# New Visual Studio 2013 Diagnostic Tools

## Overview

In this lab, you will learn about some of the new diagnostic tools that were introduced with Visual Studio 2012 updates, as well as the new diagnostic tools introduced in Visual Studio 2013. You will also be introduced to the enhanced asynchronous debugging features found in Visual Studio 2013.

## Objectives

In this hands-on lab, you will learn how to do the following:

- Use the Performance and Diagnostics Hub
- Use the UI Responsiveness Tools for JavaScript and XAML Windows Store applications
- Use the Energy Consumption Tool
- Analyze JavaScript Memory Usage
- Create and Analyze Managed Memory Dumps
- Use the enhanced asynchronous debugging features in Visual Studio 2013

## Prerequisites

The following are required to complete this hands-on lab:

- Windows 8.1
- Microsoft Visual Studio 2013 (with Update 2 RC applied)

## Notes

Estimated time to complete this lab: 60 minutes.

Note: You can log into the virtual machine with user name "User" and password "P2ssw0rd".

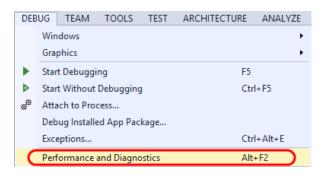
Note: This lab may make references to code and other assets that are needed to complete the exercises. You can find these assets on the desktop in a folder named **TechEd 2014**. Within that folder, you will find additional folders that match the name of the lab you are working on.

## Exercise 1: Introduction to Performance and Diagnostics Hub

In this exercise, you will you will learn about the new Performance and Diagnostics Hub in Visual Studio 2013. The new hub brings together existing tools into one location, and makes it easier to see what tools are available for the current project based on the current language, application type, or platform. Future updates will also be surfaced here, increasing the discoverability for developers.

#### Task 1: Introduction to Performance and Diagnostics Hub

- 1. Open Visual Studio 2013.
- 2. To open the **Performance and Diagnostics hub**, you can either select it from the **Debug** menu or by pressing the **Alt+F2** shortcut.



3. By default, the hub will only show you the tools that are available for the target based on language, application type, or platform. Since you don't have an analysis target chosen, no tools are currently shown.

Performance and Diagnostics Recently Opened Sessions No recent items were found.	Analysis Target  Choose Target ▼	
	Available Tools	Show all tools
	Select a target to see available tools.	
	Start	
Select the Show All Tool	<b>Is</b> link to view all of the tools included in the hub.	
Available Tools		
		Show all tools
Select a target to see available	tools.	

- 5. Note that all tools are now shown, albeit currently disabled.
   Not Applicable Tools 
   CPU Sampling
  - Examine which native and managed functions are using the CPU most frequently
  - HTML UI Responsiveness Examine where time is spent in your website or application
  - □ JavaScript Memory Investigate the JavaScript heap to help find issues such as memory leaks
  - XAML UI Responsiveness

4.

- Examine where time is spent in your application
- Energy Consumption Examine where energy is consumed in your application
- JavaScript Function Timing
   Examine where time is spent in your JavaScript code
- Performance Wizard CPU Sampling, Instrumentation, .NET Memory allocation, and Resource Contention

Note: Some of the tools shown here are not new in Visual Studio 2012 and 2013, but have simply been moved into the hub. These include the **Performance Wizard** and **JavaScript Function Timing** tools. New tools since Visual Studio 2012 include **CPU Sampling**, **JavaScript Memory**, and **HTML UI Responsiveness** (some were introduced in product updates). New tools since Visual Studio 2013 include **XAML UI Responsiveness** and **Energy Consumption**.

 Select the "Show target specific tools" link to return to the default hub view. Analysis Target

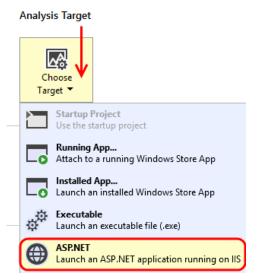


Available Tools

Select a target to see available tools.

Show target specific tools

7. Select the Choose Target dropdown and select the ASP.NET option.



8. Note that the hub now shows the Performance Wizard as being the only available tool, and therefore automatically selects it. If you were to instead select the Executable analysis target, you would see the same thing.

## Analysis Target



Launch an ASP.NET application running on IIS

## Available Tools

 Performance Wizard
 CPU Sampling, Instrumentation, .NET Memory allocation, and Resource Contention

ASP.NET

9. Select the **Change Target** dropdown (the name changed after selecting the first target) and then select the **Installed App...** option.

Analysis Target

Change Target ▼		ASP.NET Launch an ASP.NET application runn	ing on IIS		
	Startup Project Use the startup project Running App				
	Installed	Attach to a running Windows Store App Installed App Launch an installed Windows Store App			
 a <sup>©</sup>	<b>Executal</b> Launch a				
	ASP.NET Launch a	n ASP.NET application running on IIS			

10. In the Select Installed App Package window, select the Simulator option from the dropdown, and then search for (type 'calc' into the search box) and select the Windows Calculator app package. Click the Select button to continue.

Select Installed App Package	?	×
Simulator		~
App packages installed for the current session		
calc		×
Microsoft.WindowsCalculator_6.3.9600.20278_x64_8wekyb3d8bbwe Package Full Name: Microsoft.WindowsCalculator_6.3.9600.20278_x64_8wekyb3d8bbwe Version: 6.3.9600.20278		Ŷ
		~
Refresh Select	Cance	21

- The available tools now include only those that can be used to analyze a XAML-based Windows Store application, which include CPU <u>SamplingUsage</u>, XAML UI Responsiveness, and Energy Consumption.
- 12. Select the CPU Usage tool and then click Start.

#### Analysis Target



Installed App Microsoft.WindowsCalculator\_6.3.9600.20278\_x64\_8wekyb3d8bbwe Simulator



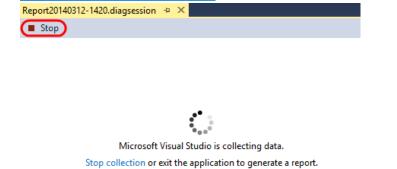
- Energy Consumption Examine where energy is consumed in your application
- XAML UI Responsiveness
   Examine where time is spent in your application

Start

13. In the Simulator window, go ahead and perform some fancy mathematics.

Standard Scientific Converter									
			s	qrt(8712)					
93.33809511662427322091145579784									
М	CE	с	Ø	÷					
±	7	8	9	×					
$\checkmark$	4	5	6						
%	1	2	3	+					
1/ <i>x</i>	(	)		=					





**15.** After the report is generated, you should see the report opened in **Summary** view. This view is a starting point in your investigation into performance issues. From each line in the Summary view, you can move to more detailed views by right-clicking the function or module name.

Note: Your report may be different from the one shown in the screenshot.

🔍 Zoom In 🔍 Reset Zoor	m XIII Clear Selection							
Diagnostic session: 48.443	seconds (48.443 seco	onds selected)			Vser mar	k 🔶 App	o lifecycle mai	rk
	10s	20s		30s		40s		
<ul> <li>CPU utilization (% of a</li> </ul>	all processors)					Pro	cess CPU Usa	ige
100							10	00
0					maple.	anna	0	
	Create d	letailed report	<b>▼</b> Filt	er view 🔻	Search		ç	o 🗸
Function Name					Total CPU	(%) 🔻	S	Self
numbers.exe (PID: 4192)					100	0.00 %		
4 [External Code]					100	0.00 %		1
numbers.exe					12	2.51 %		
numbers.exe					11	1.76 %		

- 16. Although analyzing applications using the CPU Usage tool is not the focus of this lab, you can learn more about using the tool with Windows 8 and Windows Server 2012 applications (and above) in <u>this</u> MSDN article. The purpose of this task was to introduce the Performance and Diagnostics hub as a central starting point to access most performance and diagnostic tools from within Visual Studio 2013.
- 17. Close the profiling report but keep Visual Studio open.

## Exercise 2: UI Responsiveness and Energy Consumption Tools

In this exercise, you will take a look at the diagnostic data that is collected when using the UI Responsiveness tools for Windows Store apps and then use that data to help create a fix that will improve a sample application. The key to maintaining a responsive app is keeping the UI thread as free as possible. After that, you will see what diagnostic data the Energy Consumption tool provides.

Note: Some of the new tools being shown in this lab are also available using the <u>F12 developer tools</u> in Internet Explorer 11.

### Task 1: Create a JavaScript Windows Store Application for Testing

In this task, you will create a JavaScript Windows Store app that will be used for testing in the next task.

