Azure Fast Start for Mobile Application Development

Module 11: Azure Application Insights

Student Lab Manual

Instructor Edition (Book Title Hidden Style)

Version 1.0

**Conditions and Terms of Use**

Microsoft Confidential

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

© 2015 Microsoft Corporation. All rights reserved.

**Copyright and Trademarks**

© 2015 Microsoft Corporation. All rights reserved.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see Use of Microsoft Copyrighted Content at  
<http://www.microsoft.com/en-us/legal/intellectualproperty/Permissions/default.aspx>

Azure, Microsoft, Internet Explorer, Visual Studio, and Windows® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

Contents

[Lab 1: Application Insights for Mobile Apps 6](#_Toc432602205)

[Exercise 1: Get started 7](#_Toc432602206)

[Exercise 2: Tracking page view and custom events 11](#_Toc432602207)

[Exercise 3: Add traces and capture exception 16](#_Toc432602208)

[Exercise 4: Detect and diagnose crashes 19](#_Toc432602209)

# Lab 1: Application Insights for Mobile Apps

#### Introduction

Application Insights (AI) is an extensible analytics service that helps you understand the performance and usage of your live application. It is designed for developers to help you continuously improve the performance and usability of your app.

It works with both web and stand-alone apps on a wide variety of platforms: .NET or J2EE, hosted on-premises or in the cloud, device apps on Windows, iOS, Android, Macintosh OS X, and other platforms. AI currently supports iOS, Android, and Windows apps, J2EE and ASP.NET web applications, Windows Communication Foundation (WCF) services. Web Apps can run on Microsoft Azure or your own on-premises servers. The AI JavaScript software development kit (SDK) runs in any web pages. Application Insights works by adding an SDK into your app, which sends telemetry to the Azure portal. There are different SDKs for the many combinations of platforms, languages, and IDEs that are supported.

In this lab, we will introduce you to AI with some simple instrumentation and show you how you can get to the information in the Azure portal.

#### Objectives

After completing this lab, you will be able to:

* Set up Application Insights for Mobile applications (Windows).
* Set up Application Insights for Web Application and API.
* Monitor your user and application performance.

#### Prerequisites

* Microsoft Visual Studio 2015 and **Azure SDK 2.7.1 or later**.
* Azure Subscription.

#### Estimated Time to Complete This Lab

60 minutes

#### Scenario

Your mobile application has been released and downloaded many times. However, now you need a way to gather telemetry data to prioritize feature development and ensure your team has a user centric approach. Furthermore, you also need to have insight when something goes wrong to quickly detect and troubleshoot your issues. To achieve that goal, you will need KPI and reliable data to rely on.

## Exercise 1: Get Started

Application Insights enables you monitor your deployed application for usage and performance.

All Windows applications—including desktop apps, background services, and worker roles—can use the Application Insights core SDK to send telemetry to Application Insights. You can also add Application Insights SDK to a class library project.

The core SDK just provides an API. Unlike the Web or device SDKs, it does not include any modules that collect data automatically. So, you have to write code to send your own telemetry. Some of the other packages such as the performance counter collector will also work in a desktop app.

#### Objectives

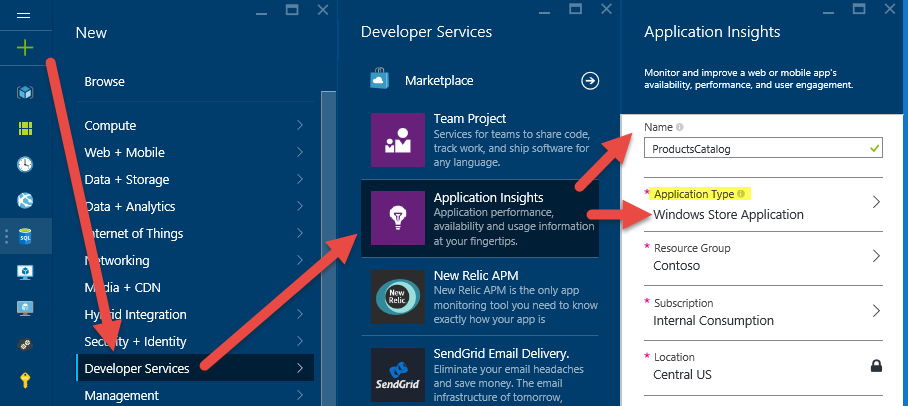
In this exercise, you will:

* Create a new Application Insights (AI) resource from the portal.
* Install AI SDK in your application.

Task 1: Create AI on Portal

Sign in to [Azure Preview Portal](http://portal.azure.com/)

Create an Application Insights resource. This is where you will see data from your app.



Choose your application type.

