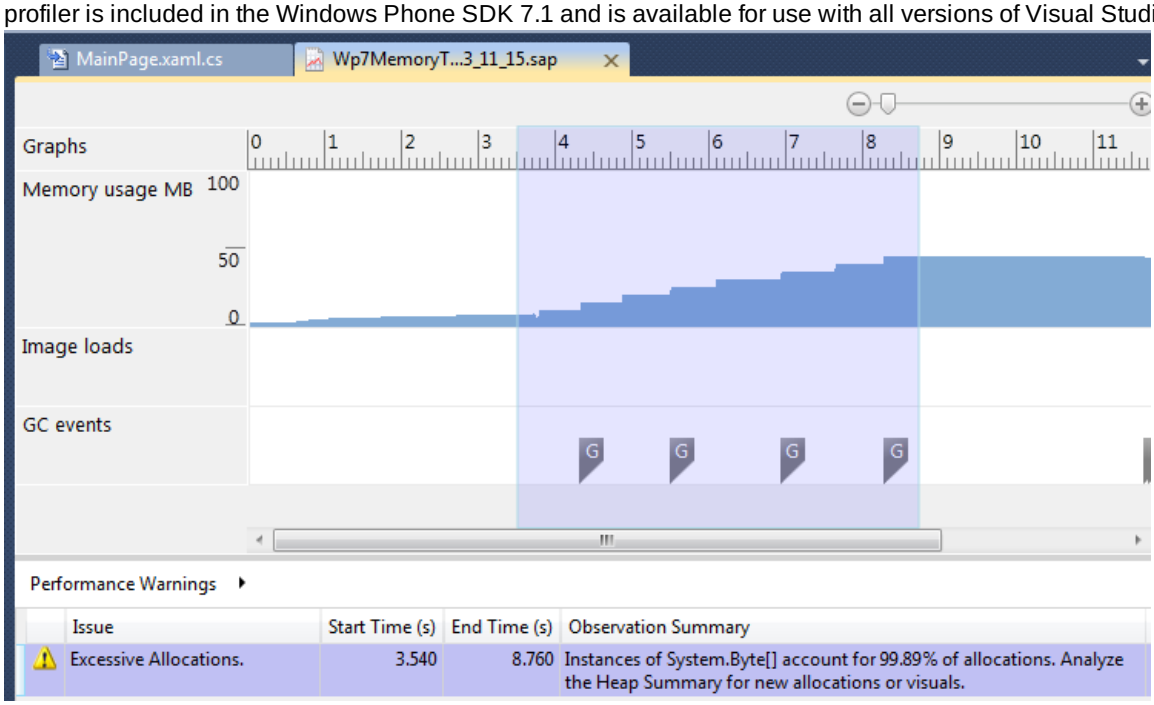
Techniques for memory analysis of Windows Phone apps

This article provides an overview of the techniques available for analysing the memory use of Windows Phone apps.

18 Mar 2012

Windows Phone Memory Profiler

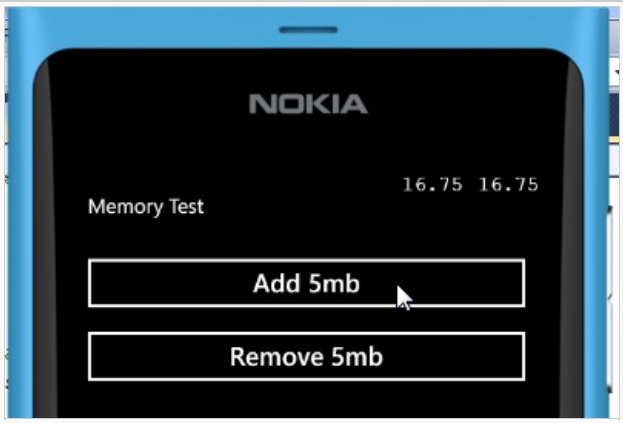
Run the *Windows Phone Memory Profiler* to get a time-axis graph of application memory consumption, a breakdown of important events (garbage collection, image creation, etc.), and get automated analysis of your application's memory use. The profiler is included in the Windows Phone SDK 7.1 and is available for use with all versions of Visual Studio.



