Adobe® Experience Cloud
Universal Windows Platform SDK 4.x for Experience Cloud Solutions
## Contents

**Universal Windows Platform SDK 4.x for Experience Cloud Solutions** ............................................. 4

**Release Notes** ........................................................................................................................................ 5

**Getting Started** ..................................................................................................................................... 6
  Developer Quick Start................................................................................................................................. 6

**Configuration** ........................................................................................................................................ 9
  ADBMobileConfig.json Config.................................................................................................................. 9
  SDK Methods.......................................................................................................................................... 11

**Analytics** ............................................................................................................................................... 14
  Products Variable................................................................................................................................... 19
    Products Variable with Merchandising eVars and Product-Specific Events............................................. 20
  Event Serialization................................................................................................................................... 20
  Video Analytics....................................................................................................................................... 21
  Analytics Methods................................................................................................................................. 25

**Target** ................................................................................................................................................... 29
  Target Methods................................................................................................................................... 29

**Audience Manager** ............................................................................................................................... 32
  Audience Manager Methods................................................................................................................... 32

**Lifecycle Metrics** ................................................................................................................................. 34

**Extensions** .......................................................................................................................................... 38
  Windows Visual Studio Extensions for Experience Cloud Solutions 4.x SDK........................................ 38

**Using Bloodhound to Test Mobile Applications** ............................................................................... 40
Universal Windows Platform SDK 4.x for Experience Cloud Solutions

Universal Windows Platform SDK 4.x for Experience Cloud Solutions lets you measure native Windows 10 applications, deliver targeted content within your app, and leverage and collect audience data through audience management.

Last Updated: September 17, 2015

Supported Platforms
Windows 10 or later.

Supported IDE
Visual Studio 2015 or later.

Adobe Mobile Services
Adobe Mobile services provides a new UI that brings together mobile marketing capabilities for mobile applications from across the Adobe Experience Cloud. Initially, the Mobile service provides seamless integration of app analytics and targeting capabilities from the Adobe Analytics and Adobe Target solutions.

Learn more at Adobe Mobile services documentation.
Release Notes


Current Release Notes
In addition to significant performance increases, version 4.x adds the following new features:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Universal Windows Platform SDK 4.x for Experience Cloud Solutions</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>

For more information about the release notes for all solutions, see Adobe Experience Cloud Release Notes.
Getting Started

Information to help you get starting with the Universal Windows Platform SDK for Experience Cloud Solutions

Complete the tasks in Developer Quick Start to add the Universal Windows Platform SDK to your project. Next, review the SDK Methods and then implement Analytics, Target, and Audience Manager.

If you are upgrading from a previous version, you'll want to take a look at the 4.x Migration Guide.

Developer Quick Start

This guide walks you through the steps to implement the Universal Windows Platform library.

- Get the SDK
- Add the Library and Config File to your Project - C#
- Add the Library and Config File to your Project - C++
- Add the Library and Config File to your Project - WinJS
- Update The ADBMobileConfig.json Config File
- Optional - Install and use the Visual Studio Extensions for ADBMobile on Windows Universal App Store apps

You'll need Visual Studio 2013 or later to implement the SDK.

Get the SDK

After you unzip the SDK download, you'll have a separate folder for each supported architecture and platform combination. You will also have an ADBMobileConfig.json file that is explained later in this guide.

Select the Correct Version

Different .dll/.winmd files are provided for each supported architecture (x86, x64, ARM).

Note: The version of ADBMobile.winmd does not reflect the version of the library. The .winmd file contains metadata only, and as such will have a version number of 255.255.255.255 which is accepted behavior according to Microsoft (see How do I add assembly information for a WinRT C++ / CX component dll?). In order to check the version of the library you are using, you must check the version of the underlying ADBMobile.dll file.

Syntax Differences

The Universal Windows Platform library can be used in several programming languages. The examples in this guide are in WinJS (JavaScript), and might need to be modified if you are using a different language. Note that when you consume winmd methods from winJS (JavaScript), all methods automatically have their first letter lowercased.

The main difference between the implementations is the data structure used for context data.

Additionally, when using the SDK in a WinJS project, use an empty string ("" or '') instead of null for empty string values.

Add the Library and Config File to your Project - C#

1. Launch Visual Studio and open your solution.
2. In the Solution Explorer, right-click References and select Add Reference.
3. Select the Correct Version of the library and then browse to the associated ADBMobile.winmd file. Click Add.
4. Verify that ADBMobile.winmd is checked in the Reference Manager window and click OK.
5. In the Solution Explorer, right-click References and select Add Reference.
6. (Skip this step if you also have a C++ project in your solution) In the Windows tab on the left, select Extensions, then select Visual C++ 2015 Runtime for Universal Windows Platform Apps.

7. Add the following line to your class:
   ```
   using ADBMobile;
   ```

8. Right-click your project and select Add > Existing Item.
10. Right-click ADBMobileConfig.json in your solution and select Properties.

Add the Library and Config File to your Project - C++

1. Launch Visual Studio and open your solution.
2. In the Solution Explorer, right-click your project and select Add > References.
3. Select the Correct Version of the library and then add a reference to the associated ADBMobile.winmd file. Click Add.
4. Verify that ADBMobile.winmd is checked in the Reference Manager window and click OK.
5. Add the following line to your class:
   ```
   using namespace ADBMobile;
   ```

6. Right-click your project and select Add > Existing Item.
7. Browse to ADBMobileConfig.json and click Add.
8. Right-click ADBMobileConfig.json in your solution and select Properties.
9. On the General tab, change Content to Yes and click OK.