- 1. Select File | New | Project.
- 2. Under **Templates | JavaScript**, select the **Blank App (Windows)** template, enter **JS\_Perf\_Tester** in the **Name** field and then click the **OK** button to create the solution.

				New Pro	ject			? 🗙
▷ Recent		.NET Fr	amework 4.5	- Sort by: Defa	ault	•		E Search Installed Templates (Ctrl+E) 👂 -
▲ Installed		<b>∑</b> J	Blank App (U	Universal Apps)	Jav	aScript	4	Type: JavaScript
✓ Templates ▷ Visual Basic ▷ Visual C#	Î			Windows Phone)		aScript		A project for a single-page Windows app that has no predefined controls or layout.
▷ Visual C++▷ Visual F#		<b>∑</b> j	Blank App (V	Nindows)	Jav	aScript	)	1
SQL Server		JS	Hub/Pivot A	App (Universal Apps)	Jav	aScript		
Python Online	-	2L		App (Universal Apps) here to go online and		aScript	*	
Name:	JS_Perf_Tester							
Location:	c:\users\user\doo	cuments	visual studio 2	2013\Projects				Browse
Solution:	Create new solut	ion				-		
Solution name:	JS_Perf_Tester							Create directory for solution
								Add to source control
								OK Cancel

3. If you do not have a developer license for Windows 8.1 installed yet, you will be prompted to agree to the terms and then install a license. Click the "I Agree" button if you agree with the terms. You will also need to click **Yes** in the User Account Control dialog box that appears.

Developer License	×
Get a developer license for Windows 8.1	
You need a developer license to develop this style of app for Windows 8.1. A developer license lets you install and test the app on this computer before Microsoft tests and certifies it.	
You may use the developer license only for the purpose of developing, testing and evaluating apps. In all other respects, the Windows 8.1 Software License Terms govern your use of Windows 8.1 and the developer license.	
When you get a developer license, some data is sent to Microsoft about your use of the developer license. Read the <u>privacy statement</u> for more information.	
If you agree to these terms and want to install a developer license, click 'I Agree'.	
Cance	I

4. If you are installing a developer license, you may also need to sign in with your Microsoft account. Go ahead and sign in to finish the developer license installation.

Developer License	×
Sign in	
Microsoft account What's this?	
Password	
Sign in	
Can't access your account?	
Don't have a Microsoft account? Sign up now	
Privacy & Cookies Terms ©2014 Microsoft	

5. After installing a developer license using a Microsoft account, you will be given 30 days before it expires. Click the **Close** button.



6. Open **default.html** and replace the existing contents of the **body** with the following markup.

HTML
<div class="wrapper">

```
<br/>
```

7. Open default.css and add the following CSS to the end.

```
HTML
#content {
    margin-left: 200px;
    margin-top: 200px;
}
```

8. Open **default.js** and replace the entire existing contents with the following code.

```
JavaScript
(function () {
    "use strict";
    var app = WinJS.Application;
    var activation = Windows.ApplicationModel.Activation;
    var content;
    var wrapper;
    app.onactivated = function (args) {
         if (args.detail.kind === activation.ActivationKind.launch) {
             if (args.detail.previousExecutionState !==
activation.ApplicationExecutionState.terminated) {
                 content = document.getElementById("content");
                 wrapper = document.querySelector(".wrapper");
                                                                                           Formatted: Portuguese (Brazil)
                 content.addEventListener("click", handler);
             } else {
}
             args.setPromise(WinJS.UI.processAll());
         }
    };
    app.oncheckpoint = function (args) {
    };
    app.start();
    var idx = 0;
    var count = 0;
    var max = 5000;
    var text = ["eenie", "meenie", "minie", "moe"];
var color = ["red", "crimson", "maroon", "purple"];
```

```
function increment() {
        setTimeout(function () {
            idx++;
            count++;
            if (idx > 3) { idx = 0; }
            if (count < max) { increment(); }</pre>
        }, 1000);
    }
    function setValues() {
        content = document.getElementById("content");
        content.removeNode(true);
        var newNode = document.createElement("button");
        newNode.id = "content";
        newNode.textContent = text[idx];
        newNode.style.backgroundColor = color[idx];
        wrapper.appendChild(newNode);
    }
    function update() {
        setTimeout(function () {
            setValues();
            if (count < max) { update(); }</pre>
        });
    }
    function handler(args) {
        performance.mark("Click");
        content.textContent = "eenie";
        increment();
        update();
    }
})();
```

- 9. Press F5 to compile the application and start debugging.
- 10. Click the "Waiting for values" button and verify that the button text and color are updated approximately once per second. This is by design.

11. Close the app by pressing Alt+F4. To return to the Desktop, either press Win+D or select the Desktop tile on the Start screen.

#### Task 2: Using the HTML UI Responsiveness Tool

In this task, you will use the HTML UI Responsiveness Tool to profile the Windows Store application you created in the previous task, and then use that data to help locate a performance problem and fix it.

- Before we start using the HTML UI Responsiveness tool, let's configure the application to use the Windows Simulator. In Solution Explorer, select the project node and press Alt+Enter to open the project properties window.
- Select the Debugging page, select the Simulator debugger option and then click OK. One advantage of using the simulator here is that you can place it next to Visual Studio and easily switch between the running app and the profiler.

	JS_Perf_Tester Proper	rty Pages ? 💌			
Configuration: Active(Debug)	✓ Platform: Active(A	Any CPU) V Configuration Manager			
<ul> <li>Configuration Properties</li> <li>General</li> </ul>	Debugger to launch:	Ŷ			
Debugging		•			
	Launch Application	Yes			
	Allow Local Network Loopback	Yes			
	Debugger Type	Script Only			
	Debugger Type				
Specifies which debugger to enable. When set to Mixed, the debuggers for both managed and native code are invoked.					
		OK Cancel Apply			

3. To open the **Performance and Diagnostics hub**, you can either select it from the **Debug** menu or by pressing the **Alt+F2** shortcut.

DEE	BUG	TEAM	TOOLS	TEST	ARCHITECTURE	ANALYZE					
	Wir	ndows				•					
	Graphics •										
►	Sta	rt Debuggi	ng		F5						
►	Sta	rt Without	Debugging	)	Ctrl+F5						
e <sup>®</sup>	Atta	ach to Pro	cess								
	Debug Installed App Package										
	Exc	eptions			Ctr	l+Alt+E					
(	Per	formance	and Diagno	ostics	Alt	+F2					

4. Select the **HTML UI Responsiveness** tool from the list of available tools and then click **Start**. In this case, you are using the default analysis target which is the startup project.

#### Analysis Target



Startup Project JS\_Perf\_Tester

#### Available Tools



HTML UI Responsiveness

## Examine where time is spent in your website or application

#### □ JavaScript Memory Investigate the JavaScript heap to help find issues such as memory leaks



Note: When you start the profiler, you may see a User Account Control prompt requesting permission to run VsEtwCollector.exe. Click **Yes**.

5. In the simulator that is now running your app, click the "Waiting for values" button and then let it run for about **10** seconds. You should see the button text and color update about once per second.

Waiting for values		

6. Switch back to Visual Studio and click the Stop button to stop the profiler.

- Energy Consumption Examine where energy is consumed in your application
- ☐ JavaScript Function Timing Examine where time is spent in your JavaScript code

 After the diagnostic session report is shown, take a look at the CPU utilization graph and note that it increases dramatically after a few seconds (at the point in time where you clicked the button).

🔍 Zooi	n in 🔍 Reset zoom	$^{\times}_{lili}$ Clear selection	- I					
Diagn	ostic session: 14.907 se	conds	<b>↓</b>				♦ App lifecycle mark 🔻 U	Jser mark
	• • • • • • • •	2.5s		5s	7.5s	10s	12.5s	
	CPU utilization (%)				Loading Scrip	oting 🧧 GC 📕 Styling	Rendering Image decoding	Other
100								100
				_				

8. The timeline bar at the top also shows various app lifecycle and user marks. If you hold the mouse cursor over the user mark (orange triangle) you should see the text "Click" with a time appended. This was recorded by the application using JavaScript in the button click handler. Disposite session: 14.907 seconds.

