# Debugging and Profiling in Visual Studio 2015

05 January 2016
by Manuel Meyer

Visual Studio 2015 has some greatly improved features for debugging and troubleshooting .NET applications. Probably the most significant one is in the diagnostic tools hub that allows profiling while debugging. Manuel Meyer explains all these useful new features

Visual Studio 2015 introduces some interesting new features that are designed to assist the troubleshooting of .NET applications. In this article I will be showing how these new debugging and profiling features can be used.

The new features are:

- PerfTips
- Edit & Continue for x64 based systems
- Lambda Expression Evaluation in the Watch and Immediate Window
- Diagnostic Tools Window
- Live Visual Tree and Live Property Explorer
- Diagnostic Tools Hub (Performance and Diagnostics Hub)

## PerfTips

Debugging tips - the little helpers that pop up during a debugging session and show the values of variables - have been around for a long time. Every developer relies on them to simplify the debugging process. PerfTips follow the same objective, an easier and more productive debugging session. In contrast to debugging tips, PerfTips show timing information while stepping through code. Before PerfTips, it was tedious to collect line-by-line timing information because it involved inserting instrumentation code such as the **System.Diagnostics.Stopwatch** class. Because PerfTips are part of Visual Studio 2015, there is no need to insert custom code any more. PerfTips measure the time that passes between two debugger breaks. It is irrelevant whether you use 'Step Into', 'Step Over' or 'Run to Cursor' to measure a single instruction or an entire block of code. The measured timespan is not only displayed in the PerfTip, but also recorded as a list in the Diagnostic Tools window. (More on the Diagnostic Tools Window later). The PerfTip in Figure one indicates that we lost over one second while stepping over the **BuildOpenMenu ( )** method.

```
public MainWindow()
{
    InitializeComponent();

    family.CurrentChanged +=new EventHandler(People_CurrentChanged);

    // Build the Open Menu, recent opened files are part of the open menu
    BuildOpenMenu();

    BuildSkinsMenu(); ≤ 1 357ms elapsed

    // The welcome screen is the initial view
    ShowWelcomeScreen();
}
```

**Figure 1: The PerfTips shows that we lost 1.357 seconds (Code courtesy of Family.Show [1])**

## Edit & Continue for x64 Based Systems

The 'edit and continue' functionality has been around for several years, but before Visual Studio 2015, it only worked if the debugger was attached to a 32bit process. With 'edit and continue', code can be modified during a debugging session. The modified code gets recompiled in the background and is immediately effective. The benefits of this are obvious. With 'edit and continue', it is possible to execute a line of code, inspect the result, modify the code, move the next statement cursor to a location before the modified line and step over it again. A restart of the debugger is not required.

In Visual Studio 2015, 'edit & continue' is available for both 32 and 64 bit processes. However, 64bit edit & continue requires .NET framework 4.5.1 or above.

# Lambda Expression Evaluation in the Watch and Immediate Windows

Visual Studio 2015 supports the evaluation of lambda expressions in the debugging windows. This is a convenient feature when looking at collections during a debugging session. Let's imagine a list of people that contains 50000 entries and the developer wants to check if the person with last name "Meyer" is in the list. Previously, the only possible way of doing this was to add the collection to a quick watch window, expand its children and scroll through the huge list until the name is found. If the list contains a large number of items, this approach is painfully tedious. Fortunately, Visual Studio 2015 allows lambda expressions in the debugging windows. The person can be found with a simple LINQ statement (see figure 2).