Add the Library and Config File to your Project - WinJS

1. Launch Visual Studio and open your solution.
2. In the Solution Explorer, right-click References and select Add Reference.
3. Select the Correct Version of the library and then browse to the associated ADBMobile.winmd file. Click Add.
4. Verify that ADBMobile.winmd is checked in the Reference Manager window and click OK.
5. In the Solution Explorer, right-click References and select Add Reference.
6. (Skip this step if you also have a C++ project in your solution) In the Windows tab on the left, select Extensions, then select Visual C++ 2015 Runtime for Universal Windows Platform Apps.
7. Right-click your project and select Add > Existing Item.
8. Browse to ADBMobileConfig.json and click Add.
9. Right-click ADBMobileConfig.json in your solution and select Properties.
10. With File Properties selected, ensure Package Action is set to Content (for JavaScript projects, the file is set to Content by default).

Update The ADBMobileConfig.json Config File

The ADBMobileConfig.json file contains global SDK settings, and is located at your project root after you complete the Add the Library and Config File to your Project steps in the previous section. If your ADBMobileConfig.json file was not pre-configured by Adobe mobile services, then you need to update a few values to get started.

The following is an example of an ADBMobileConfig.json file:

```json
{
   "version" : "1.0",
   "analytics" : {
      "rsids" : "coolApp",
      "server" : "my.CoolApp.com",
      "charset" : "UTF-8",
      "ssl" : false,
      "offlineEnabled" : true,
      "lifecycleTimeout" : 300,
   }
}
```
At a minimum, update the following values for the Solutions you are using:

- **Analytics**: rsids and server
- **Target**: clientCode
- **Audience Management**: server

For more details, see .

**Debugging**

When you want to enable debugging for the SDK, you have to call `ADBMobile.Config.setDebugLogging(true);`.

For C# and JS apps, you have to enable native code debugging by completing the following steps (native code debugging is the default setting for C++ apps):

**C#**

- Right-click project, select **Properties > Debug tab**. Change the debugger type drop down to **Native Only**.

**JS**

- Right-click project, select **Properties > Configuration Properties > Debug tab**. Change the debugger type drop down to **Native Only**.

That’s it! You’re now ready to implement Analytics, Target, and Audience Management in your Universal Windows Platform app.

Where to go from here:

- **SDK Methods**
- **Analytics**
- **Target**
- **Audience Manager**
# Configuration

Information to help you configure the Universal Windows Platform SDK, including JSON configuration, hit batching, and SDK methods.

## ADBMobileConfig.json Config

Information to help you use the ADBMobile JSON Config file.

The SDK currently has support for multiple Adobe Experience Cloud Solutions, including Analytics, Target, and Audience Manager. Methods are prefixed according to the solution. Configuration methods are prefixed with "Config."

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsids</td>
<td>(Required by Analytics) One or more report suites to receive Analytics data. Multiple report suite IDs should be comma-separated with no space between.</td>
</tr>
<tr>
<td>server</td>
<td>(Required by Analytics and Audience Management). Analytics or Audience Management server, based on the parent node.</td>
</tr>
<tr>
<td></td>
<td>This variable should be populated with the server domain, without an &quot;http://&quot; or https://&quot; protocol prefix. The protocol prefix is handled automatically by the library based on the ssl variable.</td>
</tr>
<tr>
<td></td>
<td>If ssl is true, a secure connection is made to this server. If ssl is false, a non-secure connection is made to this server.</td>
</tr>
<tr>
<td>charset</td>
<td>Defines the character set you are using for the data sent to Analytics. The charset is used to convert incoming data into UTF-8 for storage and reporting. See Using the charSet Property.</td>
</tr>
<tr>
<td>ssl</td>
<td>Default: false</td>
</tr>
<tr>
<td></td>
<td>Enables (true) or disables (false) sending measurement data via SSL (HTTPS).</td>
</tr>
<tr>
<td>offlineEnabled</td>
<td>Default: false</td>
</tr>
<tr>
<td></td>
<td>When enabled (true), hits are queued while the device is offline and sent later when the device is online. Your report suite must be timestamp-enabled to use offline tracking.</td>
</tr>
<tr>
<td></td>
<td><strong>Important</strong>: If time stamps are enabled on your report suite, your offlineEnabled configuration property must be true. If your report suite is not timestamp enabled, your offlineEnabled configuration property must be false. If this is not configured correctly, data will be lost. If you are not sure if a report suite is timestamp enabled, contact Customer Care.</td>
</tr>
<tr>
<td></td>
<td>If you are currently reporting AppMeasurement data to a report suite that also collects data from JavaScript, you might need to set up a separate report suite for mobile data, or include a custom timestamp on all JavaScript hits using the s.timestamp variable.</td>
</tr>
<tr>
<td>lifecycleTimeout</td>
<td>Default: 300 seconds</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Specifies the length of time, in seconds, that must elapse between app launches before the launch is considered a new session. This timeout also applies when your application is sent to the background and reactivated. The time that your app spends in the background is not included in the session length.</td>
</tr>
<tr>
<td>batchLimit</td>
<td>Default: 0 (No batching) Send hits in batches. For example, if set to 50, hits are queued until 50 are stored, then all queued hits are sent. Requires offlineEnabled=true.</td>
</tr>
</tbody>
</table>
| privacyDefault| Default: optedin  
- optedin - hits are sent immediately.  
- optedout - hits are discarded.  
- optunknown - If your report suite is timestamp-enabled, hits are saved until the privacy status changes to opt-in (then hits are sent) or opt-out (then hits are discarded). If your report suite is not timestamp-enabled, hits are discarded until the privacy status changes to opt in. |
| poi           | Each POI array holds the POI name, latitude, longitude, and radius (in meters) for the area of the point. The POI name can be any string.  
When a trackLocation call is sent, if the current coordinates are within a defined POI, a context data variable is populated and sent with the trackLocation call. |
|              | "poi" : [  
["san francisco",37.757144,-122.44812,7000],  
["santa cruz",36.972935,-122.01725,600]  
] |
| clientCode    | (Required by Target) Your assigned client code. |
| timeout       | Determines how long target waits for a response. |

The following is an example of an ADBMobileConfig.json file:

```json
{
  "version" : "1.0",
  "analytics" : {
    "rsids" : "coolApp",
    "server" : "my.CoolApp.com",
    "charset" : "UTF-8",
    "ssl" : false,
    "offlineEnabled" : true,
    "lifecycleTimeout" : 5,
    "privacyDefault" : "optedin",
    "poi" : [  
      ["san francisco",37.757144,-122.44812,7000],  
      ["santa cruz",36.972935,-122.01725,600]  
    ],
    "target" : {  
      "clientCode" : "myTargetClientCode",
      "timeout" : 1
    }
  }
}
```
SDK Methods

Classes and methods provided by the Universal Windows Platform library.