The following section describes a simple test app that we can use to allocate and deallocate memory during profiling, followed by a section showing how the Memory profiler is used.

Sample app for testing memory analysis

To demonstrate how the profiler works we first create a sample Windows Phone 7 app that allocates and deallocates 5MB of memory at a time. This sample application is used throughout the article to demonstrate the different memory analysis techniques.



Sample app for testing memory analysis screenshot

Create a new Windows Phone 7.5 project after installing the Windows Phone SDK 7.1.1. Open up **MainPage.xaml** and replace existing “ContentPanel” with a stackpanel and 2 buttons.

```
<!--ContentPanel - place additional content here-->
<StackPanel x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <Button Content="Add 5MB" Click="Add_Click" />
    <Button Content="Remove 5MB" Click="Remove_Click" />
</StackPanel>
```

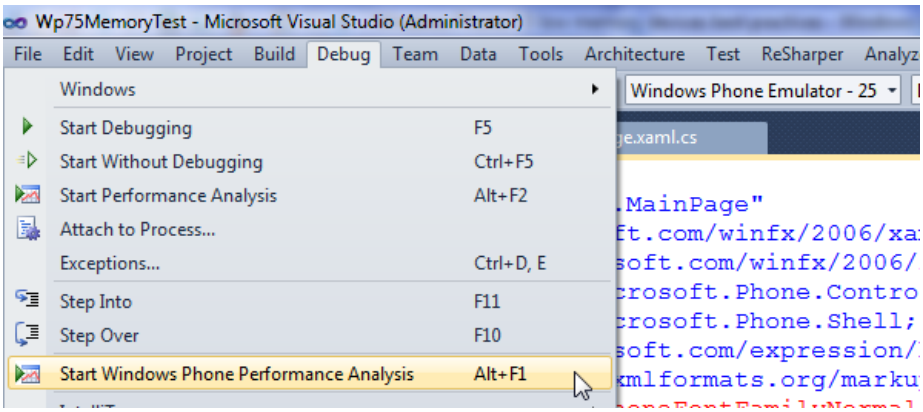
Next, implement the event handlers in **MainPage.xaml.cs** to allocate and deallocate 5MB of memory.

```
List<Byte[]> inMemoryCache = new List<byte[]>();

private void Add_Click(object sender, RoutedEventArgs e)
{
    inMemoryCache.Add(new Byte[1024 * 1024 * 5]);
}

private void Remove_Click(object sender, RoutedEventArgs e)
{
    if (inMemoryCache.Count > 0)
        inMemoryCache.RemoveAt(inMemoryCache.Count - 1);
    GC.Collect();
}
```

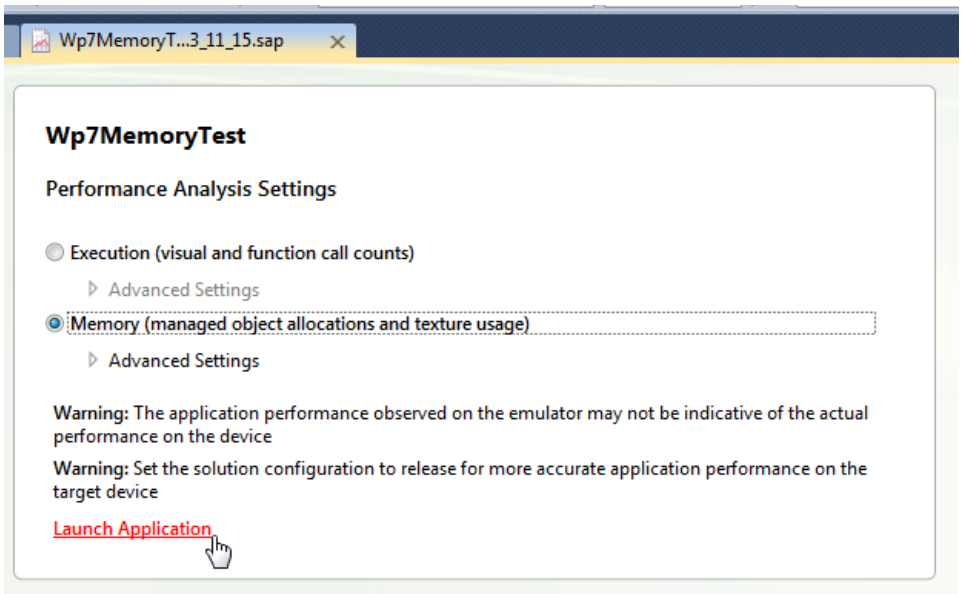
Profiling

In order to start memory profiling:

1. Open the top **Debug** menu item and choose **Start Windows Phone Performance Analysis.**

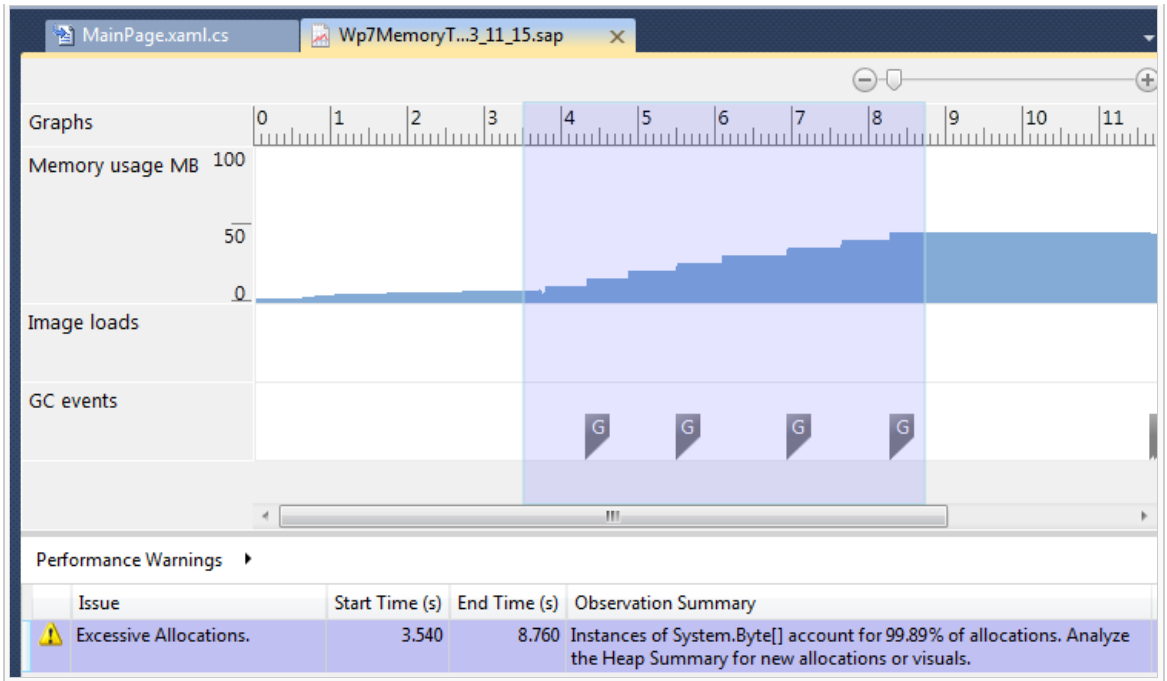


   - Make sure to select "Memory" instead of "Execution" profiling and launch the application.



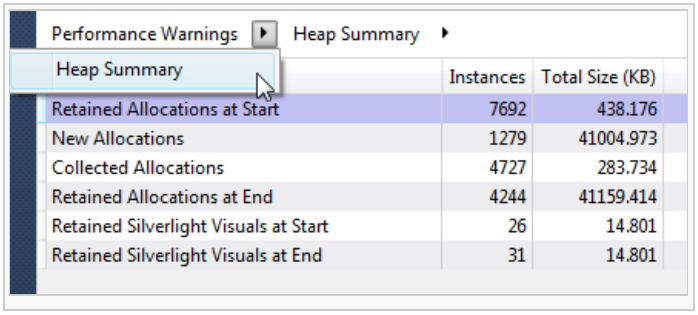
2. Once the app is running click the **Add 5MB** button a few times
3. Then select **Stop Profiling**

You can see the app's memory allocation graph and markers for when garbage collection took place. Select the time span you're interested in and select **Start Analysis**.