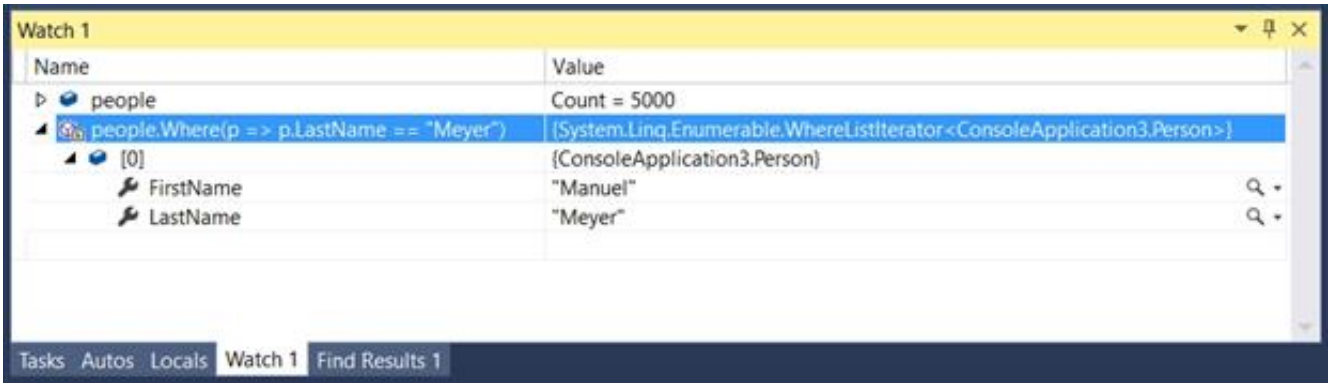


**Figure 2: A LINQ statement with a lambda expression in the watch window**

# The Diagnostic Tools Window

The Diagnostic Tools Window allows to apply profiling tools to your debugging work. It can be opened by clicking '*Show Diagnostic Tools'* from the *'Debug'* menu. The window shows sampling data for CPU and memory usage during an active debugging session. In addition to CPU and memory, markers in the graph also indicate garbage collector activity (See figure 3). In earlier versions of Visual Studio, this data could only be recorded by means of a profiling session. Instead of repeating the steps of running a profiling session and analyzing the results over and over again, the Diagnostic Tools Window will display the data in real time while the debugging session is active. A developer can immediately notice spikes in either CPU, memory or garbage collections and eventually correlate them to the code fragment being debugged or to an action taken in the user interface.

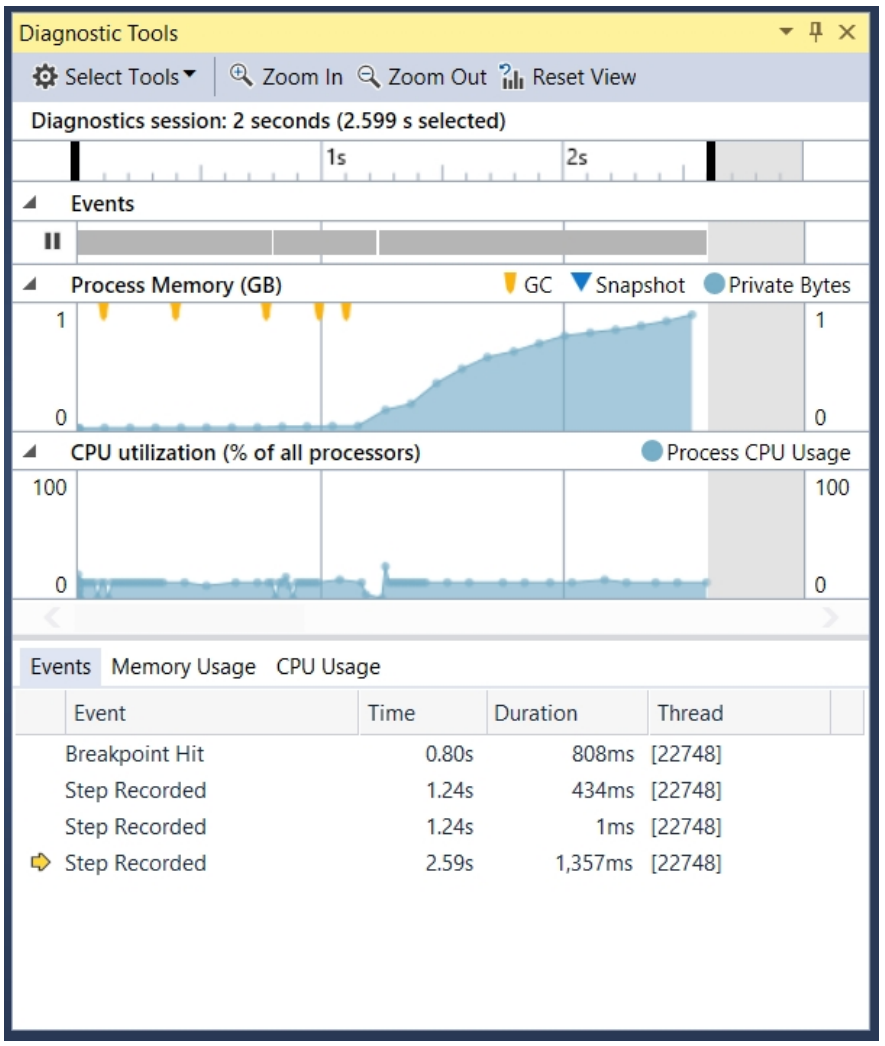


**Figure 3 : The diagnostic tools window showing cpu utilization, memory consumption, garbage collection and PerfTip data**