**Note:** When you consume winmd methods from winJS (JavaScript), all methods automatically have their first letter lowercased.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetVersion</td>
<td>Returns the current version of the Adobe Mobile library.</td>
</tr>
<tr>
<td>winJS: getVersion</td>
<td>Syntax:</td>
</tr>
<tr>
<td></td>
<td>static Platform::String ^GetVersion();</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>var ADB = ADBMobile;</td>
</tr>
<tr>
<td></td>
<td>var libVersion = ADB.Config.getVersion();</td>
</tr>
<tr>
<td>GetPrivacyStatusAsync</td>
<td>Returns the enum representation of the privacy status for current user.</td>
</tr>
<tr>
<td>winJS: getPrivacyStatusAsync</td>
<td>• ADBMobilePrivacyStatusOptIn - hits are sent immediately.</td>
</tr>
<tr>
<td></td>
<td>• ADBMobilePrivacyStatusOptOut - hits are discarded.</td>
</tr>
<tr>
<td></td>
<td>• ADBMobilePrivacyStatusUnknown - If your report suite is timestamp-enabled, hits are saved until the privacy status changes to opt-in (then hits are sent) or opt-out (then hits are discarded). If your report suite is not timestamp-enabled, hits are discarded until the privacy status changes to opt-in.</td>
</tr>
<tr>
<td></td>
<td>Default: The default value is set in ADBMobileConfig.json</td>
</tr>
<tr>
<td></td>
<td>Syntax:</td>
</tr>
<tr>
<td></td>
<td>static Windows::Foundation::IAsyncOperation&lt;ADBMobilePrivacyStatus&gt; ^getPrivacyStatusAsync();</td>
</tr>
<tr>
<td></td>
<td>public enum class ADBMobilePrivacyStatus : int {</td>
</tr>
<tr>
<td></td>
<td>ADBMobilePrivacyStatusOptIn = 1,</td>
</tr>
<tr>
<td></td>
<td>ADBMobilePrivacyStatusOptOut = 2,</td>
</tr>
<tr>
<td></td>
<td>ADBMobilePrivacyStatusUnknown = 3</td>
</tr>
<tr>
<td></td>
<td>});</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>var ADB = ADBMobile;</td>
</tr>
<tr>
<td></td>
<td>var status;</td>
</tr>
<tr>
<td></td>
<td>ADB.Config.getPrivacyStatusAsync.then(function(privacyStatus) {</td>
</tr>
<tr>
<td></td>
<td>status = privacyStatus;</td>
</tr>
<tr>
<td></td>
<td>});</td>
</tr>
<tr>
<td>SetPrivacyStatus</td>
<td>Sets the privacy status for the current user to status. Set to one of the following values:</td>
</tr>
<tr>
<td>winJS: setPrivacyStatus</td>
<td>• ADBMobilePrivacyStatusOptIn - hits are sent immediately.</td>
</tr>
<tr>
<td></td>
<td>• ADBMobilePrivacyStatusOptOut - hits are discarded.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ADBMobilePrivacyStatusUnknown</td>
<td>- If your report suite is timestamp-enabled, hits are saved until the privacy status changes to opt-in (then hits are sent) or opt-out (then hits are discarded). If your report suite is not timestamp-enabled, hits are discarded until the privacy status changes to opt in.</td>
</tr>
</tbody>
</table>

Syntax:
```csharp
static void SetPrivacyStatus(ADBMobilePrivacyStatus status);
```

```csharp
public enum class ADBMobilePrivacyStatus : int {
    ADBMobilePrivacyStatusOptIn = 1,
    ADBMobilePrivacyStatusOptOut = 2,
    ADBMobilePrivacyStatusUnknown = 3
};
```

Example:
```csharp
var ADB = ADBMobile;
ADB.Config.setPrivacyStatus(ADB.ADBMobilePrivacyStatus.adbmobilePrivacyStatusOptIn);
```

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetLifetimeValue</td>
<td>Returns the lifetime value of the current user.</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
<tr>
<td></td>
<td>Syntax:</td>
</tr>
</tbody>
</table>
|                     | ```csharp
static float GetLifetimeValue();
``` |

Example:
```csharp
var ADB = ADBMobile;
var ltv = ADB.Config.getLifetimeValue();
```

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetUserIdentifier</td>
<td>Returns the custom user identifier if a custom identifier has been set. Returns null if a custom identifier is not set.</td>
</tr>
<tr>
<td></td>
<td>Default: null</td>
</tr>
<tr>
<td></td>
<td>Note: If your app upgrades from the Experience Cloud 3.x to 4.x SDK, the previous ID service (either custom or automatically generated) is retrieved and stored as the custom user identifier. This preserves visitor data between upgrades of the SDK. For new installations on the 4.x SDK, user identifier is null until set.</td>
</tr>
<tr>
<td></td>
<td>Syntax:</td>
</tr>
</tbody>
</table>
|                     | ```csharp
static Platform::String ^GetUserIdentifier();
``` |

Example:
```csharp
var ADB = ADBMobile;
var userId = ADB.Config.getUserIdentifier();
```