• • • • • • •	2.5s	5s
<ul> <li>CPU utilization (%)</li> </ul>		Click at 3.78 s
100		
1 A A A A A A A A A A A A A A A A A A A		

 Hold the mouse cursor over the app lifecycle mark near the beginning of the timeline, and take note of the different events that were recorded around this time period as the application was first loading - representing Navigation, DOMContentLoaded, and Load event types. Diagnostic session: 14.907 seconds

- (	• • • • • • • •	2.5s	5s	7.5s
▲ CP 100	DOMContentLoaded (ms-app	50f-1a91-43c5-830a-472fd550f9 px://5c59b50f-1a91-43c5-830a-4 a91-43c5-830a-472fd550f936/de	172fd550f936/default.html) at 2	27.16 ms

10. By default, you will be viewing data for the entire length of the diagnostic session. Select a couple seconds from the middle portion of the CPU utilization graph using a **click-and-drag** selection technique, after the point where the button was clicked.

Note: the non-shaded area represents the selection.

Diagn	nostic session: 14.907 seconds	2.5s	5s	7.5s	10s	
4	CPU utilization (%)				Loading Scripting	GC Styling
100						
			$\rightarrow$	←		
	1					

11. Also take note of the **Visual throughput (FPS)** graph. In this case, the FPS remains at 60 throughout the diagnostic session, therefore there are no dropped frames. Periods of excessive CPU utilization can result in low or inconsistent frame rates. If you develop rich media apps and games, the visual throughput graph may provide more important data than the CPU utilization graph.

4	Visual throughput (FPS)			Frames per sec	cond
					60
60					00
30	·				30

12. Click the **Zoom In** button from the top-left of the diagnostic window.

🖲 Zoom in	२ Reset zoom	🏭 Clear selection			
Diagnostic se	ssion: 14.907 se	conds			
•		2.5s	. 🔻	5s	7.5s

13. Zooming into the selection shows the selected period in more detail.

🔍 Zoom in 🔍 Reset zoom	× IIII Clear selection						
Diagnostic session: 14.907 sec	onds					<ul> <li>App</li> </ul>	lifecycle mark 🔻 User mark
5.5s	5.75s	6s	6.25s	6.5s	6.75s	7s	7.25s
CPU utilization (%)							Image decoding Other

14. The **Timeline Details** section shows the filtered events for the selected period. By default, the timeline is sorted sequentially, meaning that events that occurred earlier in time will appear higher (and to the left) than events which occurred later. These events confirm the visible trends that you saw in the CPU utilization graph, in that there are many events taking place over short periods of time.

Timeline details: 2.01 seconds

Event name	5.5s		
Timer (Anonymous fun	0.25 ms (0.15 ms)		
Timer (Anonymous fun	0.57 ms (0.33 ms)		
Timer (Anonymous fun	0.27 ms (0.16 ms)		
▷ Layout	0.3 ms (0.12 ms)		
Timer (Anonymous fun	0.17 ms (0.093 ms)		
Paint [24132]	0.48 ms (0.23 ms)		
Timer (Anonymous fun	0.15 ms (0.081 ms)		
Timer (Anonymous fun	0.15 ms (0.078 ms)		
Timer (Anonymous fun	0.5 ms (0.3 ms)		

15. As is the case with the CPU utilization graph, the colors next to each event correspond to categories such as Loading, Scripting, Styling, Rendering, and so on. The first number next to each event corresponds to the **inclusive duration** time, and the second number to the **exclusive duration** time.

#### Timeline details: 2.01 seconds

Eve	ent name	5.5s		
⊳	Timer (Anonymous fun			
⊳	Timer (Anonymous fun	0.57 ms (0.33 ms)		
⊳	Timer (Anonymous fun			
⊳	Layout	0.3 ms (0.12 ms)		
⊳	Timer (Anonymous fun	0.17 ms (0.093 ms)		
Þ	Paint [24132]	0.48 ms (0.23 ms)		

## 16. Select and expand the first Timer event.

Timeline of	details:	2.01 se	conds
-------------	----------	---------	-------

Eve	ent name	5.5s		
$\bigcirc$	Timer (Anonymous fun	0.25 ms (0.15 ms)		
⊳	Timer (Anonymous fun	0.57 ms (0.33 ms)		
⊳	Timer (Anonymous fun	0.27 ms (0.16 ms)		
⊳	Layout	0.3 ms (0.12 ms)		

17. Selecting an event will show a details pane to the right. This shows us that this Timer event executed an anonymous function in default.js and kicked off a number of other processes as a result (represented by the child events). Note that the **Delay** property for the timer is **0**, meaning no delay.

Event name	6s	7s	Tir	ner		
<ul> <li>∠ Timer (Anonymous fun documentcreateEL</li> <li>&gt; Style calculation set style</li> <li>&gt; appendChild()</li> <li>&gt; Timer (Anonymous fun</li> <li>&gt; Layout</li> <li>&gt; Timer (Anonymous fun</li> <li>&gt; Pimit [24132]</li> <li>&gt; Timer (Anonymous fun</li> </ul>	0.25 ms (0.15 ms)           0.0063 ms           0.039 ms (0.0003 ms)           0.0088 ms           0.0708 ms (0.0036 ms)           0.0708 ms (0.0036 ms)           0.071 ms (0.33 ms)           0.071 ms (0.34 ms)           0.071 ms (0.072 ms)           0.17 ms (0.093 ms)           0.17 ms (0.094 ms)           0.15 ms (0.078 ms)           0.16 ms (0.097 ms)           0.16 ms (0.097 ms)           0.27 ms (0.16 ms)           0.28 ms (0.14 ms)           0.28 ms (0.16 ms)           0.28 ms (0.16 ms)           0.28 ms (0.16 ms)           0.28 ms (0.16 ms)           0.28 ms (0.23 ms)		DU DU Sti Th So Na Tir De A : ass	uration (inclusive): aration (exclusive): art time: uread: uurce location: ame: mer type: elay:	0.25 ms 0.15 ms 5.49 s UI thread default.js (69,20) Anonymous function Timeout timer 0 s d which resulted in the execution of its	

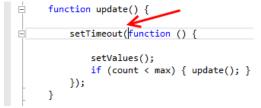
18. Take a look through the various child events recorded for the same timer event. This shows that there is a call to document.createElement(), followed by a style calculation, and finally a call to appendChild().

Time	Timeline details: 2.01 seconds					
Event	t name	бѕ				
⊿ T	imer (Anonymous fun	0.25 ms (0.15 ms)				
	document.createEl	0.0063 ms				
Þ	Style calculation	0.039 ms (0.0003 ms)				
	set style	0.0088 ms				
Þ	appendChild()	0.048 ms (0.0036 ms)				

- 19. In the short time span selected (approximately one to two seconds), there are a great number of Timer, Layout, and Paint events taking place. **Timer** events occur most frequently, far more frequently than the one update per second that is visibly apparent after you run the app and click the button.
- 20. Click the **Source Location** link to navigate to **default.js** and the location of the anonymous function that was called.

Timer		
Duration (inclusive):	0.5 ms	
Duration (exclusive):	0.22 ms	
Start time:	4.72 s	
Thread:	UI thread	
Callback function:	Anonymous function	
Timer type:	Timeout	
Delay:	0 s	
A scheduled timer elap execution of its associ	psed which resulted in the ated callback.	

21. You should now be looking at the anonymous function definition.



- 22. As you can see, this anonymous function calls setValues(), which updates the button in the UI. Unfortunately, it appears that this function is running too frequently (as you just saw quite a few associated Timer events), and is therefore updating the UI when it isn't necessary. This is due to the use of a default timeout value of 0 in the call to setTimeout().
- 23. To fix the problem, add a second parameter to the setTimeout() call that specifies a 1000 millisecond delay. You can do this by replacing the update() function with the following. This will fix the issue with the excessive Timer events.

```
JavaScript
function update() {
    setTimeout(function () {
        setValues();
        if (count < max) { update(); }
    }, 1000);
}</pre>
```

24. Run the HTML UI Responsiveness tool again, click the button, and then check the CPU utilization graph to verify that this reduces utilization as expected. Also note that the excessive Timer events are now gone.

Diagnostic session: 14.652 se	2.5s	5s	7.5s	10		fecycle mark Vser	mark
<ul> <li>CPU utilization (%)</li> </ul>		Loading	Scripting	GC Sty	ling Rendering	Image decoding	Other
100							100
Visual throughput (FP)	5)					Frames per s	second
60 30							60 30
Timeline details: 14.652 seco	nds				Sort	by: Start time	~
Event name	• • • • • • • • • • •	5s   1	0s				
Speculative downloadi HTML parsing Speculative downloadi CSS parsing (ui-darkcss) HTML parsing Style calculation DOM event (DOMCont Windows Runtime eve	3.71 ms 0.0003 ms 8.42 ms 13.86 ms (0.62 ms) 0.047 ms (0.0006 ms) 0.12 ms (0.0097 ms)			^	Select an event within details	the timeline to view	its

25. If you are interested in learning more about analyzing UI responsiveness in JavaScript applications, you can take a look at <u>this</u> MSDN article.