Another great feature is the list of PerfTips. Every time the debugger breaks and displays a PerfTip, it adds the measured time to a list in the diagnostic tools window. The list provides an overview of the measurements and allows you to jump to the corresponding line of code.

The benefits of the performance and diagnostics window go far beyond showing CPU utilization and memory consumption. A snapshot of the managed memory can be recorded by the click of a button. Once more than one snapshot is recorded, the window automatically calculates the difference in number of objects on the heap and their total size in bytes. Both values are displayed as links. When clicked, these links show a detailed list of all the objects present in memory. The snapshot feature is a great tool for tracking down memory problems such as memory leaks. Keep in mind that this tool is fully integrated into the familiar debugging experience and does not even require a restart of the debugger.

# The Live Visual Tree and Live Property Explorer

The Live Visual Tree and the Live Property Explorer are tools that specifically target Windows Presentation Foundation (WPF) applications and Universal Windows Apps. They allow the inspection of a running application immediately by looking at the visual tree. The visual tree is the internal representation of the user interface and contains all the visual elements of the application. This experience is very similar to the web developer tools used in browsers by using the ‘inspect element’ command. The Live Visual Tree allows a developer to select an item on the user interface by clicking on it. The element and its properties can be inspected and modified. Any changes immediately show up in the running application. Before Visual Studio 2015, this functionality could only be achieved by using third-party products such as Snoop [2] or WPF Inspector [3].

Figure 4 shows an example of the live visual tree of a WPF application. The left side of the image is the Family.Show application, a WPF reference implementation developed by Vertigo and available on Codeplex [1]. The image of Prince Charles is selected and the Live Visual Tree on the right hand side shows the details. The Window conveniently shows the entire structure of the application from the **MainWindow** class down to the image control that has been selected. The column on the right side shows the number of child visuals for each element.

