIT\_TRACE and give it value as 1



1. Run Git command and get the detailed verbose to identify the issue.



**Summary**

In this chapter, we covered debugging in different ways and learned to use tools available in Visual Studio and Xamarin (Android) for debugging. This chapter also explains Android Device Log also known as logcat to read and write logs when debugging application on a physical device.   
In coming chapter we’ll be going through the entire development, testing and debugging process with some case studies.