#### Task 3: Using the XAML UI Responsiveness Tool

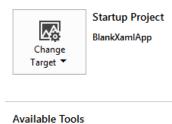
In this task, you will use the XAML UI Responsiveness Tool to profile a XAML Windows Store application, and then use that data to help locate a performance problem and fix it.

- 1. Open the BlankXamlApp.sln solution file found in the lab's Source\BlankXamlApp folder.
- 2. Configure the project to use the **Windows Simulator** when debugging as you did at the beginning of the previous exercise (although this time look for the Target Device setting on the Debug tab).

Application	Configuration: Active (Debug) V Platform: Active (Any CPU) V
Build	Configuration. Active (bebudg)
Build Events	Start Action
Debug*	Do not launch, but debug my code when it starts
Reference Paths	✓ Allow Local Network Loopback
Signing	Allow Edeal Network Edepback
Code Analysis	Start Options
	Target device: Simulator 🗸 🗸
	Remote machine: Find
	✓ Use authentication
	Uninstall and then re-install my package. All information about the application state is deleted

3. Open the **Performance and Diagnostics hub**, select the **XAML UI Responsiveness** tool, and then click **Start**.

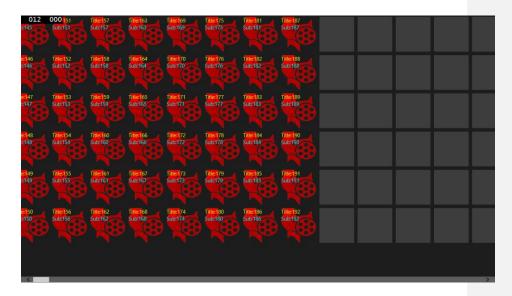
Analysis Target



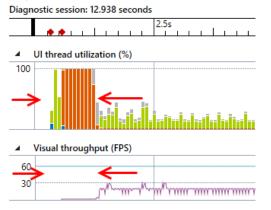




4. The application simply shows a grid view with a bunch of data items. Each data item shows a few lines of data-bound text and contains a background image. As you scroll to the right, you should notice that the animation is not very smooth and that it can take quite a while before new tiles are shown. The app does not feel very responsive.

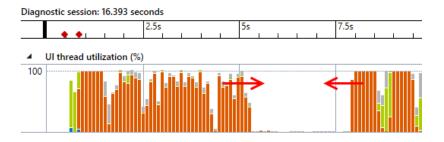


- 5. Switch back to Visual Studio and click the Stop button to stop the profiler.
- 6. After the diagnostic session report is shown, take a look at the UI thread utilization (%) graph. This shows that it took quite a while and required quite a bit of layout work before the initial tiles were visible. The Visual throughput graph also bears this out as the FPS was close to zero during this time period.



Note: You can also add user marks to XAML-based Windows Store applications, although we will not do so in this lab.

7. Looking further down the UI thread utilization graph, there may be a period where no layout was occurring before you started scrolling through more items.



8. Looking further down the UI thread utilization graph, note that layout code starts to dominate the UI thread once again, and the visual throughput drops significantly.



9. The XAML UI Responsiveness tool shows two detailed views below the graphs. The first view is named Hot Elements, and this contains a horizontal bar graph that represents all of the elements that participated in layout during the selected portion of the diagnostics session. They are grouped by their template and sorted in descending order by the time they took for layout. Note that the GridView control and its children were responsible for a large portion of the layout time.

ype	45		Windows.UI.Xaml.Cor	ntrols.GridView
Windows.UI.Xaml.Controls.GridView (1)	4.52 s			
<ul> <li>Windows.UI.Xaml.Controls.GridViewItem (115)</li> </ul>	88.86 ms	$\sim$	Element Name:	
Windows.UI.Xaml.Controls.Frame (1)	0.43 ms		Template Name:	default
Windows.UI.Xaml.Controls.Primitives.ListViewBasel	0.19 ms			
Windows.UI.Xaml.Controls.ScrollContentPresenter.	0.076 ms		Element Count:	49
<ul> <li>Windows.UI.Xaml.Controls.ScrollViewer (1)</li> </ul>	0.041 ms			
> Windows.UI.Xaml.!RootVisual (1)	0.025 ms			
BlankXamlApp.MainPage (1)	0.014 ms			
Windows.UI.Xaml.Controls.Grid (1)	0.01 ms			
> Windows.UI.Xaml.Controls.Border (1)	0.006 ms			
> Windows.UI.Xaml.!PopupRoot (1)	0.0036 ms			
Windows.UI.Xaml.!PrintRoot (1)	0.003 ms			

#### 10. Select the Parsing link tab.



11. The parsing view doesn't indicate that parsing is a significant contributor to overall UI thread utilization as it only took a few milliseconds.

File name		10s		ms-resource:/Files/app.xa	ml
ms-resource/Files/app.xaml ms-resource/Files/windows.ui.xaml;component ms-appx/MainPage.xaml	0.24 ms 7.65 ms 2 ms		^	Start time: End time: UI Elements (inclusive):	371.94 ms 372.18 ms 0
				UI Elements (exclusive): Duration (inclusive): Duration (exclusive):	0 0.24 ms 0.24 ms

- 12. Switch back to the Hot Elements tab.
- 13. Leave the diagnostic session window open, as you can compare this baseline to future runs.
- 14. Open MainPage.xaml and take a look at the XAML.

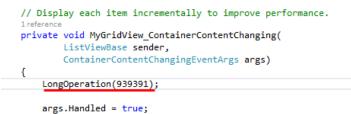
```
<GridView x:Name="myGridView"
         ItemsSource="{Binding}"
         SelectionChanged="myGridView_SelectionChanged"
         Background="{StaticResource ApplicationPageBackgroundThemeBrush}"
         ContainerContentChanging="MyGridView_ContainerContentChanging"
   <GridView.ItemTemplate>
       <DataTemplate>
           <StackPanel Height="100"
                       Width="100" Background="{StaticResource ImageBrush1}"
                <Rectangle x:Name="placeholderRectangle"
                          Fill="Red"
                          Opacity="0"/>
                <TextBlock x:Name="titleTextBlock"
                          Text="{Binding Title}"
                          Foreground="Yellow"/>
                <TextBlock x:Name="subtitleTextBlock"
                          Text="{Binding Subtitle}"
                          Foreground="Aqua"/>
                <TextBlock x:Name="descriptionTextBlock"
                          Text="{Binding Description}"
                          Foreground="Gray"/>
           </StackPanel>
       </DataTemplate>
   </GridView.ItemTemplate>
```

15. The GridView is ultimately bound to a list of data items in code, and it uses a data template definition to display each item. In an attempt to improve UI responsiveness, the developer of the application tried to take advantage of the new <u>ContainerContentChanging</u> event, but unfortunately it appears that the code added to the handler is quite expensive.

 Place the cursor anywhere on the event handler assignment for the ContainerContentChanging event (MyGridView\_ContainerContentChanging) and then press F12 to go to the definition in the code behind.

<gridview< td=""><td>x:Name="myGridView"</td></gridview<>	x:Name="myGridView"
	<pre>ItemsSource="{Binding}"</pre>
	SelectionChanged="myGridView_SelectionChanged"
	<pre>Background="{StaticResource ApplicationPageBackgroundThemeBrush}"</pre>
	ContainerContentChanging="MyGridView_ContainerContentChanging"
	>
<grid< td=""><td>/iew.ItemTemplate&gt;</td></grid<>	/iew.ItemTemplate>

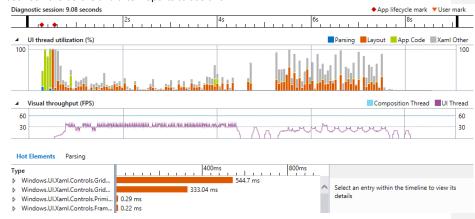
17. The first line of the event handler makes a call to a long running operation, and is likely to be a major contributor to the poor responsiveness that you just experienced.