**Note**

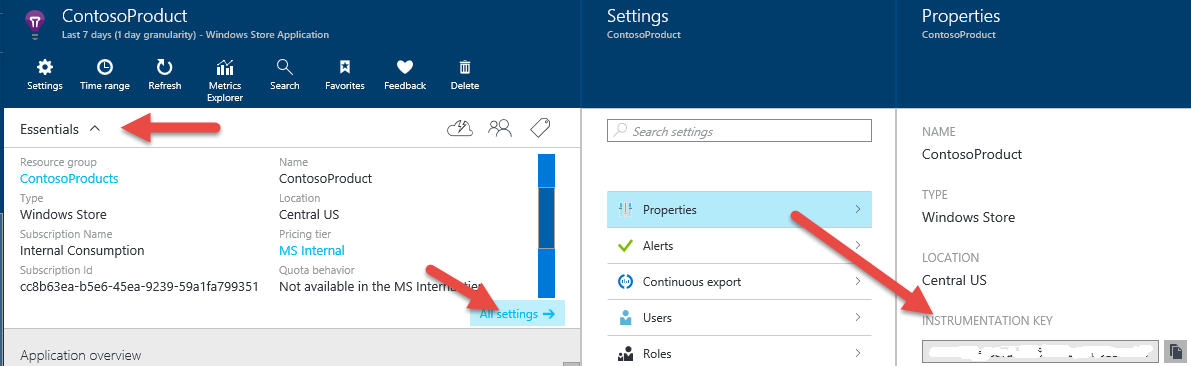
Your choice of application type affects the default selection of charts you will see.

A good practice is to have separate instances for Web and Mobile application.

Ensure you select your subscription (if you have multiple one) to find and create your resource group.

Open your new resource, and open the **Quick Start guide**.

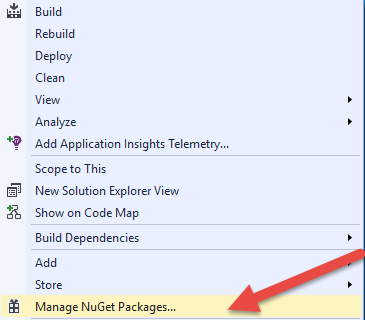
Navigate to its properties and copy the Instrumentation Key:



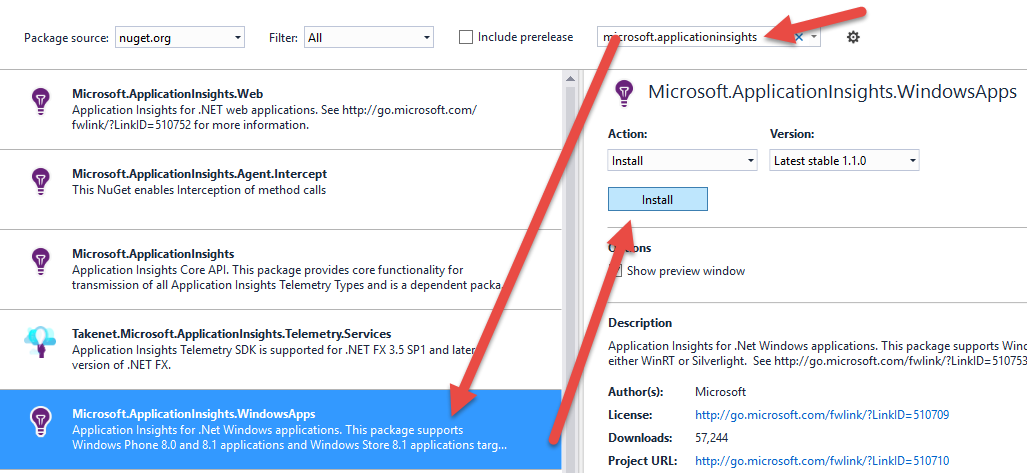
Task 2: Application Insights SDK

Open your Universal Windows solution with Visual Studio in the lab folder: Module11-ApplicationInsights\Lab1\Start\

Right-click the main project (Universal Windows) and select **Manage NuGet Packages**.



Search the SDK with **microsoft.applicationinsights** and install the WindowsApps library.



In App.xaml.cs, you can see there is a TelemetryClient and it is initialized in the constructor. Paste the key that you copied from the portal and replace the **xxxxx**.

using Microsoft.ApplicationInsights;

sealed partial class App : Application

{

public static TelemetryClient TelemetryClient;

public App()

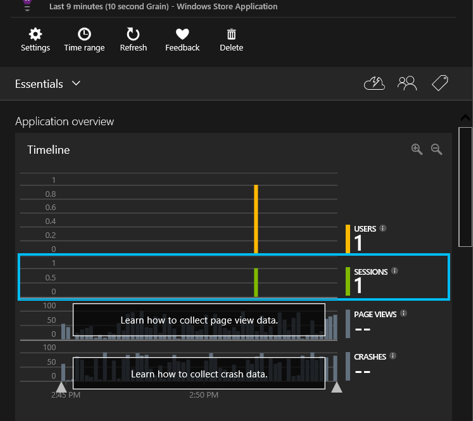
{

// App Insights Initialisation

WindowsAppInitializer.InitializeAsync("XXXXXXXXXXX");

TelemetryClient = new TelemetryClient();

When you now press **F5** to launch your app, you will see a notification in Visual Studio that the first event has been sent. Click the link. It will take you back to the Portal:



**Note** The API is uniform across all platforms, apart from a few small variations.

You will find below the list of the API’s Method (both for Web and Mobile) and what they are used for:

* **TrackPageView** Pages, screens, blades, or forms.
* **TrackEvent** User actions and other events. Used to track user behavior or to monitor performance.
* **TrackMetric** Performance measurements such as queue lengths not related to specific events.
* **TrackException** Log exceptions for diagnosis. Trace where they occur in relation to other events and examine stack traces.
* **TrackRequest**  Log the frequency and duration of server requests for performance analysis.
* **TrackTrace** Diagnostic log messages. You can also capture third-party logs.
* **TrackDependency** Log the duration and frequency of calls to external components on which your app depends.