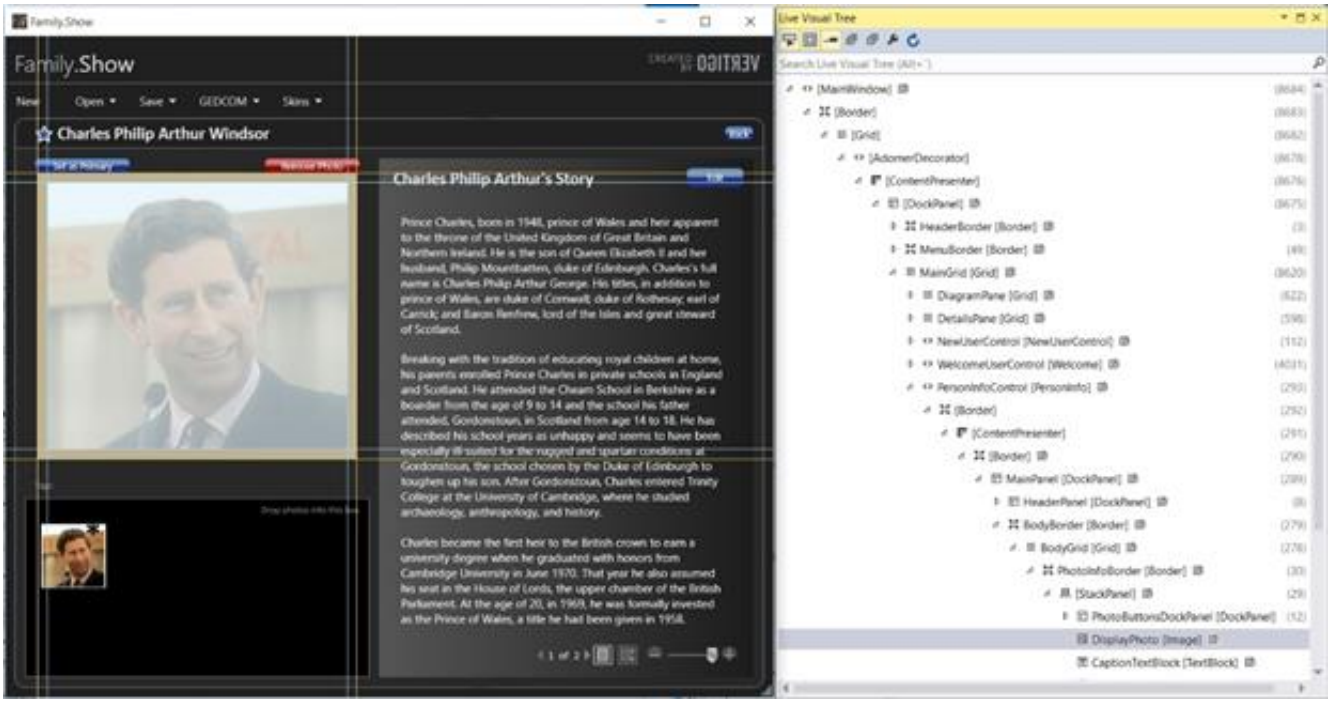


**Figure 4: Inspecting the Family.Show[1] application with the Live Visual Tree**

A right-click on an item inside the Live Visual Tree opens the context menu. From there a developer can either click ' *Go to source* ' or ' *Show Properties*'. ' *Go to source*' opens the XAML file that contains the definition of the element. ' *Show Properties* ' opens the Live Property Explorer.

The Live Property Explorer (figure 5) shows all the properties and their values. An interesting feature is the grouping. All the properties are grouped by the source of their values. This helps in finding out where a runtime property value comes from. The example shows default, computed, inherited and local values plus values that come from a XAML style definition. The values in the Live Property Explorer can be changed in order to observe the effects that the changes have on the running application.