- 18. For the purposes of this lab, assume the following about the long running operation:
  - It needs to be performed here and can't be optimized significantly
  - It can be performed asynchronously (code later in the handler is not dependent on the result)
- 19. Wrap the line of code representing the long running operation by using **Task.Run()** to offload the operation from the UI thread.

```
// Display each item incrementally to improve performance.
Ireference
private void MyGridView_ContainerContentChanging(
    ListViewBase sender,
    ContainerContentChangingEventArgs args)
{
    Task.Run(() =>
    {
       LongOperation(939391);
    });
    args.Handled = true;
    }
}
```

- 20. Run the **XAML UI Responsiveness** tool once again, performing a similar test that you previously did including scrolling through the items. You should notice a marked improvement in responsiveness as you scroll. Note that the third line of text, which is meant to be a description, still takes a significant amount of time to render.
- 21. **Stop** the profiler to view the report.

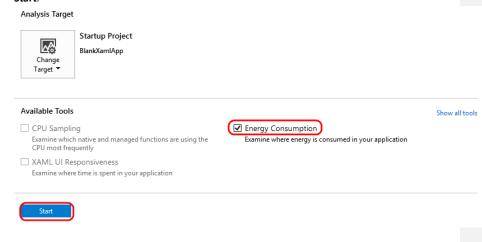


## 22. Note that the UI thread utilization percentage is much less than it was before. You can switch between the before and after reports to see this.

## Task 4: Using the Energy Consumption Tool

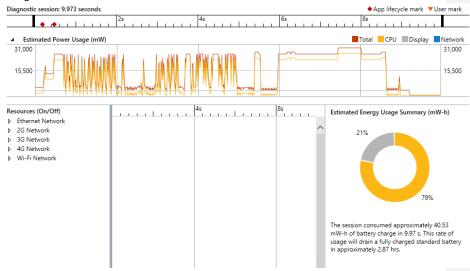
In this task, you will use the Energy Consumption Tool to profile a XAML Windows Store application, and then use that data to help give you an idea of the energy requirements. This profiler helps you analyze the power and energy consumption of Windows Store apps on low-power tablet devices that run all or part of the time on their own batteries. On a battery-powered device, an app that uses too much energy can cause so much customer dissatisfaction that, eventually, customers might even uninstall it. Optimizing energy use can increase your app's adoption and use by customers.

1. Open the **Performance and Diagnostics hub**, select the **Energy Consumption** tool, and then click **Start**.

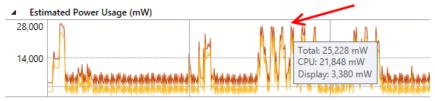


Note: Energy profiling using the Windows Store simulator or on the same computer as Visual Studio is running on is not generally recommended, as profiling on the actual device provides far more realistic data. The energy profiler **estimates** power and energy use by using a software model of standard reference device hardware that is representative of the low powered tablet devices your application might run on.

- 2. Scroll through the application for a few seconds and then Switch back to **Visual Studio** and click the **Stop** button.
- The diagnostic session report for the Energy Consumption tool shows the activity level of the display, CPU, and network connections and shows estimates of the power and total energy used during the session.

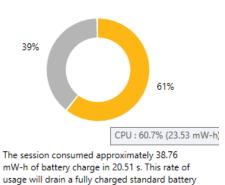


4. If you hold the mouse cursor over areas of **peak power** usage (associated with your scrolling), you'll note that the CPU energy usage spikes, but the estimated display power usage remains constant. Looking for spikes in power usage can help you identify potential areas for further optimization. Although the magnitude of the power usage numbers can be useful (if collected from a real device), it's the relative measure of any savings that you can produce through optimization that is the key -- each application has different hardware requirements.



Note: **Power** measures the rate that force is used to perform work that is done in a period of time. In electrical science, the standard unit of power is a watt, which is defined as the rate at which work is done when one ampere of current flows through an electrical potential difference of one volt. In the **Power Usage** graph, the units are displayed as milliwatts (**mW**) which are one thousandth of a watt.

5. If you hold the mouse cursor over the total energy use summary pie graph, you can see the estimate for each category. This shows that it is estimated that the CPU consumes most of the energy over the profiling period, with the remainder going to the display.



Estimated Energy Usage Summary (mW-h)

Note: To obtain the good estimates, you'll want to profile the energy use of the app on a lowpowered device that is being powered by its batteries. Because Visual Studio does not run on most of these devices, you'll need to connect your Visual Studio computer to the device using the Visual Studio <u>remote tools</u>. To connect to a remote device, you need to configure both the Visual Studio project and the remote device.

## Exercise 3: Memory Tools and Diagnostics

In this exercise, you will learn how to analyze JavaScript Windows Store applications for memory leaks and analyze managed memory dump files.

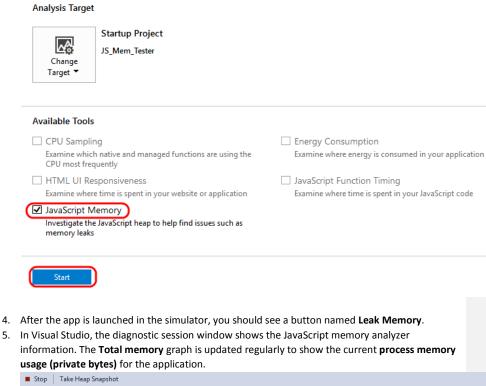
#### Task 1: Analyze JavaScript Memory Usage

in approximately 6.17 hrs.

In this task, you will use the JavaScript memory analyzer to help identify a simple memory issue in a Windows Store application.

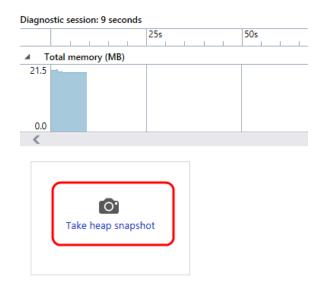
Note: **Energy** measures the total amount of power, either as a capacity or potential, as in the power capacity of a battery, or as the total amounted of power expended over a period of time. The unit of energy is a watt-hour, the amount of power of one watt constantly applied for one hour. In the **Energy Summary**, the units are displayed as milliwatt-hours (**mW-h**).

- 1. Open the JS\_Mem\_Tester.sln solution file found in the lab's Source\JS\_Mem\_Tester folder.
- 2. Configure the project to use the Windows Simulator.
- Open the Performance and Diagnostics hub, select the JavaScript Memory tool, and then click Start.



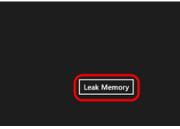
ignostic session: 9 second	s					▼User mar
_	25s	50s	1:15min	1:40min	2:05min	2:30min
Total memory (MB)					Process memory usage	(private byte
21.5						21
0.0						0.0

6. Click the Take heap snapshot button. This represents your baseline snapshot.



Note: In your own scenarios, take your first snapshot just before a suspected memory leak, if possible.

7. Switch to the app and click Leak Memory.

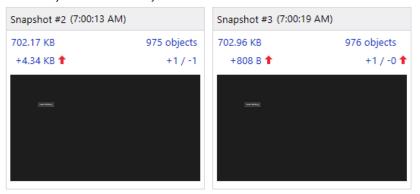


- 8. In this scenario, you suspect that you just performed an action that may result in a memory leak. Switch to Visual Studio and click **Take heap snapshot** again.
- 9. Switch to the app and click Leak Memory again.
- 10. Switch to Visual Studio and click Take heap snapshot for the third time. By taking a third snapshot in this workflow, you can filter out changes from the baseline snapshot to the second snapshot that aren't associated with memory leaks. For example, there may be expected changes such as updating headers and footers on a page, which will generate some changes in memory usage but may be unrelated to memory leaks.
- 11. In Visual Studio, click the **Stop** button to stop profiling.
- 12. Start analyzing the snapshots by comparing the first two. **Snapshot #2** shows that the heap size (shown by the red up arrow on the left) has increased by more than 4 KB compared to the

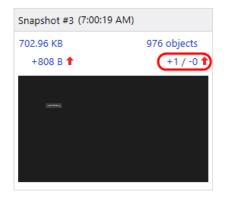
baseline snapshot. In addition, the number of objects on the heap (shown by the red up arrow on the right) has increased compared to the baseline, with one object added and one removed.

Snapshot #1 (7:00:01 /	AM)	Snapshot #2 (7:00:13	AM)
697.83 KB	975 objects	702.17 KB	975 objec
Baseline	Baseline	+4.34 KB 🕇	+1/
		(and (2004))	

13. **Snapshot #3** shows that the heap size has increased again compared to the previous snapshot, with one object added and no objects removed.