We will use some of them in the following tasks.

## Exercise 2: Tracking Page View and Custom Events

We will focus on two particulars: User behavior and Application.

Application Insights provides you with the out-of-the-box data about how many users you have, and which pages they are looking at in your app. In addition, by inserting a few lines of code in your app, you can learn more about the paths users take through your app and what they achieve with it.

This feature is also available for desktop application.

#### Objectives

In this exercise, you will:

* Add Technical data.
* Add Functional Events.

Task 1: Add Technical Events

We are interested in the following data: **Navigation** and how often users are browsing with the application and what page are viewed.

In the **View** folder, open the file: **CommonPage.cs**

Locate the method **OnNavigatedFrom** and add the following code:

protected override void OnNavigatedFrom(NavigationEventArgs e)

{

var properties = new Dictionary<string, string>

{

{ "pagename", e.SourcePageType.Name},

{ "namespage", e.SourcePageType.Namespace}

};

var tc = new TelemetryClient();

tc.TrackEvent("Navigation", properties);

\_navigationHelper.OnNavigatedFrom(e);

}

Open the **Homepage.xaml.cs** and add the following code:

protected override void OnNavigatingFrom(NavigatingCancelEventArgs e)

{

base.OnNavigatingFrom(e);

var tc = new TelemetryClient();

tc.TrackPageView(GetType().Name);

}

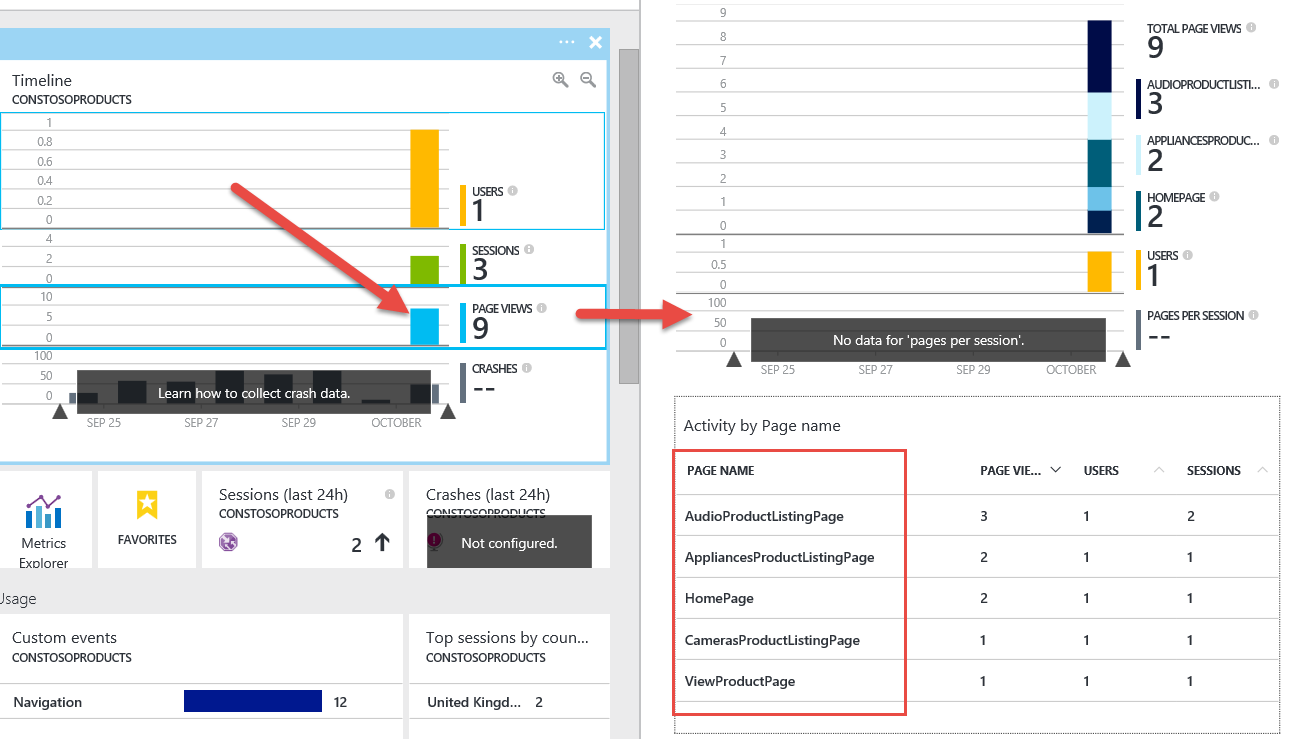
Repeat the last step and apply it to all the **\*page.xaml.cs** of the **View** folder.