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetUserIdentifier</td>
<td>Sets the user identifier to <code>identifier</code>.</td>
</tr>
<tr>
<td></td>
<td>Syntax:</td>
</tr>
</tbody>
</table>
|                     | ```csharp
static void SetUserIdentifier(Platform::String ^userIdentifier);
``` |

Example:
```csharp
var ADB = ADBMobile;
ADB.Config.setUserIdentifier("someUserId");
```
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetDebugLogging</td>
<td>Returns the current debug logging preference. Default: false</td>
</tr>
<tr>
<td>Syntax:</td>
<td>static bool GetDebugLogging();</td>
</tr>
<tr>
<td>Example:</td>
<td>var ADB = ADBMobile; var logging = ADB.Config.getDebugLogging();</td>
</tr>
<tr>
<td>SetDebugLogging</td>
<td>Sets the debug logging preference to debugLogging. Debug logging works only when using the debug version of the library, the release version ignores this setting.</td>
</tr>
<tr>
<td>Syntax:</td>
<td>static void SetDebugLogging(bool debugLogging);</td>
</tr>
<tr>
<td>Example:</td>
<td>var ADB = ADBMobile; ADB.Config.setDebugLogging(true);</td>
</tr>
<tr>
<td>CollectLifecycleData</td>
<td>Indicates to the SDK that lifecycle data should be collected for use across all solutions in the SDK. See Lifecycle Metrics.</td>
</tr>
<tr>
<td>Syntax:</td>
<td>static void CollectLifecycleData();</td>
</tr>
<tr>
<td>Example:</td>
<td>var ADB = ADBMobile; ADB.Config.collectLifecycleData();</td>
</tr>
<tr>
<td>PauseCollectingLifecycleData</td>
<td>Indicates to the SDK that your app is paused, so that lifecycle metrics are calculated correctly. For example, on pause collects a timestamp to determine previous session length. This also sets a flag so that lifecycle correctly knows that the app did not crash. See Lifecycle Metrics.</td>
</tr>
<tr>
<td>Syntax:</td>
<td>static void PauseCollectingLifecycleData();</td>
</tr>
<tr>
<td>Example:</td>
<td>var ADB = ADBMobile; ADB.Config.pauseCollectingLifecycleData();</td>
</tr>
</tbody>
</table>
Analytics

After you add the library to your project, you can make any of the Analytics method calls anywhere in your App (make sure you import ADBMobile.h to your class).

Enable Mobile Application Reports in Analytics

Before you add code, have your Analytics Administrator complete the following to enable Mobile App Lifecycle tracking. This ensures that your report suite is ready to capture metrics as you begin development.

1. Open Admin Tools > Report Suites and select your mobile report suite(s).
2. Click Edit Settings > Mobile Management > Mobile Application Reporting.

3. Click Enable Latest App Reports.

Optionally, you can also click Enable Mobile Location Tracking or Enable Legacy Reporting and Attribution for background hits.
Lifecycle metrics are now ready to be captured, and **Mobile Application Reports** appear in the **Reports** menu in the marketing reports interface.

**New Versions**

Periodically, new versions of mobile application reporting are released. New versions are not applied to your report suite automatically, you must repeat these steps to perform the upgrade. Each time you add new Experience Cloud functionality to your app, we recommend repeating these steps to ensure you have the latest configuration.

**Lifecycle Metrics**

To collect lifecycle metrics in your app, add calls to when the application is activated as shown in the following examples.

**WinJS, in default.js:**

```javascript
app.onactivated = function (args) {
  if (args.detail.kind === activation.ActivationKind.launch) {
    ...
    // launched and resumed stuff
    ADBMobile.Config.collectLifecycleData();
  }
```

---

---
If `CollectLifecycleData()` is called twice in the same session, then your application will report a crash on every call after the first. The SDK sets a flag when the application is shutdown that indicates a successful exit. If this flag is not set, `CollectLifecycleData()` reports a crash.
Events, Props, and eVars

If you've looked at the SDK Methods, you are probably wondering where to set events, eVars, props, heirs, and lists. In version 4, you can no longer assign those types of variables directly in your app. Instead, the SDK uses context data and processing rules to map your app data to Analytics variables for reporting.

Processing rules provide you several advantages:

- You can change your data mapping without submitting an update to the App Store.
- You can use meaningful names for data instead of setting variables that are specific to a report suite.
- There is little impact to sending in extra data. These values won’t appear in reports until they are mapped using processing rules.

Any values that you were assigning directly to variables should be added to context data instead.

Processing Rules

Processing rules are used to copy the data you send in context data variables to evars, props, and other variables for reporting.

Processing Rules Training @ Summit 2013

Processing Rules Help

Become authorized to use Processing Rules

We recommend grouping your context data variables using "namespaces", as it helps you keep logical ordering. For example, if you want to collect info about a product, you might define the following variables:

```
"product.type":"hat"
"product.team":"mariners"
"product.color":"blue"
```

Context data variables are sorted alphabetically in the processing rules interface, so namespaces let you quickly see variables that are in the same namespace.

Also, we have heard that some of you are naming context data keys using the evar or prop number:

```
"eVar1":"jimbo"
```

This might make it slightly easier when you perform the one time mapping in processing rules, but you lose readability during debugging and future code updates can be more difficult. Instead, we strongly recommend using descriptive names for keys and values:

```
"username":"jimbo"
```

Set context variables that define counter events to a value of "1":

```
"logon":"1"
```

Context data variables that define incrementor events can have the value to increment:

```
"levels completed":"6"
```

Note: Adobe reserves the namespace ".". Aside from that small restriction, context data variables just need to be unique in your login company to avoid collisions.

Products Variable

To set products in the mobile SDK, you must use a special syntax. See Products Variable.
(Optional) Enable Offline Tracking

To store hits when the device is offline, you can enable offline tracking in the . Pay very close attention to the timestamp requirements described in the config file reference before you enable offline tracking.

Geo-location & Points of Interest

Geo-location lets you measure location data (latitude/longitude) and pre-defined points of interest. Each TrackLocation call sends:

- Latitude/Longitude, and POI (if within a POI defined in the ADBMobileConfig.json config file). These are passed to mobile solution variables for automatic reporting.
- Distance from center & accuracy passed as context data. Capture using a processing rule.

To track a location:

```javascript
var ADB = ADBMobile;
ADB.Analytics.trackLocation(37.75345, -122.33207, null);
```

If the following POI is defined in the ADBMobileConfig.json config file:

```json
"poi" : [
  ["San Francisco",37.757144,-122.44812,7000],
]
```

When the device location is determined to be within a 7000 meter radius of the defined point, an `a.loc.poi` context data variable with the value "San Francisco" is sent in with the TrackLocation hit. An `a.loc.dist` context variable is sent with the distance in meters from the defined coordinates.

Lifetime Value

Lifetime value lets you measure and target on a lifetime value for each user. Each time you send in a value with TrackLifetimeValueIncrease, the value is added to the existing value. Lifetime value is stored on device and can be retrieved at any time by calling GetLifetimeValue. This can be used to store lifetime purchases, ad views, video completes, social shares, photo uploads, and so on.

```javascript
// Lifetime Value Example
var ADB = ADBMobile;
var purchasePrice = 39.95;
var cdata = new Windows.Foundation.Collections.PropertySet();
cdata["ItemPurchaseEvent"] = "ItemPurchaseEvent";
cdata["PurchaseItem"] = "Item453";
cdata["PurchasePrice"] = purchasePrice;
ADB.Analytics.trackLifetimeValueIncrease(purchasePrice, cdata);
```

Timed Actions

Timed actions let you measure in-app time and total time between the start and end of an action. The SDK calculates the amount of time in session and the total time (cross-session) it takes for the action to be completed. This can be used to define segments to compare on time to purchase, pass level, checkout flow, and so on.

- Total # of seconds in app between start and end - cross sessions
- Total # of seconds between start and end (clock time)

```javascript
// Timed Action Start Example
var ADB = ADBMobile;
var experience = "ExperienceName";
ADB.Analytics.trackTimedActionStart("TimeUntilPurchase", cdata);

// Timed Action Update Example
var ADB = ADBMobile;
var cdataUpdate = new Windows.Foundation.Collections.PropertySet();
```
cdataUpdate["ImageLiked"] = imageName;
ADB.Analytics.trackTimedActionStart("TimeUntilPurchase", cdata);

// Timed Action End Example
var ADB = ADBMobile;
ADB.Analytics.trackTimedActionEnd("TimeUntilPurchase");

Products Variable

The *products* variable cannot be set using processing rules. In the mobile SDK, you must use a special syntax within the context data parameter to set *products* directly on the server call.

To set the *products* variable, set a context data key to "&&products", and set the value using the syntax defined for the *products* variable:

```javascript
cdata["&&products"] = "Category;Product;Quantity;Price[,Category;Product;Quantity;Price]";
```

For example:

```javascript
//create a context data dictionary
var cdata = new Windows.Foundation.Collections.PropertySet();

// add products, a purchase id, a purchase context data key, and any other data you want to collect.
// Note the special syntax for products
Cdata["&&products"] = ";Running Shoes;1;69.95;Running Socks;10;29.99";
cdata["m.purchaseid"] = "1234567890";
cdata["m.purchase"] = "1";

var ADB = ADBMobile;
// send the tracking call - use either a trackAction or TrackState call.
// trackAction example:
ADB.Analytics.trackAction("purchase", cdata);
// trackState example:
ADB.Analytics.trackState("Order Confirmation", cdata);
```

Note that *products* is set directly on the image request, and the other variables are set as context data:

All context data variables must be mapped using processing rules:
You do not need to map the `products` variable using processing rules since it is set directly on the image request by the SDK.

**Products Variable with Merchandising eVars and Product-Specific Events**

An example of the `products` variable with Merchandising eVars and product-specific events.

```javascript
// create a context data dictionary
var cdata = new Windows.Foundation.Collections.PropertySet();

// add products, a purchase id, a purchase context data key, and any other data you want to collect.
// Note the special syntax for products
var event1 = "Running Shoes;1;69.95;event1=5.5;eVar1=Merchandising,;Running Socks;10;29.99";
cdata["m.purchaseid"] = "1234567890";
cdata["m.purchase"] = "1";

var ADB = ADBMobile;
// send the tracking call - use either a trackAction or TrackState call.
// trackAction example:
ADB.Analytics.trackAction("purchase", cdata);
// trackState example:
ADB.Analytics.trackState("Order Confirmation", cdata);
```

💡 **Note:** If you trigger a product-specific event using the `&products` variable, you must also set that event in the `&events` variable, otherwise the event is filtered out during processing.

**Event Serialization**

Event serialization is not supported by processing rules. In the mobile SDK, you must use a special syntax within the context data parameter to set serialized events directly on the server call.

See [Event Serialization].

```javascript
cdata["&events"] = "event1:12341234";
```
For example:

```javascript
// create a context data dictionary
var cdata = new Windows.Foundation.Collections.PropertySet();

// add events
cdata["&events"] = "event1:12341234";