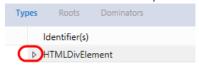
14. In **Snapshot #3**, select the "**+1 / -0**" link to view a differential view of the objects on the heap compared to Snapshot #2.



15. The differential view of heap objects shows the **Types** view by default, sorted by object count difference. This shows you the objects that were added between snapshot 2 and 3.

Types	Roots	Dominators	Scope: Objects add	ded between S	Snapshot #2	and #3 (1) 🔽	Identifier filter		•
	Identifier(s)		Туре	Size	Size diff.	Retained size	Retained size diff.	Count	
⊳	HTMLDivElen	nent		672 B		744 B	+744 B	1	

16. Expand the **HTMLDivElement** group at the top of the list to view additional information about the **div** elements for the two snapshots.



17. The div element that has been added to the heap between Snapshot #2 and Snapshot #3 represents a potential memory leak in the app.

Types	Roots	Dominators	Scope:	Objects ad	ded betwee	n Snapshot #2 a	and #3 (1) 🔽 Ident	ifier filter
Id	lentifier(s)		Туре	Size	Size diff.	Retained size	Retained size diff.	Count
	TMLDivEle	ment		672 B		744 B	+744 B	1
	<div id="&lt;/th"><th>'item"&gt;</th><th>HTMLDivElement</th><th>672 B</th><th></th><th>744 B</th><th>+744 B</th><th></th></div>	'item">	HTMLDivElement	672 B		744 B	+744 B	

18. Select the **Dominators** tab. This view shows a list of heap objects that have exclusive references to other objects.



Types Roots	Dominators	Sc	ope: Objec	ts added bet	ween Snapshot	#2 and #3 (1) 🗸	Identifier filt	er
Identifier(s)		Туре	Size	Size diff.	Retained size	Retained size di	ff.	
<div id="iter&lt;/td&gt;&lt;th&gt;n"><td>HTMLDivElement</td><td>672 B</td><td></td><td>744 B</td><td>+74</td><td>4 B</td><td></td></div>	HTMLDivElement	672 B		744 B	+74	4 B		
Object references								
Identifier(c)			-	me	Size	Size diff	*	Retained size diff

- Identifier(s)
   Type
   Size
   Size diff.
   Retained size diff.

   4 <div id="item">
   HTMLDivElement
   672 B
   744 B
   +744 B

   > <div class="wrapper">
   HTMLDivElement
   740 B
   +64 B
   4.15 KB
   +808 B
- 19. In this scenario, the **Dominators** view shows similar information to the Types view, but the information is sorted by retained size instead of object count. When you remove a dominator from memory, you reclaim all memory that the object retains. A diff view of the dominators can be helpful to quickly identify the objects that consume the most memory.
- 20. Some knowledge of the app helps at this point; choosing the Leak Memory button should remove a DIV element as well as add an element, so the code doesn't seem to be working right (that is, it leaks memory). The next task shows how to fix that.

#### Task 2: Fixing the JavaScript Memory Leak

In this task, you will fix the memory leak.

- 1. From the analysis of the JavaScript memory usage, you determined that div elements with an ID of "item" may be leaking. Open the **default.js** script file from the **js** folder for the project.
- Scroll down and locate the initialize function. This is called each time the run function is called, which is on first load of the app and each time the button is clicked. It appears that it is attempting to remove a cached div element with a call to removeNode, so that doesn't explain why a leak is occurring.

```
function initialize() {
    if (wrapper != null) {
        elem.removeNode(true);
    }
```

```
}
```

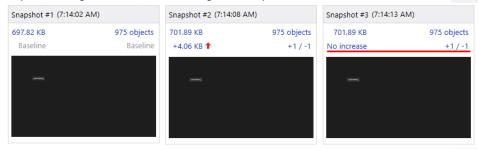
 Take a look at the load function. In part, it creates a new div element and appends it as a child of the wrapper div element. However, it doesn't update the cached div element (which the initialize method uses later in an attempt to remove the old div element).

```
function load() {
    wrapper = document.querySelector(".wrapper");
    var newDiv = document.createElement("div");
    var img = new Image();
    img.src = "/images/logo-scale-100.png";
    newDiv.style.backgroundImage = 'url(' + img.src + ')';
    newDiv.style.zIndex = "-1";
    newDiv.id = "item";
    //elem = newDiv;
    wrapper.appendChild(newDiv);
}
```

4. Fix the memory leak by uncommenting the line that assigns the new div element to the cached element reference named "elem".

```
newDiv.style.zIndex = "-1";
newDiv.id = "item";
elem = newDiv;
wrapper.appendChild(newDiv);
}
```

5. Using the same steps as before, use the JavaScript Memory tool to analyze the memory usage of the app by taking a baseline snapshot, clicking the Leak Memory button, taking a second snapshot, clicking the button, and then taking a third snapshot.



- Snapshot #3 now shows the heap size has no increase over Snapshot #2, and the object count is shown as +1 / -1, which indicates that one object was added and one removed. This is the expected behavior for the app, so the memory leak has been fixed.
- 7. You can close any Visual Studio instances you have open at this point.

#### Task 3: Debug Managed Memory

In this task, you will learn how to use the Debug Managed Memory feature found in Visual Studio Ultimate 2013 to help diagnose memory issues from production environments. These memory issues can fit in to a number of categories, including memory leaks, inefficient memory usage, and unnecessary allocations.

Note: this task requires Visual Studio Ultimate 2013.

- Open the SampleLeak.sln solution file found in the lab's Source\SampleLeak folder. This is just a MVC web application created from the Visual Studio 2013 template, with a memory leak introduced that occurs when loading the home page of the application.
- 2. Press **Ctrl+F5** to start the web application without attaching the debugger. The project should already be configured to use IIS Express when debugging.

Note: It will take a moment to restore NuGet packages.

3. After starting the web application, IIS Express will start up and host the application, and Visual Studio will launch a browser window and navigate to the home page of the site.

ASP.NE	Г	
ASP.NET is a free web fra using HTML, CSS and Jav	mework for building great Web site vaScript.	es and Web applications
Learn more »		
Getting started	Get more libraries	Web Hosting
Getting started ASP.NET MVC gives you a powerful, patterns- based way to build dynamic websites that enables a clean separation of concerns and gives you full control over markup for enjoyable, agile development.	Get more libraries NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.	Web Hosting You can easily find a web hosting company that offers the right mix of features and price for your applications.

- 4. In Visual Studio, press Ctrl+Alt+P to open the Attach to Process window.
- 5. In the Attach to Process window, click the Select button to the right of the Attach To setting.

ransport:	Default		
)ualifier:	RDAVIS-8		Find

6. In the Select Code Type window, select the option to "Automatically determine the type of code to debug" and then click OK.

Select Code Type ?								
Automatically determine the type of code to debug								
O Debug these code types:								
GPU - Software Emulator	^							
Managed (v3.5, v3.0, v2.0)								
Managed (v4.5, v4.0)								
Native Native								
Script								
Silverlight	~							
OK Cancel								

7. Select iisexpress.exe from the list of available processes and then click Attach.

Transport: Default						
ualifier:	RDAVIS-	8		¥	<u>F</u> ind	
ansport Information The default transport lets Monitor (MSVSMON.EXE)		t processes on this computer or a remote co	mputer running the Micros	oft Visual Studio Remote	Debugging	
tach to:	Automat	ic: Managed (v4.5, v4.0) code			<u>S</u> elect.	
vailable Processes	ID	Title	Туре	User Name	Session	
iisexpress.exe	21020		~ .	RDAVIS-8\Richard [a	1	
iisexpresstray.exe	14116		Managed (v4	RDAVIS-8\Richard [a	1	-
ipoint.exe	4964		×64	RDAVIS-8\Richard	1	
itype.exe	4812		×64	RDAVIS-8\Richard	1	
jucheck.exe	2284		×86	RDAVIS-8\Richard [a	1	
jusched.exe	13724		×86	RDAVIS-8\Richard	б	1
jusched.exe	7980		×86	RDAVIS-8\Richard	1	
livecomm.exe	17128		×64	RDAVIS-8\Richard [I	1	
lync.exe	6616	Lync	×86	RDAVIS-8\Richard	1	
Microsoft.Alm.Shared	13348		Managed (v4	RDAVIS-8\Richard [a	1	
ки: e:	10004	Kati sa	N.A. L	DDAVIC OD:-L	-1	-
Show processes from a	all users				Refresh	

- 8. Select Debug | Break All from the main menu or by pressing Ctrl+Alt+Break.
- 9. Select **Debug | Save Dump As** from the main menu.