Run the application (press F5) and simulate a user scenario (load different pages and products).

Close the application.

Go to **Azure Portal** in the Application Insights dashboard.

To list the application activity, on the overview graph, click **page view**.

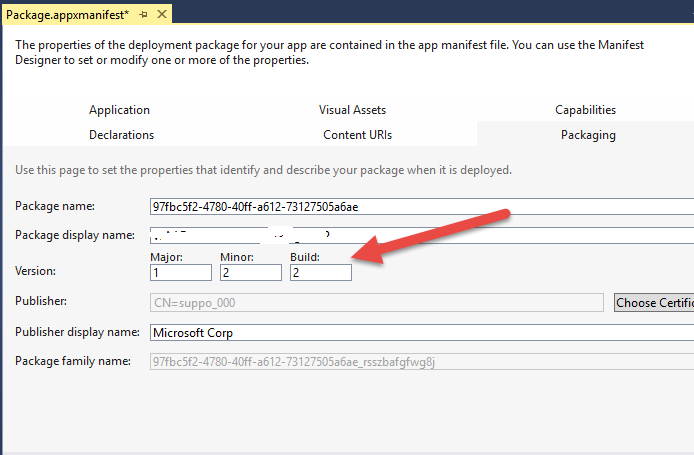


Task 2: Change Application Version

An interesting feature that AI provides is the ability to compare data between versions. From now onwards, try to execute this optional steps between Exercises or Tasks, to simulate several.

In the solution explorer, open the file **Package.appxmanifest**.

Change the version (Minor) by incrementing the current value by one.



Run the application.

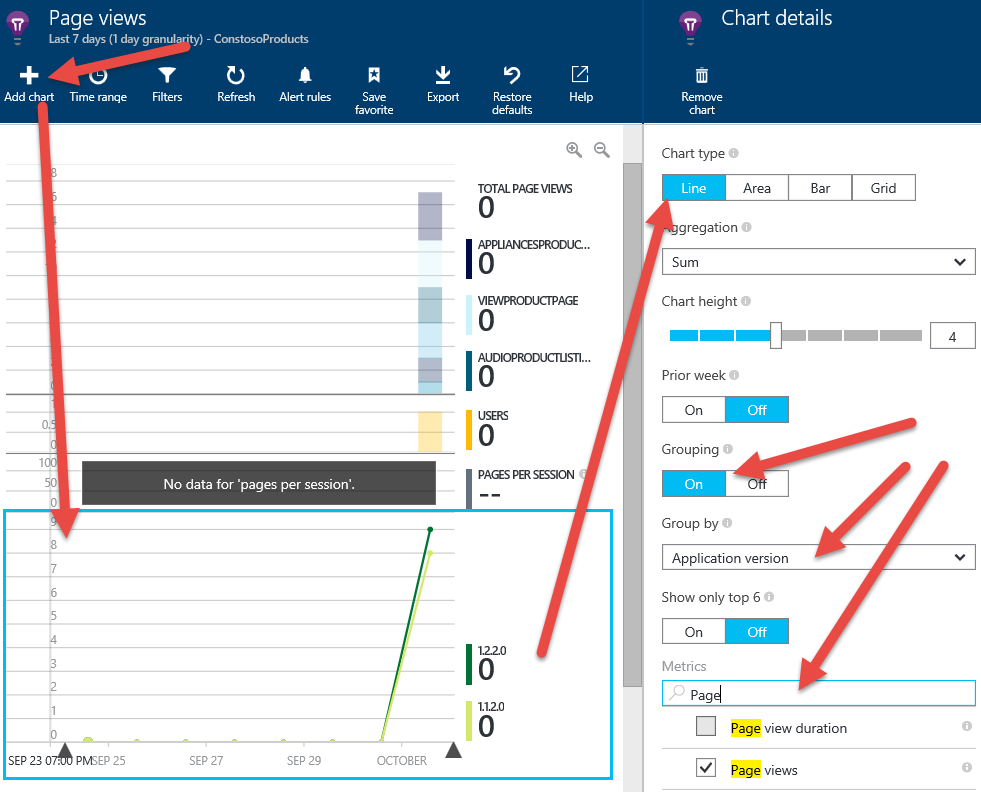
Load different pages.

Go to **Application Insights dashboard** ([the Application Insights portal](http://portal.azure.com/)).

To list the application activity (just as the previous Task), on the overview graph, click **page view**.

Add a new chart (from the menu).

1. Configure the chart (choose **Line**).
2. Enable Grouping (select **On**).
3. In the Metric pane, search for Page view (select the entry).