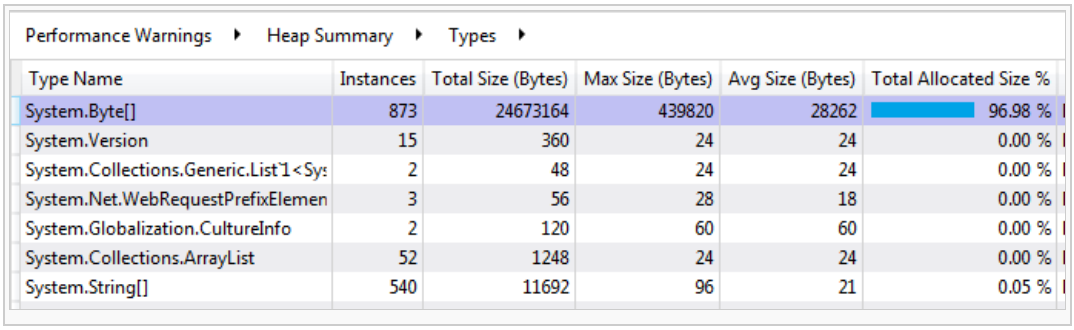


Memory allocation graph

The analysis of the selected time span shows that too many `Byte[]` have been allocated. We can also examine the Heap Summary as was suggested and see a summary of the Silverlight/CLR memory allocation state.



We can even drill all the way down to see how many `Byte[]` were allocated during this time and their total allocated size %.

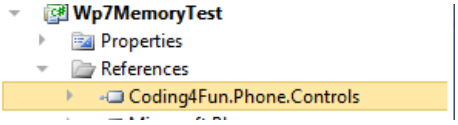


You can also use Windows Phone 7 Performance Analysis for CPU profiling. For more information see How to: Capture and Analyze Performance Data Using Windows Phone Performance Analysis (MSDN).

Add on-screen coding4fun <MemoryCounter />

Use the `<MemoryCounter />` control in the Microsoft coding4fun Windows Phone Toolkit to display the current memory use and peak memory while developing your app.

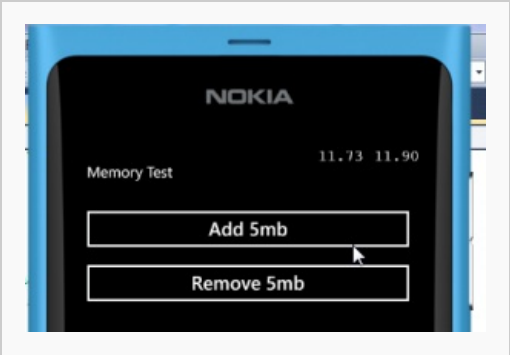
Start by adding the **Coding4fun** assembly to the project. The coding4fun website has instructions on how to add the correct DLLs to your project either by adding references or by using NuGet.



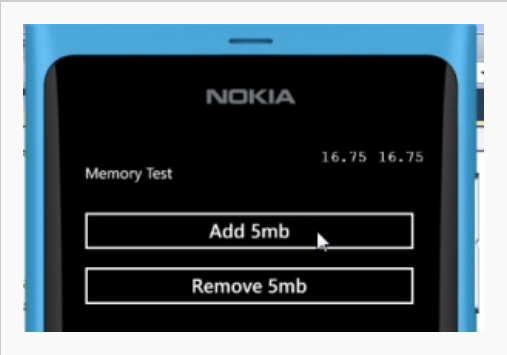
After adding the `Coding4Fun.Phone.Controls` assembly reference to your project (and unblocking it if needed) you can now add a `<MemoryCounter />` to your page.

```
<!--ContentPanel - place additional content here-->
<StackPanel x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <Button Content="Add 5MB" Click="Add_Click" />
        <Button Content="Remove 5MB" Click="Remove_Click" />
</StackPanel>
<coding4fun:MemoryCounter
    xmlns:coding4fun="clr-
namespace:Coding4Fun.Phone.Controls;assembly=Coding4Fun.Phone.Controls"/>
```

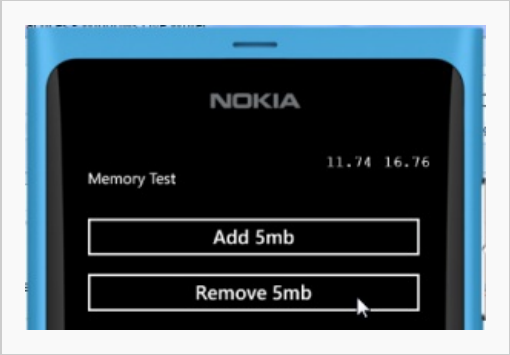
The screenshots below show the above code running in our sample test app. Note how the total and peak memory changes when the users clicks the **Add 5mb** and **Remove 5mb** buttons.