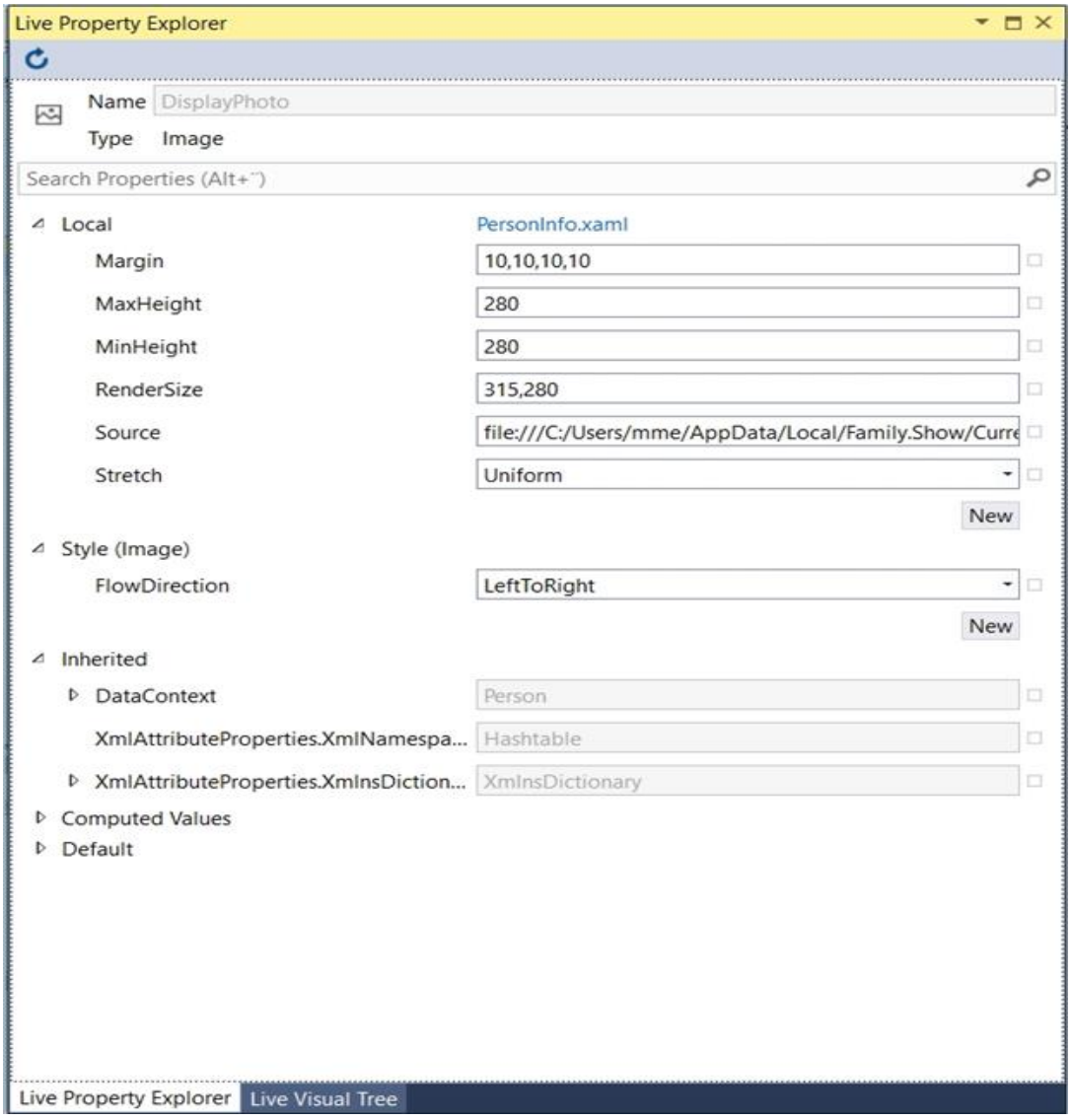


**Figure 5: The Live Property Explorer in action**

# The Diagnostic Tools Hub

The Diagnostic Tools Hub was introduced in the 2013 version of Visual Studio. It can be found under '*debug*' -> '*Start Diagnostic Tools without Debugging*' and serves as a central starting point for all the tools related to performance and diagnostics. The hub follows a simple and clear concept. It provides a large number of small and simple tools that record and display only one measurement such as CPU utilization or memory consumption. All the tools follow the same process of starting the application, recording data, calculating results and displaying the data in a Visual Studio window. The tools provide the same structure for the output data - a timeline visualization and details for every recorded measurement. The timeline shows a bar or line chart of the main measurement and may include custom markers for special events or data points. The detail view is different for each tool and presents the recorded data in more details. It allows the developer to drill down to the level of a single measurement and shows summary information in the form of a pie chart. Since the tools have a standardized output format it is possible to run multiple tools in the same session and get a combined result view. An example is to combine the user interface thread activity with CPU utilization data.

In the beginning, the tools were primarily focused on Windows Store Apps. Since the tools are, by design, relatively small and simple, they are evolving very fast and so modifications or new tools in Visual Studio updates are very common. In Visual Studio 2015, many of the tools have been extended to support other technologies such as the Windows Presentation Foundation.

The old Visual Studio Profiler, which provides CPU and memory sampling and instrumentation, does not fit into the concept of small diagnostic tools with a standardized output. For backwards compatibility it has, nonetheless, been integrated into the Performance and Diagnostics Hub. The idea is that the features within the old Profiler will slowly transform into isolated tools that are available in the hub and match its underlying idea.

In Visual Studio 2015 the hub contains the following set of tools:

- Application Timeline: Breaks down UI thread activity for XAML based applications

- CPU Usage: Classic CPU sampling
- GPU Usage: Shows graphics card instructions for DirectX applications
- Memory Usage: Shows memory consumption to identify memory leaks
- Performance Wizard: The classic Visual Studio Profiler
- Energy Consumption: Shows estimated energy consumption for mobile applications
- HTML UI Responsiveness: Breaks down UI thread activity for HTML based Apps
- JavaScript Memory: Investigates memory usage in HTML Apps
- Network: Profiles network traffic

Figure 6 shows the overview of the recording of a WPF application-startup that used the Timeline and CPU Usage tools. The Timeline tool records the utilization of the UI thread and the visual throughput in frames per second (FPS). It breaks down the UI thread activity into parsing of XAML, layout, rendering, I/O, application code and other XAML activities. Colors in the histogram help to identify those areas that require deeper investigation. The CPU utilization graph shows the CPU utilization during the session. The CPU runs at about 12.5 percent which corresponds to the UI thread being a 100% busy on an eight core machine. By selecting a range in the timeline, the details view can be filtered to an interesting section for further investigation.