Task 3: Add Functional Events

Now, we will focus on some particular feature of the application. You may want to know how often they are used and whether you should keep them in the next version. Application Insights can help you take decisions based on real usage date. In this example, we will target:

Search: what term or product are users looking for

1. In the folder ViewModel, open the file: SearchResultsViewModel.cs
2. Locate the method **PerformSearch**, and add the following code in bold:

private async Task PerformSearch(string param)

{

SearchPhrase = param ?? "";

var allItems = ((param??"")=="") ? new List<Product>() : (await DataManager.ProductDataSource.SearchProductsAsync(param));

**var nbResults = allItems.Count;**

**// Provide properties by which you can filter events:**

**var properties = new Dictionary<string, string>**

**{**

**{ "Search.Param", param },**

**{ "Search.Results", nbResults.ToString() }**

**};**

**var tc = new TelemetryClient();**

**tc.TrackEvent("Search", properties);**

1. Then, locate the following code:

SearchPhrase = param ?? "";

var allItems = ((param??"")=="") ? new List<Product>() : (await DataManager.ProductDataSource.SearchProductsAsync(param));

if (allItems?.Any() ?? false)

{

1. Insert the telemetry code below:

// Track nb search results

tc.TrackMetric("Search result", nbResults , properties);

// We want to track case of exact match

if (nbResults == 1)

{

var product = allItems.FirstOrDefault();

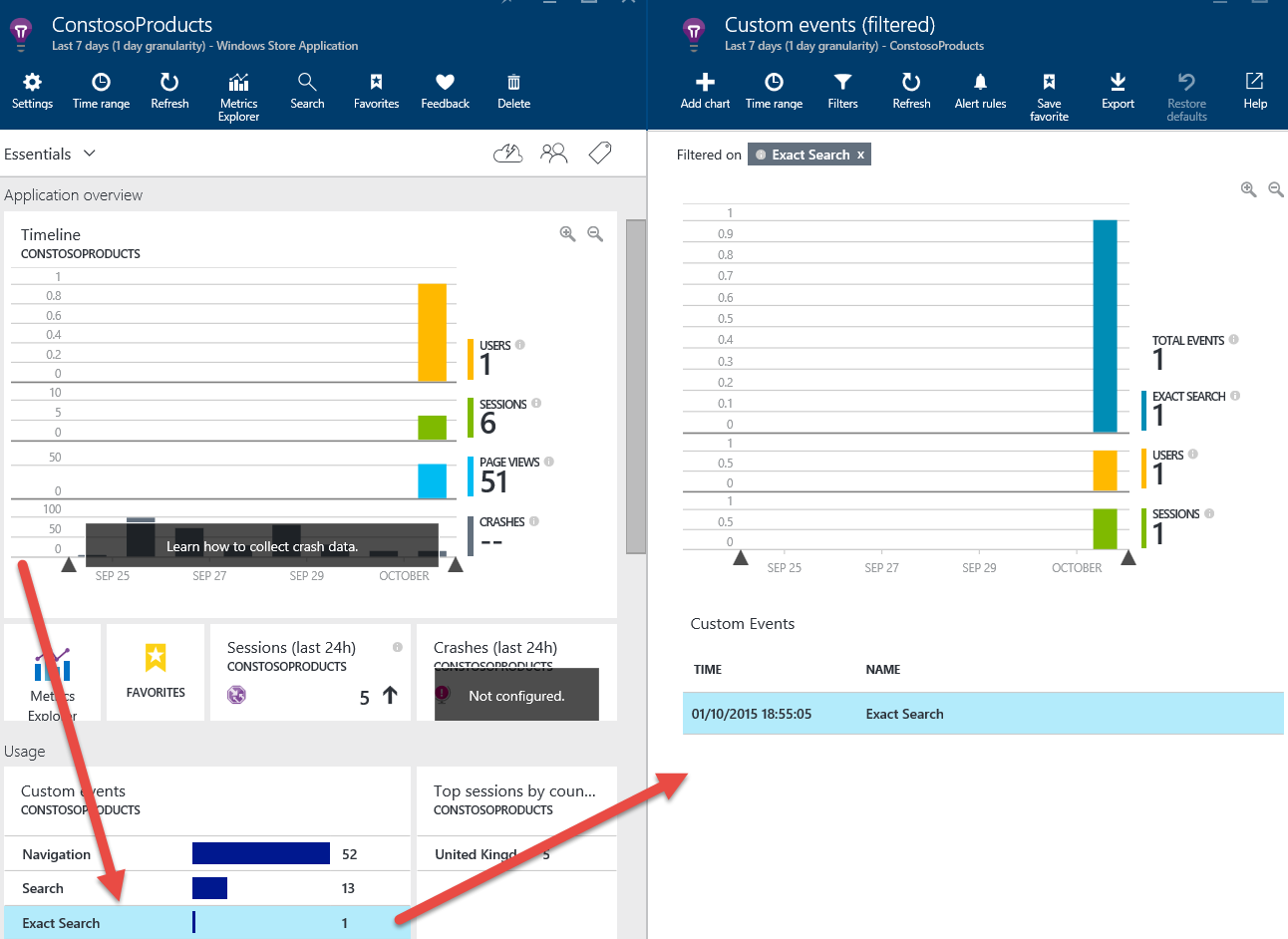
properties.Add("Product.Id", product.Id);

properties.Add("Product.Name", product.Name);