Note: Although there are caveats to keep in mind, you could create the dump file through other means such as using Task Manager or by using free tools such as ProcDump. Please see <u>this</u> blog post from the Visual Studio Debugger Team Blog for more information if interested.

10. In the **Save Dump As** window, name the file **iisexpress\_baseline.dmp**, choose to save the dump file with heap information (this is the default "save as type" option), choose a location to save to, and then click **Save**.

M	Save Dump As					
€ ∋ - ↑ 🎚	« bette bij a temp a statto	framingfilt = Reffling +	✓ C Search PerfDiag	م		
File name:	iisexpress_baseline.dmp			~		
Save as type:	Minidump with Heap (*.dmp)			~		
💌 Browse Folders			Save	Cancel		

- 11. In Visual Studio, press F5 to let the IIS worker process continue running.
- 12. Return to the browser window and then press F5 to refresh the page five times.

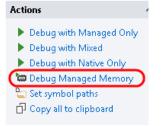
- 13. In Visual Studio, select Debug | Break All.
- 14. Select **Debug | Save Dump As** to save a dump file to the same location as the first dump, but this time with the name **isexpress\_leak.dmp**.
- 15. Press Shift+F5 to stop debugging.
- 16. In an Explorer window, navigate to the location where the dump files were saved.
- 17. Double-click on the **iisexpress\_leak.dmp** file to open it with Visual Studio Ultimate 2013.

Note: The process the dump file was collected against must have been running on .NET 4.5 or higher.

18. Once the file is open in Visual Studio Ultimate, you will be presented with the dump file summary page. This shows when the dump was created, the architecture of the process, the version of Windows, and what version of the runtime (CLR version) the process was running. Minidump File Summary

3/17/2014 10:45:57 PM			ł
Dump Summary      Dump File Last Write Time Process Name Process Architecture Exception Code Exception Information Heap Information      A System Information	iisexpress_leak.dmp :	Actions  Debug with Managed Only Debug with Mixed Debug with Mixed Debug with Mixe Only Debug Managed Memory Copy all to clipboard	•
OS Version CLR Version(s)	6.3.9600 4.0.30319.34011	-	

19. Click the **Debug Managed Memory** action link to the right of the dump summary.



20. Once the analysis is complete, you should see the new managed memory analysis view. The top pane contains a list of objects in the heap, grouped by their type name with columns that show you the count and total size. When a type or instance is selected in the top pane, the bottom pane will be updated with objects that are referencing this type or instance which prevent it from being garbage collected (at the time of the snapshot).

Note: By default, the view settings are set to show Just My Code.

Managed Memory (iisexpress.exe)	Compare to: View Settings 🔻	Select baseline Search	~				
O View Settings have filtered some object types (Just My Code, Collapse Small Objects)							
Object Type	Count	Size (Bytes)	Inclusive Size (Bytes) 🔻				
List <sampleleak.models.user></sampleleak.models.user>	1	72	6,144,816				
SampleLeak.Models.User	6	6,144,744	6,144,744				
> Hashtable	435	302,948	906,372				
CacheSingle	16	70,872	550,020				
BundleResponse	4	466,520	466,520				
RuntimeTypeCache	126	274,124	274,124				
RuntimeConfigurationRecord	4	1,276	267,672				
ArrayList	520	101,480	257,184				

Select a type or instance to view its reference graph.

- 21. Select the List<SampleLeak.Models.User> object from the top of the list.
- 22. The **Paths to Root** view shows that this list is rooted in the static variable
- SampleLeak.Data.UserRepository.m\_userCache.
- 23. Select the **Referenced Types** tab.
- 24. Starting with the List<SampleLeak.Models.User> object from the References list, expand all child references. With everything expanded, you should be able to see that there were a number of User instances referenced and that byte arrays are taking up most of the memory. Using this information, you could then investigate the code to see why the User instances were using so much memory, perhaps providing a big savings on memory usage.
- 25. To investigate a potential memory leak, it is useful to compare to a baseline memory dump. Select the dropdown next to the **Compare To** option and then select the **Browse** option.



- 26. In the Select File to Compare With dialog, select the iisexpress\_baseline.dmp file and then click Open.
- 27. After analysis is complete, you will see a few additional columns that show differences in object counts and sizes. Note that there are more User objects than there were in the baseline dump, and that there was a large increase in memory usage. This is a good indication that the application may be leaking User objects, so you would now be able to raise the issue with the developers who would be able to use the dump files to help pinpoint the issue and create a fix.

Managed Memory (iisexpress.	exe)				Compare to:	D:\tem	p\VS2013Traress_baseline.dmp	~
				Vie	w Settings 🔻	Search	1	
<ol> <li>View Settings have filtered some obj</li> </ol>	ect types (Just N	/ly Code, Collap	se Small Objects)					
Object Type	Count	Count Diff.	Size (Bytes)	Size Diff. (B	Inclusive Size (	Bytes)	Inclusive Size Diff. (Bytes) 🔻	
List <sampleleak.models.user></sampleleak.models.user>	1	٥	104	+32	13,	313,716	+5,120,652	
SampleLeak.Models.User		+5	13,313,612	+5,120,620		313,612	+5,120,620	
CacheSingle	16	0	116,028	+13,440		541,596	+28,228	
Hashtable	518	+1	462,592	+106,840	1)	008,492	+10,912	
RuntimeType	27,089	0	765,064	+376		765,064	0	
BundleResponse	4	0	466,496	0		466,496	0	
RuntimeTypeCache	0	-129	0	-277,488		0	-277, <b>4</b> 88	
RuntimeConfigurationRecord	0	-4	0	-1,276		0	-295,776	
ArrayList	0	-597	0	-119,992		0	-314,572	

28. You can now stop debugging and close Visual Studio.

## Exercise 4: Other Debugging Improvements

In this exercise, you will take a quick look at some asynchronous debugging enhancements to Visual Studio 2013, as well as automatic method return value inspection.

#### Task 1: Asynchronous Debugging

In this task, you will learn about asynchronous debugging enhancements that have been made in Visual Studio 2013 that make it easier to understand and follow asynchronous tasks.

- 1. Open the BlankXamlApp.sln solution file found in the lab's Source\BlankXamlApp folder.
- 2. Open the MainPage.xaml.cs code file, locate the myGridView\_SelectionChanged event handler, and un-comment the line of code that calls the asynchronous DoWork method. The DoWork method is an asynchronous method which itself makes use of additional asynchronous code.



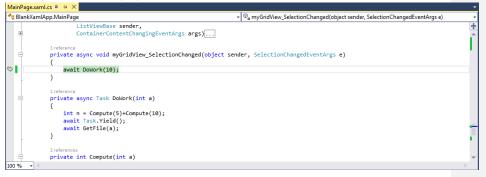
- 3. Press F5 to start debugging the application.
- 4. Select one of the tiles shown in the application to exercise the DoWork method.



 Visual Studio should show a dialog window describing an unhandled exception in the application. This shows that the application tried to access a file named **10.png**, but that it was not found. Click the **Break** button.

Microsoft Visual Studio							
	An exception of type 'System.IO.FileNotFoundException' occurred in mscorlib.dll but was not handled in user code						
	WinRT information: An item cannot be found with the specified name (10.png).						
	Additional information: The system cannot find the file specified.						
	If there is a handler for this exception, the program may be safely continued.						
Break when this exception type is user-unhandled Open Exception Settings							
	Break Continue Ignore						

 After breaking into the debugger, you should see that Visual Studio has opened MainPage.xaml.cs and highlighted the call to DoWork.



7. Developers typically rely on the Call Stack window to tell them how their application got to their current location, but this was not the case for asynchronous calls prior to Visual Studio 2013 and Windows 8.1. The latest enhancements made to the call stack window for asynchronous debugging provides additional stack frames to aid in understanding how the program reached a location inside an asynchronous call. These enhancements work across all of the languages that Visual Studio supports for Windows app development (C++, JavaScript, C#/VB).

- In managed code, the async and await <u>pattern</u> to asynchronous programming created situations where you had asynchronous methods awaiting other asynchronous methods to return, but call stack information was not made available until now.
- 9. In the **Call Stack** window (**Debug | Windows | Call Stack**), you can see that the debugger is currently set to the call stack where the async **DoWork** method is called, within the selection changed handler.