**Figure 6: A recording of application startup with the timeline and CPU usage tools**

Figure 7 shows the 'details' view of the same recording. Every event is categorized and displayed by a bar whose length matches the duration of the event. It is very easy to see how much time was spent on parsing, layout, reading and writing to disks or garbage collection. Some of the events, such as the red layout event on the bottom, can be expanded to show more detail. In case of a layout operation, the inclusive and exclusive times for every single item in the visual tree are displayed.

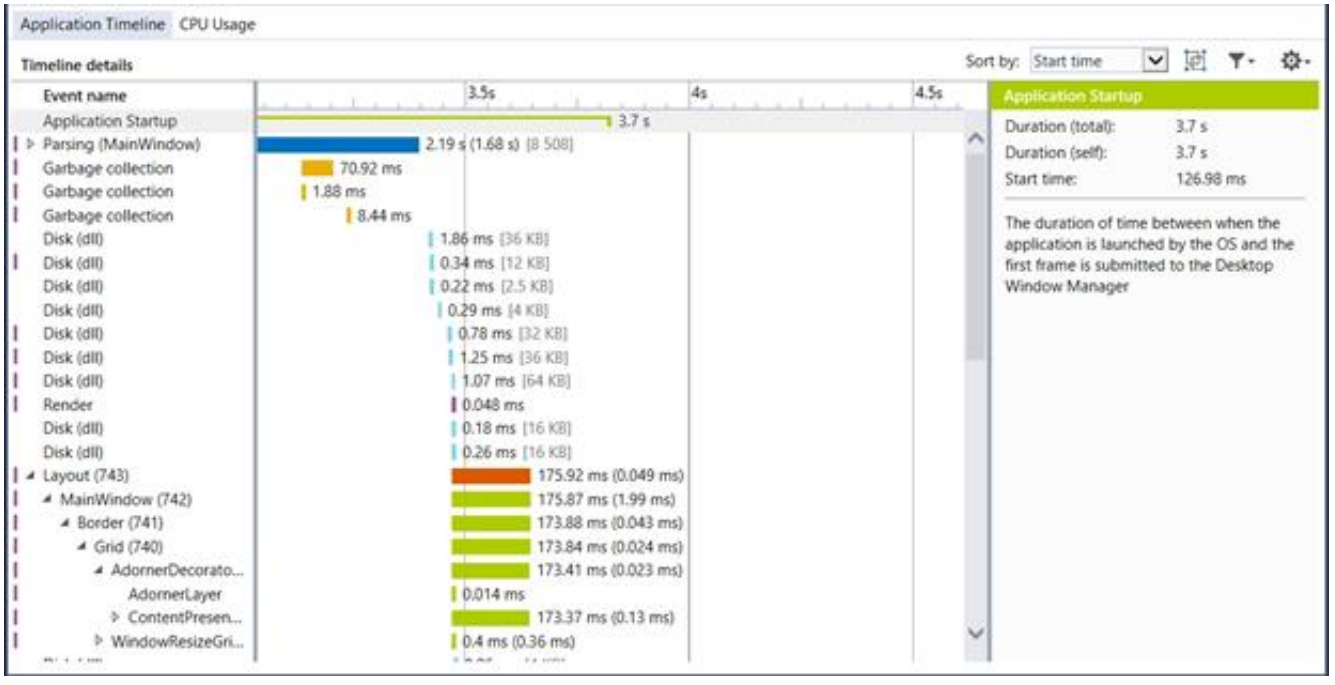


**Figure 7: The details view of the timeline recording**

# Summary

Visual Studio 2015 comes with a many great debugging features that can help developers to be more productive. It is worth noting that all the tools that are presented in this article can be found in every edition of Visual Studio, down to the free Visual Studio Community Edition [4]. There are some tools for debugging such as PerfTips or the Live Visual Tree Explorer and others for profiling, like the Timeline and CPU Usage tools from the Performance and Diagnostics Hub. A most significant feature is the Diagnostic Tools Window. For the first time in Visual Studio, a profiling tool can be used while debugging. Visual Studio 2015 integrates application profiling with the comfortable debugging experience that developers use every day.

References:

1. Family.Show WPF Reference Application on Codeplex: https://familyshow.codeplex.com/
2. Snoop – The WPF Spy Utility: https://snoopwpf.codeplex.com/
3. WPF Inspector: http://www.wpftutorial.net/Inspector.html
4. Visual Studio Community: https://www.visualstudio.com/en-us/products/visual-studio-community-vs.aspx