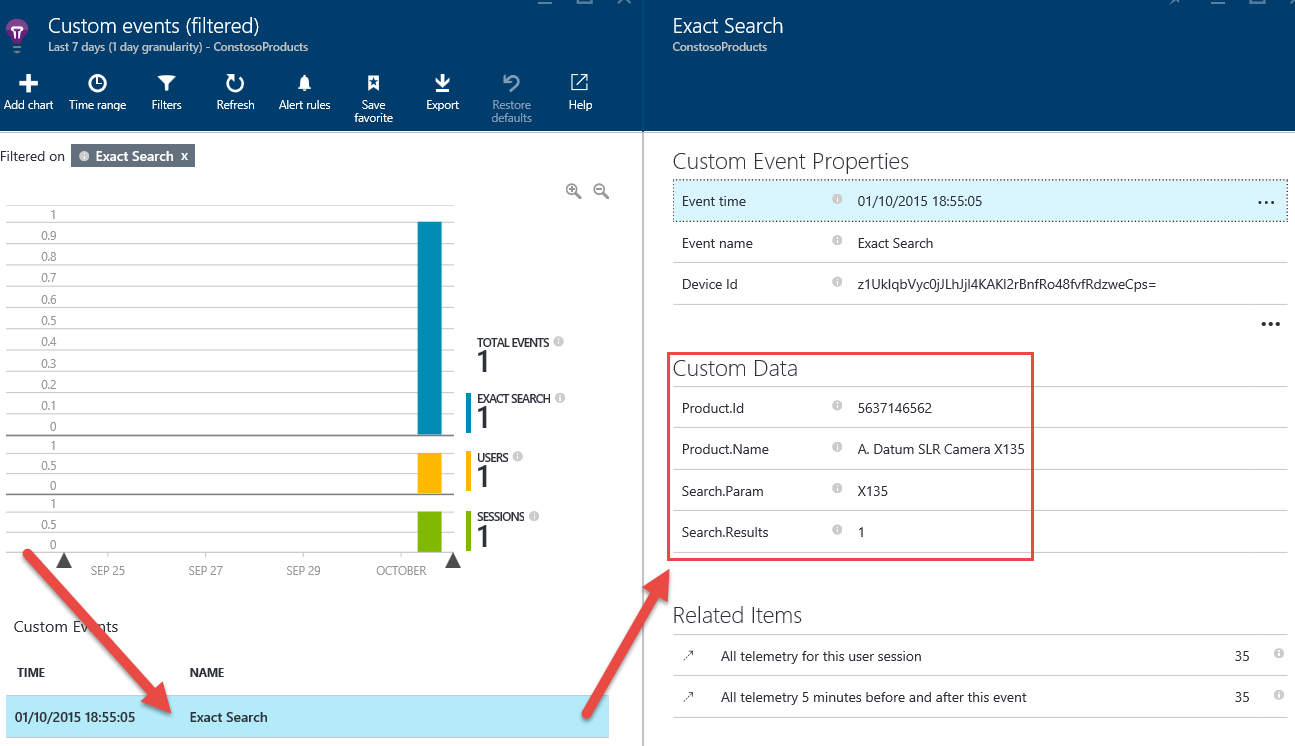
tc.TrackEvent("Exact Search", properties);

}

1. Now, press **F5** and run the app. Perform some queries like car, audio, fabrikam and X135 for an exact match
2. Go back to the Azure portal and click **Refresh** on the Application Insights page for your app. You will now have some of the events displayed under Diagnostic search.



1. To see the details, in the grid, click one of the entry:



## 

## Exercise 3: Add traces and capture exception

If your users experience crashes in your app, you would like to know about it quickly, and you would like details about what happened. With Application Insights, you can monitor how frequently crashes occur, get alerts when they occur, and investigate reports of individual incidents.

#### Objectives

In this exercise, you will:

* Add Traces.
* Capture exception and context to prepare troubleshooting.

Task 1: Add Traces

Application Insights is not a logging framework. However, if you are not using one or you cannot use **System.Diagnotics.Trace**, you can still use the telemetry API and use the TrackTrace method to collect trace information and additional context data. We will show our method to achieve this goal with a few lines of code.

1. Open the App.xaml.cs file of your Mobile App and add the following code:

protected override async void OnLaunched(LaunchActivatedEventArgs e)

{

var tc = new TelemetryClient();

var properties = new Dictionary<string, string>()

{

{"LocalUserTime", DateTime.Now.ToString()},

{"Day",DateTime.Today.DayOfWeek.ToString()},

{"DebuggerIsAttached",Debugger.IsAttached.ToString()}

};

tc.TrackTrace("App Start", SeverityLevel.Verbose, properties);

**Note** In this example, note the sample code use and overload of the **TrackTrace** method that make use of properties and the severity level. Both options are optional and have overload that enable you to send simpler traces.