C	Call Stack						
	Name	Language					
	[External Code]						
\$	BlankXamlApp.exelBlankXamlApp.MainPage.myGridView_SelectionChanged(object sender, Windows.UI.Xaml.Controls.SelectionChangedEventArgs e) Line 185	C#					
	[Resuming Async Method]						
	[External Code]						
	BlankXamlApp.exelBlankXamlApp.MainPage.DoWork(int a) Line 192	C#					
	[Resuming Async Method]						
	[External Code]						
	BlankXamlApp.exelBlankXamlApp.MainPage.GetFile(int a) Line 205	C#					
	[Resuming Async Method]						
	[External Code]						

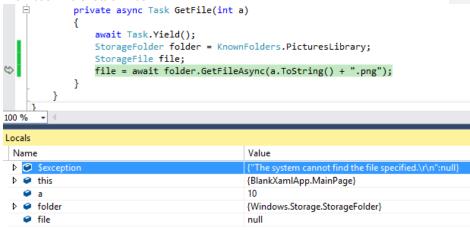
10. Also note that the additional call stack frames that have been added to the Call Stack window, namely the **DoWork** and **GetFile** methods. In managed code, Visual Studio shows the asynchronous methods that are awaiting the current asynchronous method, rather than the call stack when the task was created, as is the case for <u>C++</u> and JavaScript debugging.

Note: The call stack from the screenshot above is showing 'Just My Code'. If you see a lot of additional external code in the Call Stack window, you can right-click on the window and **de-select** the **Show External Code** option to get a cleaner view.

11. **Double-click** on the **DoWork** call stack frame. Note that Visual Studio takes you to the line of code that was waiting for the async call to GetFile to return.

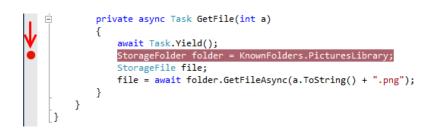
	<pre>1reference private async Task DoWork(int a) {</pre>
	<pre>int n = Compute(5)+Compute(10);</pre>
0	<pre>await Task.Yield(); await GetFile(a);</pre>
$\sim$	await Getriie(a);
Ca	II Stack
	Name
	[External Code]
	BlankXamlApp.exelBlankXamlApp.MainPage.myGridView_SelectionChanged(object s
	[Resuming Async Method]
	[External Code]
6	BlankXamlApp.exelBlankXamlApp.MainPage.DoWork(int a) Line 192
	[Resuming Async Method]
	[External Code]
	BlankXamlApp.exe!BlankXamlApp.MainPage.GetFile(int a) Line 205
	[Resuming Async Method]
	[External Code]

- 12. Hold the mouse cursor over the parameter sent to the **GetFile** method. This confirms that the filename requested from the GetFile method was '**10**'.
  - 1reference
    private async Task DoWork(int a)
    {
     int n = Compute(5)+Compute(10);
     await Task.Yield();
     await GetFile(a);
    }
- 13. **Double-click** on the **GetFile** call stack frame. This shows how the final folder and path were constructed and the location where the exception originated. You can view additional diagnostic information in the **Locals** window.



## 14. Select Debug | Stop Debugging.

- 15. Now you will take a quick tour of the asynchronous debugging improvements through the **Tasks** window. The Parallel Tasks window was introduced in Visual Studio 2010, but has been enhanced and renamed to be just the Tasks window in Visual Studio 2013. This can be useful for debugging hung and excessively long tasks, and like the upgraded call stacks, this is supported for all languages that are supported in Visual Studio for developing Windows Store apps.
- 16. Set a breakpoint on the second line of the GetFile method.



- 17. Press **F5** to start debugging and then select one of the tiles once again.
- 18. The breakpoint should be hit and Visual Studio will break into the debugger.
- 19. The Tasks window (Debug | Windows | Tasks) should show two tasks that are in the Awaiting



Task	Tasks 👻 🕂 🖓							
77	*							
		ID	Status	Start Time	Duration	Location Task		
۷		81	( Awaiting	1.820	0.737	BlankXamlApp.MainPage.myGridView_SelectionChanged()	BlankXamlApp.MainPage.myGridView_SelectionChanged()	
۲	٩	79	( Awaiting	1.819	0.738	BlankXamlApp.MainPage.DoWork()	BlankXamlApp.MainPage.DoWork()	
٣	⇔	82	Active	1.825	0.732	BlankXamlApp.MainPage.GetFile	Async: <getfile>d_c</getfile>	
Mair	nPage	KamlAp	s → × p.MainPage			<ul> <li>♥<sub>a</sub> GetFile(int a)</li> </ul>		•
0	<pre>inference private async Task GetFile(int a) { await Task.Vield(); StorageFile file; file = await folder.GetFileAsync(a.ToString() + ".png"); } }</pre>							

20. Hold the mouse cursor over one of the tasks in the **Awaiting** state to see which task is being awaited.

Task	s								
$\overline{\nabla^{*}}$									
		ID	Status	Start Time	Duration	Location			
¥		81	Awaiting	1.820	0.737	BlankXamlApp.MainPage.myGridView_SelectionChanged()			
¥	⇔	79	( Awaiting	1.819	0.738	BlankXamlApp.MainPage.DoWork()			
¥	⇔	82	Activ	1 075		BlankyamlApp.MainPage.GetFile			
	"Task 79" is waiting on this object: "Task 82" (Owned by thread 24500).								

21. The **Start Time** represents the time when the task was created relative to the time that you started debugging. The **Duration** is how long that the task has been running for. This information can help you understand the execution order of your asynchronous tasks and to find which ones may have been running for longer than expected.

Tasks							
$\overline{\nabla^{i}}$							
_		ID	Status	Start Time	Duration		
¥		81	Awaiting	1.820	0.737		
¥	⇔	79	Awaiting	1.819	0.738		
¥	⇔	82	Active	1.825	0.732		

22. The **Location** column shows the current location in code. By hovering over the location, you will see a call stack including the asynchronous calls. You can double-click on the individual frames to navigate to the code if desired (or use the Call Stack window).

Location		Task			
BlankXamIA	pp.MainPage.myGridView_SelectionChanged()	BlankXamlApp.MainPage.myGridView_SelectionChanged()			
BlankXamIA	.pp.MainPage.DoWork()	BlankXamlApp.MainPage.DoWork()			
BlankXam	BlankXamlApp.exe!BlankXamlApp.MainPage.GetFile(int a) Line 203 [Resuming Async Method] [External Code] [Async Call] BlankXamlApp.exe!BlankXamlApp.MainPage.DoWork(int a) Line 192 [Async Call] BlankXamlApp.exe!BlankXamlApp.MainPage.myGridView_SelectionChanged(object sender, Windows.UI.Xaml.				

23. The **Task** column helps you identify and distinguish between different tasks. For example, "Async: <GetFile> d\_c" is the lambda function in the continuation of the GetFile asyncronous

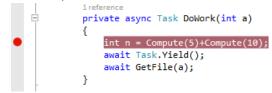
method.				
Task				
BlankXamlApp.MainPage.myGridView_SelectionChanged()				
BlankXamlApp.MainPage.DoWork()				
Async: <getfile>d_c</getfile>				

## 24. Select Debug | Stop Debugging.

#### Task 2: Method Return Value Inspection

In this task, you will take a look at an enhancement added to the Autos window in Visual Studio 2013 that enables you to quickly determine the return values for functions. This is particularly useful for situations where the return values are not stored in local variables or you are using nested function calls.

1. Set a breakpoint on the **first** line of the **DoWork** method.



- 2. Press F5 to start debugging and then select one of the tiles once again.
- 3. In the **Autos** window (**Debug | Windows | Autos**), note that everything appears as expected with local variables and the current class instance.

	{ int n await	<pre>sync Task DoWork(int a) = Compute(5)+Compute(10); Task.Yield(); GetFile(a);</pre>	
Autos			
Name		Value	Туре
🤗 a		10	int
🤗 n		0	int
👂 🧭 this		{BlankXamIApp.MainPage}	BlankXamIApp.MainPage

- 4. Press **F10** once to step over the line that computes a value and assigns the result to the local variable 'n'.
- 5. Note that the **Autos** window now automatically shows what the two calls to the Compute method returned. This is what is referred to as method return value inspection.

Autos							
Name	Value	Туре					
° 🥥 BlankXamlApp.MainPage.Compute returned	25	int					
° 🥥 BlankXamlApp.MainPage.Compute returned	100	int					
🤗 n	125	int					
👂 🧉 this	{BlankXamlApp.M	a BlankXamlApp.MainPage					

6. Stop debugging and close Visual Studio.