Adding 5MB



Adding 5MB



Removing 5MB

Note: Passing WP7 Memory Consumption requirements with the Coding4Fun MemoryCounter tool (WindowsPhoneGeek) fully explains `<MemoryCounter />`. The sample code above is actually an adaptation from the sample code in that article.

Create your own memory profiler

Windows Phone provides low level reporting properties that you can use to create your own "pseudo" memory profiler. This might be useful if you want to get memory profiling logs from consumer devices.

Warning: Be careful when deploying your own memory profiler. Excessive sampling of memory conditions might itself impact memory and CPU performance adversely.

The `DeviceStatus` class has four of the properties needed:

```
namespace Microsoft.Phone.Info
{
    public static class DeviceStatus
    {
        public static long ApplicationCurrentMemoryUsage { get; }
        public static long ApplicationPeakMemoryUsage { get; }
        public static long ApplicationMemoryUsageLimit { get; }
```

```
        public static long DeviceTotalMemory { get; }
    }
}
```

In addition, in the Windows Phone 7.5 upgrade you can call `DeviceExtendedProperties.GetValue("ApplicationWorkingSetLimit")` to get the memory working set limit for the current device.

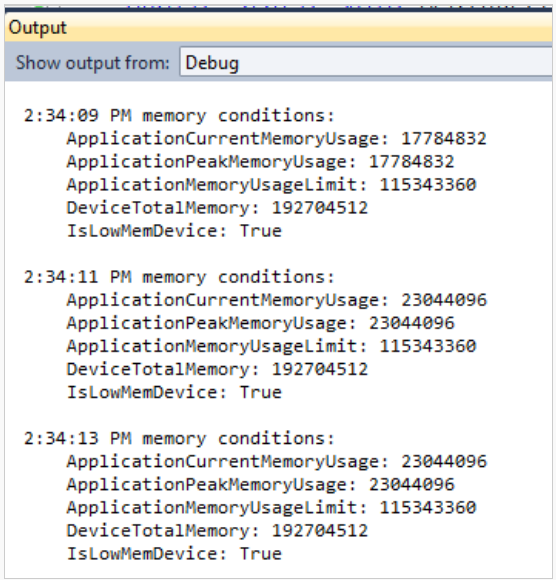
Warning: Note that if the Windows Phone 7.5 update isn't installed then the `GetValue()` invocation will `throw` an exception.

An example pseudo memory profiler is given below. This creates an asynchronous timer that records memory conditions and writes them to a report in an `IsoStore` every 2 seconds. In an ideal world the example would write the report in `IsoStore` to a web service next time the app restarts - for the sake of brevity this code is omitted.

```
public static class LowMemoryHelper
{
    private static Timer timer = null;

    public static void BeginRecording()
    {
        // before we start recording we can clean up the previous session.
        // e.g. Get a logging file from IsoStore and upload to the server

        // start a timer to report memory conditions every 2 seconds
        timer = new Timer(state =>
            {
                // every 2 seconds do something
                string report =
                    DateTime.Now.ToLongTimeString() + " memory conditions: " +
                    Environment.NewLine +
                    "\tApplicationCurrentMemoryUsage: " +
                        DeviceStatus.ApplicationCurrentMemoryUsage +
                        Environment.NewLine +
                    "\tApplicationPeakMemoryUsage: " +
                        DeviceStatus.ApplicationPeakMemoryUsage +
                        Environment.NewLine +
                    "\tApplicationMemoryUsageLimit: " +
                        DeviceStatus.ApplicationMemoryUsageLimit +
                        Environment.NewLine +
                    "\tDeviceTotalMemory: " + DeviceStatus.DeviceTotalMemory +
Environment.NewLine +
                    "\tApplicationWorkingSetLimit: " +
                        DeviceExtendedProperties.GetValue("ApplicationWorkingSetLimit")
+
                        Environment.NewLine;

                // write to IsoStore or debug conolse
                Debug.WriteLine(report);
            },
            null,
            TimeSpan.FromSeconds(2),
            TimeSpan.FromSeconds(2));
    }
}

// Code to execute when the application is launching (eg, from Start)
// This code will not execute when the application is reactivated
```

```
private void Application_Launching(object sender, LaunchingEventArgs e)
    {
        LowMemoryHelper.BeginRecording();
    }
```