Include the namespace at the top section of the file with the other using statements:

using Microsoft.ApplicationInsights;

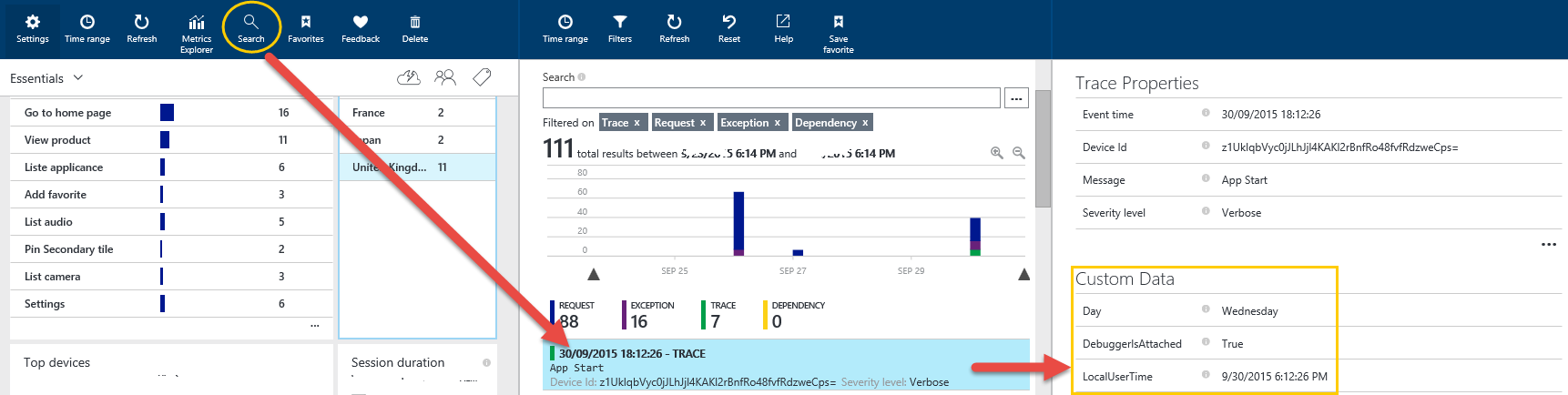
using Microsoft.ApplicationInsights.DataContracts;

using System.Collections.Generic;

Go to the portal in your AI Instance.

On the top menu, click **Search**.

1. Select a trace entry.
2. Click to see the details. A new blade will appear, where you will find all the context data you gathered in the trace.



Task 2: Capture Exception

One of the strength of AI is to enable to correlate information to obtain better results. This is especially true for troubleshooting. One common need for developers when trying to find the root cause is analyzing stacktrace when exceptions are occurring. Let us see how to include this information in your telemetry data.

Open the App.xaml.cs file of your Mobile App. In the method OnLaunched, add the following code (just before the previous code added in Task 1):

protected override async void OnLaunched(LaunchActivatedEventArgs e)

{

//telemetryClient.TrackTrace("App is launched");

var tc = new TelemetryClient();

***// Load file from harddive***

***try***

***{***

***var offLineDb = File.ReadAllText(@"c:\productsDB.xml");***

***}***

***catch (IOException ioe)***

***{***

***tc.TrackException(ioe);***

***tc.TrackTrace("An exception occured while loading backup database products", SeverityLevel.Critical);***

***}***

Include the namespace at the top section of the file with the other using statements:

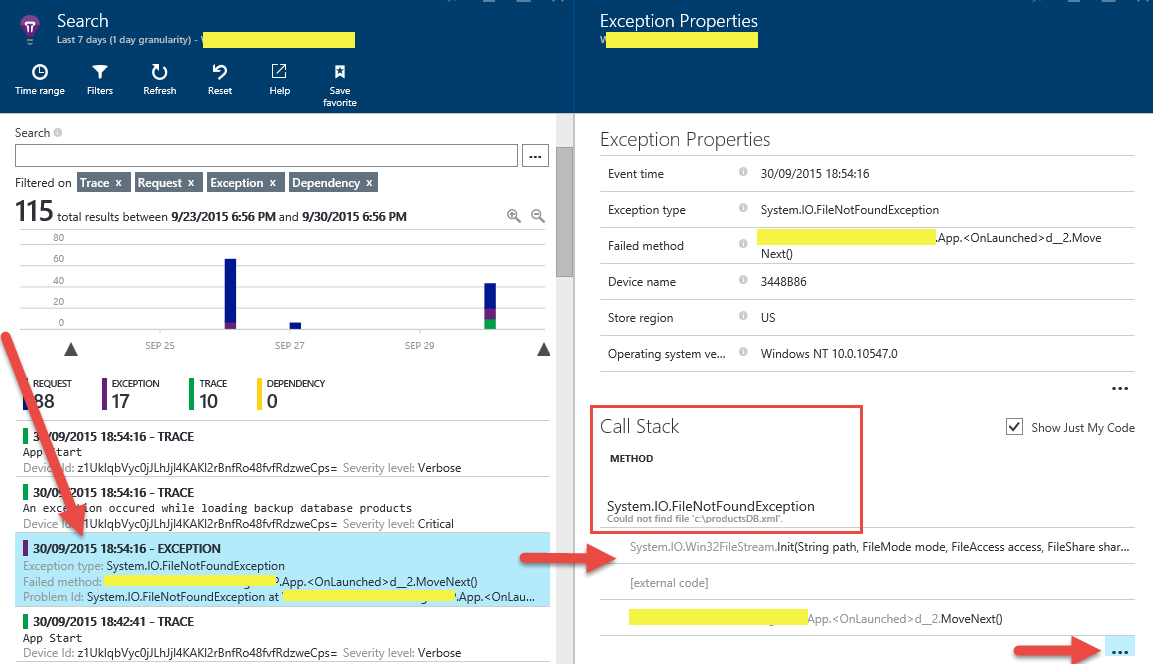
using Microsoft.ApplicationInsights;

using Microsoft.ApplicationInsights.DataContracts;

Go to the portal in your AI Instance.