var ADB = ADBMobile;
// send the tracking call - use either a trackAction or TrackState call.
// trackAction example:
ADB.Analytics.trackAction("action", cdata);
// trackState example:
ADB.Analytics.trackState("State Name", cdata);
```

**Video Analytics**

Information to help you with Video Analytics.

Video measurement is described in detail in the *Measuring Video in Analytics* guide. The general process to measure video is very similar across all AppMeasurement platforms. This quick start section provides a basic overview of the developer tasks along with code samples.

**Map Player Events to Analytics Variables**

The following table lists the media data that is sent to Analytics. Use processing rules to map the context data in the Context Data Variable column to an Analytics variable as described in the Variable Type column.

<table>
<thead>
<tr>
<th>Context Data Variable</th>
<th>Variable Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.media.name</td>
<td>eVar</td>
<td>(Required) Collects the name of the video, as specified in the implementation, when a visitor views the video in some way. You can add classifications for this variable. Default expiration: Visit Custom Insight (s.prop, used for video pathing)</td>
</tr>
<tr>
<td>a.media.name</td>
<td>Custom Insight (s.prop)</td>
<td>(Optional) Provides video pathing information. Pathing must be enabled for this variable by ClientCare. Event type: Custom Insight (s.prop)</td>
</tr>
<tr>
<td>a.media.segment</td>
<td>eVar</td>
<td>(Required) Collects video segment data, including the segment name and the order in which the segment occurs in the video. This variable is populated by enabling the segmentByMilestones variable when tracking player events automatically, or by setting a custom segment name when tracking player events manually. Default expiration: Page view</td>
</tr>
</tbody>
</table>

For example, when a visitor views the first segment in a video, SiteCatalyst might collect the following in the Segments eVar:

```
1:M:0-25
```
<table>
<thead>
<tr>
<th>Context Data Variable</th>
<th>Variable Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>The default video data collection method collects data at the following points: video start (play), segment begin, and video end (stop). Analytics counts the first segment view at the start of the segment, when the visitor starts watching. Subsequent segment views as the segment begins.</td>
</tr>
<tr>
<td>a.contentType</td>
<td>eVar</td>
<td>Collects data about the type of content viewed by a visitor. Hits sent by video measurement are assigned a content type of &quot;video&quot;. This variable does not need to be reserved exclusively for video tracking. Having other content report content type using this same variable lets you analyze the distribution of visitors across the different types of content. For example, you could tag other content types using values such as &quot;article&quot; or &quot;product page&quot; using this variable. From a video measurement perspective, Content Type lets you identify video visitors and thereby calculate video conversion rates.</td>
</tr>
<tr>
<td>a.media.timePlayed</td>
<td>Event</td>
<td>Counts the time, in seconds, spent watching a video since the last data collection process (image request).</td>
</tr>
<tr>
<td>a.media.view</td>
<td>Event</td>
<td>Indicates that a visitor has viewed some portion of a video. However, it does not provide any information about how much, or what part, of a video the visitor viewed.</td>
</tr>
<tr>
<td>a.media.segmentView</td>
<td>Event</td>
<td>Indicates that a visitor has viewed some portion of a video segment. However, it does not provide any information about how much, or what part, of a video the visitor viewed.</td>
</tr>
<tr>
<td>a.media.complete</td>
<td>Event</td>
<td>Indicates that a user has viewed a complete video. By default, the complete event is measured 1 second before the end of the video. During implementation, you can specify how many seconds from the end of the video you would like to consider a view complete. For live video and other streams that don’t have a defined end, you can specify a custom point to measure completes. For example, after a specific time viewed.</td>
</tr>
</tbody>
</table>

Configure Media Settings

Configure a MediaSettings object with the settings you want to use to track video:

```javascript
var mySettings = ADB.Media.settingsWith("name", 10, "playerName", "playerId");
```
Track Player Events

To measure video playback, the Play, Stop, and Close methods need to be called at the appropriate times. For example, when the player is paused, Stop, Play is called when playback starts or is resumed.

Classes

Class: MediaSettings

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>property Platform::String ^name;</td>
<td>Returns a MediaSettings object with specified parameters.</td>
</tr>
<tr>
<td>property Platform::String ^playerName;</td>
<td></td>
</tr>
<tr>
<td>property Platform::String ^playerID;</td>
<td></td>
</tr>
<tr>
<td>property double length;</td>
<td></td>
</tr>
<tr>
<td>property Platform::String ^channel;</td>
<td></td>
</tr>
<tr>
<td>property Platform::String ^milestones;</td>
<td></td>
</tr>
<tr>
<td>property Platform::String ^offsetMilestones;</td>
<td></td>
</tr>
<tr>
<td>property bool segmentByMilestones;</td>
<td></td>
</tr>
<tr>
<td>property bool segmentByOffsetMilestones;</td>
<td></td>
</tr>
<tr>
<td>property int trackSeconds;</td>
<td></td>
</tr>
<tr>
<td>property int completeCloseOffsetThreshold;</td>
<td></td>
</tr>
</tbody>
</table>

// MediaAnalytics Ad settings

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>property Platform::String ^parentName;</td>
<td>Returns a MediaSettings object for use with tracking an ad video.</td>
</tr>
<tr>
<td>property Platform::String ^parentPod;</td>
<td></td>
</tr>
<tr>
<td>property Platform::String ^CPM;</td>
<td></td>
</tr>
<tr>
<td>property double parentPodPosition;</td>
<td></td>
</tr>
<tr>
<td>property bool isMediaAd;</td>
<td></td>
</tr>
</tbody>
</table>

Media Measurement Class and Method Reference

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SettingsWith winJS: settingsWith</td>
<td>Returns a MediaSettings object with specified parameters.</td>
</tr>
<tr>
<td>Syntax:</td>
<td>static MediaSettings ^SettingsWith(Platform::String ^name, double length, Platform::String ^playerName, Platform::String ^playerID);</td>
</tr>
<tr>
<td>Example:</td>
<td>var mySettings = ADB.Media.settingsWith(&quot;name&quot;, 10, &quot;playerName&quot;, &quot;playerId&quot;);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdSettingsWith winJS: adSettingsWith</td>
<td>Returns a MediaSettings object for use with tracking an ad video.</td>
</tr>
<tr>
<td>Syntax:</td>
<td>static MediaSettings ^AdSettingsWith(Platform::String ^name, double length, Platform::String ^playerName, Platform::String ^parentName, Platform::String ^parentPod, double parentPosition, Platform::String ^CPM);</td>
</tr>
<tr>
<td>Example:</td>
<td>var myAdSettings = ADB.Media.adSettingsWith(&quot;name&quot;, 10, &quot;playerName&quot;, &quot;parentNode&quot;, &quot;parentPod&quot;, 5, &quot;myCPM&quot;);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open winJS: open</td>
<td>Tracks a media open using the settings defined in settings</td>
</tr>
<tr>
<td>Syntax:</td>
<td>static void Open(MediaSettings ^settings);</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td></td>
</tr>
<tr>
<td>Close</td>
<td>Tracks a media close for the media item named <em>name</em>.</td>
</tr>
<tr>
<td>winJS: close</td>
<td></td>
</tr>
<tr>
<td>Syntax:</td>
<td>static void Close(Platform::String ^name);</td>
</tr>
<tr>
<td>Example:</td>
<td>ADB.Media.close(&quot;mediaName&quot;);</td>
</tr>
<tr>
<td>Play</td>
<td>Tracks a media play for the media item named <em>name</em> at the given <em>offset</em> (in seconds).</td>
</tr>
<tr>
<td>winJS: play</td>
<td></td>
</tr>
<tr>
<td>Syntax:</td>
<td>static void Play(Platform::String ^name, double offset);</td>
</tr>
<tr>
<td>Example:</td>
<td>ADB.Media.play(&quot;mediaName&quot;, 0);</td>
</tr>
<tr>
<td>Complete</td>
<td>Manually mark the media item as complete at the <em>offset</em> provided (in seconds).</td>
</tr>
<tr>
<td>winJS: complete</td>
<td></td>
</tr>
<tr>
<td>Syntax:</td>
<td>static void Complete(Platform::String ^name, double offset);</td>
</tr>
<tr>
<td>Example:</td>
<td>ADB.Media.complete(&quot;mediaName&quot;, 8);</td>
</tr>
<tr>
<td>Stop</td>
<td>Notifies the media module that the video has been stopped or paused at the given <em>offset</em>.</td>
</tr>
<tr>
<td>winJS: stop</td>
<td></td>
</tr>
<tr>
<td>Syntax:</td>
<td>static void Stop(Platform::String ^name, double offset);</td>
</tr>
<tr>
<td>Example:</td>
<td>ADB.Media.stop(&quot;mediaName&quot;, 4);</td>
</tr>
<tr>
<td>Click</td>
<td>Notifies the media module that the media item has been clicked.</td>
</tr>
<tr>
<td>winJS: click</td>
<td></td>
</tr>
<tr>
<td>Syntax:</td>
<td>static void Click(Platform::String ^name, double offset);</td>
</tr>
<tr>
<td>Example:</td>
<td>ADB.Media.click(&quot;mediaName&quot;, 3);</td>
</tr>
<tr>
<td>Track</td>
<td>Sends a track action call (no page view) for the current media state.</td>
</tr>
<tr>
<td>winJS: track</td>
<td></td>
</tr>
<tr>
<td>Syntax:</td>
<td>static void Track(Platform::String ^name, Windows::Foundation::Collections::IMap&lt;Platform::String^, Platform::Object^&gt; ^contextData);</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td>ADB.Media.track(&quot;mediaName&quot;, null);</td>
</tr>
</tbody>
</table>

**Analytics Methods**

Information to help you use the Universal Windows Platform SDK with Adobe Analytics.

The SDK currently has support for multiple Adobe Experience Cloud Solutions, including Analytics, Target, and Audience Manager. Methods are prefixed according to the solution. Analytics methods are prefixed with "Analytics."

Each of these methods is used to send data into your Adobe Analytics report suite.

💡 **Note:** When you consume winmd methods from winJS (JavaScript), all methods automatically have their first letter lowercased.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TrackState</strong></td>
<td>Tracks an app state with optional context data. States are the views that are available in your app, such as &quot;home dashboard&quot;, &quot;app settings&quot;, &quot;cart&quot;, and so on. These states are similar to pages on a website, and TrackState calls increment page views. If state is empty, it displays as &quot;app name app version (build)&quot; in reports. If you see this value in reports, make sure you are setting state in each TrackState call. <strong>Note:</strong> This is the only tracking call that increments page views. Syntax: static void TrackState(Platform::String ^state, Windows::Foundation::Collections::IMap&lt;Platform::String^, Platform::Object^&gt; ^contextData); <strong>Example:</strong> var ADB = ADBMobile; ADB.Analytics.trackState(&quot;loginScreen&quot;, null);</td>
</tr>
<tr>
<td><strong>TrackAction</strong></td>