The output of the sample test app after a user as selected the **Add 5MB** button a few times is shown below. We can see that the memory conditions debug output reflects those changes.



Example output from "home grown" memory profiler

Note: The coding4fun `<MemoryCounter />` described in the preceding section is just a "pseudo" memory profiler, like the one described here. The full source code implementation for `<MemoryCounter />` is at http://coding4fun.codeplex.com/SourceControl/changeset/view/72318#1113297

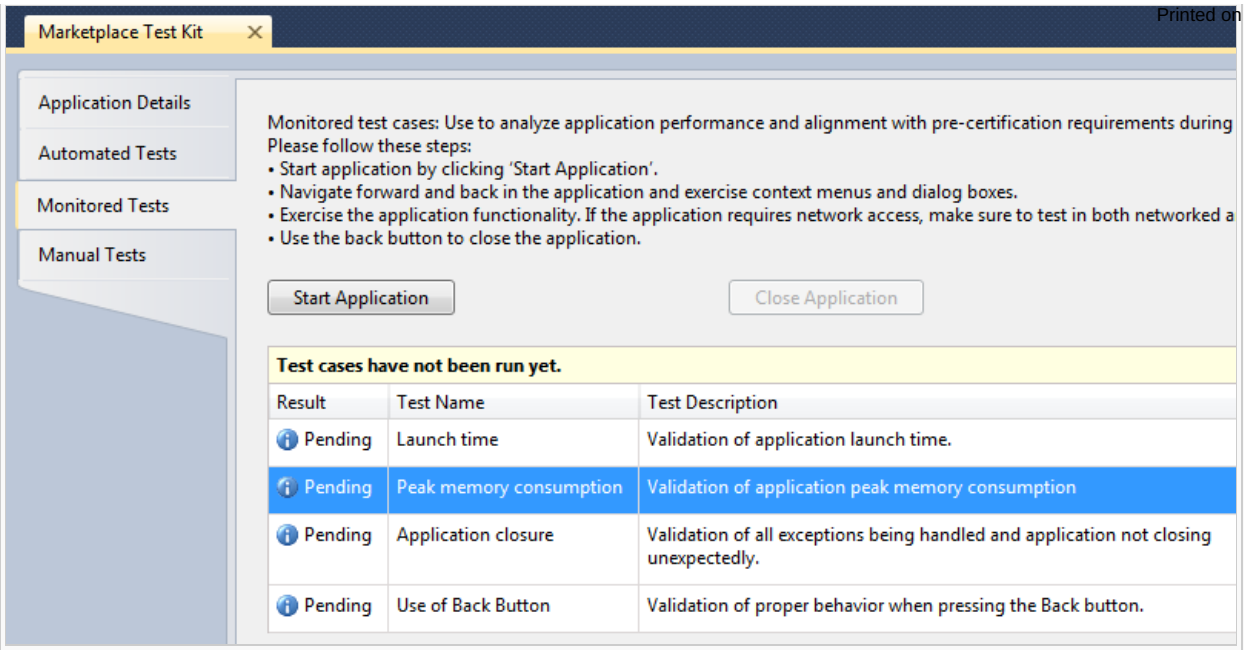
Run the Windows Phone Marketplace Test Kit

The Windows Phone SDK 7.1's Windows Phone Marketplace Test Kit includes a *Peak Memory Consumption* test which monitors whether the total peak memory for an app is under 90 MB. While we highly recommend running the Marketplace Test Kit on your app prior to marketplace submission, the tests are also useful during earlier stages of development.