Click the **Search** button.

Locate the Exception entry. Click and check the details of the page. Take some time to view the **Call Stack** section and click the ellipses to have the complete detailed view.



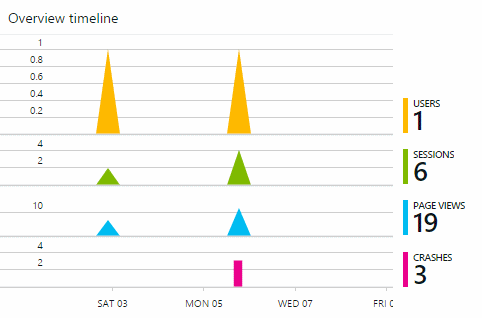
## Exercise 4: Detect and diagnose crashes

**Crash** means that the application terminates because of an uncaught exception. If your app catches an exception, you can report it with the TrackException API but continue running. In that case, it will not be logged as a crash.

Task 1: Monitor Crashes

Go to Application Insights dashboard ([the Application Insights portal](http://portal.azure.com/).)

Go to the default graph. Crashes appear on your application's overview blade at the bottom.

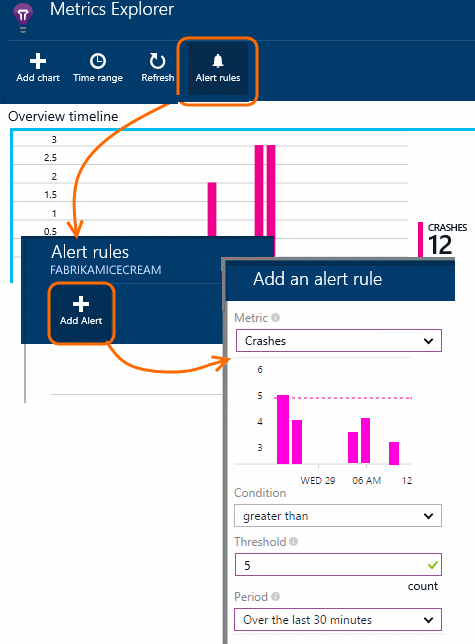


**Note** You can edit the time range shown by the chart.

Task 2: Set an Alert to Detect Crashes

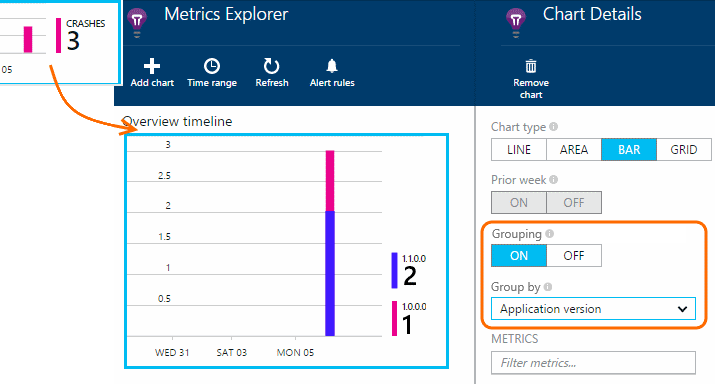
On the menu, click **Metric Explorer**.

On the menu, select **Alert rules**.

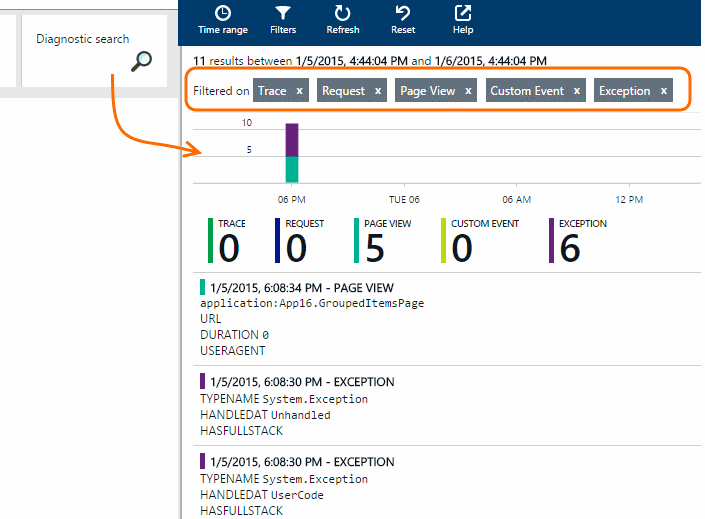


Task 3: Diagnose Crashes

To find whether some versions of your app crash more than others, click through the crashes chart, and then segment by Application Version:



To discover the exceptions that are causing crashes, open **Diagnostic Search**. You might want to remove other types of telemetry to focus on the exceptions.



Click any exception to see its details, including associated properties and stack trace.