To run the tests:

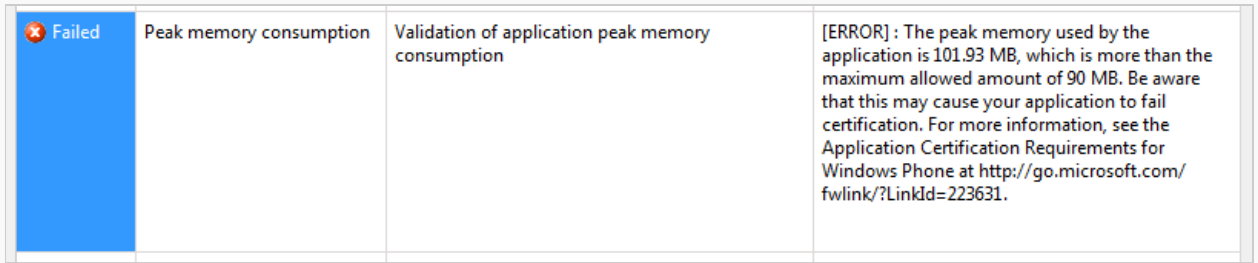
- Right click on your project in **Solution Explorer** and choose **Open Marketplace Test Kit**.

  Under **Monitored Tests** you'll see the **Peak Memory Consumption** test.
- Click **Start Application** to run the test
- Test all app functionality
- Click **Close Application** to complete the test.



Screenshot of the Peak Memory Consumption test

If your application uses more than 90MB at any point a test failure will be reported, as shown in the screenshot below.



Screenshot of a peak memory consumption test failure

For more information on the Marketplace Test Kit and memory testing see Get Your Windows Phone Applications in the Marketplace Faster (Cheryl Simmons) and Windows Phone Marketplace Test Kit (MSDN).

Check the AppHub crash reports

Whenever a deployed app crashes on a consumer device, a crash report is sent back to AppHub. An `OutOfMemoryException` in an AppHub crash report indicates that your app has a memory problem that should be fixed.

You can inspect crash reports for your apps through your AppHub account, filtering reports to only display crashes in a specific date range. The reports include information about what exceptions have occurred, and in what functions, along with a stack trace of the function calls that lead up to them. As an example, the screenshot below shows a real report for an `OutOfMemoryException` that occurred 18 times in the field along with the associated stack trace.

F2

Frame	Image	Function	Offset
0	coredll.dll	xxx_RaiseException	19
1		0	
2		0	
3		0	
4		TransitionStub	0
5		.AgentRequest.Complete	224
6		.AgentRequest.OnAgentComplete	132
7		Microsoft.Phone.BackgroundAgent.FireRequestComplete	108
8		MyApp.Background.ScheduledAgent.Complete	64
9		.__c__DisplayClasse._OnInvoke_b__4	72
10		.__c__DisplayClass14._CheckForUpdates_b__10	1332
11		.__c__DisplayClass29._WithHammockImplRaw_b__28	152
12		Hammock.RestClient.CompleteWithQuery	260
13		Hammock.RestClient.QueryResponseCallback	388
14		.__c__DisplayClass13._BeginRequestImpl_b__10	96
15		Hammock.Web.WebQuery.OnQueryResponse	92
16		Hammock.Web.WebQuery.GetAsyncResponseCallback	944
17		.__c__DisplayClassa._InvokeGetResponseCallback_b__8	76
18		.WorkItem.WaitCallback_Context	104
19		System.Threading.ExecutionContext.Run	324

A	B	C	D	E	
App Name	Phone Version	Problem Function	Exception Type	Crash Count	Stack Trace
MyApp	Mango	.AgentRequest.Complete	OutOfMemoryException	18	0 coredll.dll
MyApp	Mango	Unknown	OutOfMemoryException	32	0 coredll.dll

Crash report courtesy Morten Nielsen.

Tip: Don't rely on AppHub crash reports as a way to **validate** your apps for 256 MB devices! Firstly, it is much better to detect errors before your users have a bad experience, and secondly there is no *guarantee* that the AppHub will detect all `OutOfMemoryExceptions`.

However if you see errors, they are indicative that more work is needed!