<td>Tracks an action in your app. Actions are the things that happen in your app that you want to measure, such as &quot;logons&quot;, &quot;banner taps&quot;, &quot;feed subscriptions&quot;, and other metrics. Syntax: static void TrackAction(Platform::String ^action, Windows::Foundation::Collections::IMap&lt;Platform::String^, Platform::Object^&gt; ^contextData); <strong>Example:</strong> var ADB = ADBMobile; ADB.Analytics.trackAction(&quot;Button Click&quot;, null);</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>GetTrackingIdentifierAsync</td>
<td>Returns the automatically generated visitor ID for Analytics. This is an app-specific unique visitor ID that is generated on initial launch and then stored and used from that point forward. This ID is preserved between app upgrades, and is removed on uninstall.</td>
</tr>
<tr>
<td>Syntax:</td>
<td>static Windows::Foundation::IAsyncOperation<a href="">Platform::String^</a> GetTrackingIdentifierAsync();</td>
</tr>
<tr>
<td>Example:</td>
<td>var trackingIdentifier; ADBMobile.Analytics.getTrackingIdentifierAsync().then(function (trackingid) { trackingIdentifier = trackingid; });</td>
</tr>
<tr>
<td>TrackLocation</td>
<td>Sends the current x y coordinates. Also uses points of interest defined in the ADBMobileConfig.json file to determine if the location provided as a parameter is within any of your POI. If the current coordinates are within a defined POI, a context data variable is populated and sent with the trackLocation call.</td>
</tr>
<tr>
<td>Syntax:</td>
<td>static void TrackLocation(double lat, double lon, double accuracy, Windows::Foundation::Collections::IMap&lt;Platform::String^, Platform::Object^&gt; ^contextData);</td>
</tr>
<tr>
<td>Example:</td>
<td>var ADB = ADBMobile; ADB.Analytics.trackLocation(47.60621, -122.33207, null);</td>
</tr>
<tr>
<td>TrackLifetimeValueIncrease</td>
<td>Adds amount to the user's lifetime value.</td>
</tr>
<tr>
<td>Syntax:</td>
<td>static void TrackLifetimeValueIncrease(float amount, Windows::Foundation::Collections::IMap&lt;Platform::String^, Platform::Object^&gt; ^contextData);</td>
</tr>
<tr>
<td>Example:</td>
<td>var ADB = ADBMobile; ADB.Analytics.trackLifetimeValueIncrease(10, null);</td>
</tr>
<tr>
<td>TrackTimedActionStart</td>
<td>Start a timed action with name action.</td>
</tr>
<tr>
<td>Syntax:</td>
<td>static void TrackTimedActionStart(Platform::String ^action, Windows::Foundation::Collections::IMap&lt;Platform::String^, Platform::Object^&gt; ^contextData);</td>
</tr>
<tr>
<td>Note:</td>
<td>This call does not send a hit.</td>
</tr>
</tbody>
</table>

If you call this method for an action that has already started, the previous timed action is overwritten.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| TrackTimedActionUpdate | Pass in `contextData` to update the context data associated with the given action. The data passed in is appended to the existing data for the given action, and overwrites the data if the same key is already defined for `action`.  

**Note:** *This call does not send a hit.*  

**Syntax:**  
```csharp
static void TrackTimedActionUpdate(Platform::String ^action,  
Windows::Foundation::Collections::IMap<Platform::String^,  
Platform::Object^> ^contextData);
```

**Example:**  
```javascript
var ADB = ADBMobile;  
var contextData = new Windows.Foundation.Collections.PropertySet();  
contextData["quantity"] = 3;  
ADB.Analytics.trackTimedActionUpdate("cartToCheckout", contextData);
```

| TrackTimedActionExistsAsync | Returns true if the given timed action exists, and false if it does not.  

**Syntax:**  
```csharp
static Windows::Foundation::IAsyncOperation<bool>  
^TrackTimedActionExistsAsync(Platform::String ^action);
```

**Example:**  
```javascript
ADBMobile.Analytics.trackTimedActionExistsAsync("signUp").then(function  
(exists) {  
actionExists = exists;  
});
```

| TrackTimedActionEnd | End a timed action.  

**Syntax:**  
```csharp
static void TrackTimedActionEnd(Platform::String ^action);
```

**Example:**  
```javascript
var ADB = ADBMobile;  
ADB.Analytics.trackTimedActionEnd("cartToCheckout");
```

| ClearTrackingQueue | Clears all stored hits from the Analytics tracking queue.  

**Syntax:**  
```csharp
static void ClearTrackingQueue();
```

**Example:**  
```javascript
ADBMobile.Analytics.clearTrackingQueue();
```

| GetQueueSizeAsync | Returns the number of hits currently stored in the Analytics queue.  

**Syntax:**  
```csharp
static int GetQueueSizeAsync();
```

**Example:**  
```javascript
ADBMobile.Analytics.GetQueueSizeAsync();
```
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| winJS: getQueueSizeAsync | **Syntax:**  
|                     | `static Windows::Foundation::IAsyncOperation<int> ^GetQueueSizeAsync();`  
|                     | **Example:**  
|                     | var queueSize;  
|                     | ADBMobile.Analytics.getQueueSizeAsync().then(function (size) {  
|                     | queueSize = size;  
|                     | });  
|                     | 28Analytics |
Target

Information to help you deliver targeted content within Universal Windows Platform applications.

Target Methods

List of Target methods provided by the Universal Windows Platform library.

The SDK currently has support for multiple Adobe Experience Cloud Solutions, including Analytics, Target, and Audience Manager.

*Lifecycle Metrics* are sent as parameters to each mbox load.

💡 *Note*: When you consume winmd methods from winJS (JavaScript), all methods automatically have their first letter lowercased.

Class Reference: TargetLocationRequest

Properties:

- `property Platform::String ^name;`
- `property Platform::String ^defaultContent;`
- `property Windows::Foundation::Collections::IMap<Platform::String^, Platform::Object^> ^parameters;`

String Constants (for ease of use when setting keys for custom parameters):

- `static property Platform::String ^TARGET_PARAMETER_ORDER_ID { Platform::String ^get() { return L"orderId"; } }`
- `static property Platform::String ^TARGET_PARAMETER_ORDER_TOTAL { Platform::String ^get() { return L"orderTotal"; } }`
- `static property Platform::String ^TARGET_PARAMETER_PRODUCT_PURCHASE_ID { Platform::String ^get() { return L"productPurchasedId"; } }`
- `static property Platform::String ^TARGET_PARAMETER_CATEGORY_ID { Platform::String ^get() { return L"categoryId"; } }`
- `static property Platform::String ^TARGET_PARAMETER_MBOX_3RDPARTY_ID { Platform::String ^get() { return L"mbox3rdPartyId"; } }`
- `static property Platform::String ^TARGET_PARAMETER_MBOX_PAGE_VALUE { Platform::String ^get() { return L"mboxPageValue"; } }`
- `static property Platform::String ^TARGET_PARAMETER_MBOX_PC { Platform::String ^get() { return L"mboxPC"; } }`
- `static property Platform::String ^TARGET_PARAMETER_MBOX_SESSION_ID { Platform::String ^get() { return L"mboxSession"; } }`
- `static property Platform::String ^TARGET_PARAMETER_MBOX_HOST { Platform::String ^get() { return L"mboxHost"; } }`

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoadRequest</td>
<td>Sends request to your configured Target server and returns the string value of the offer generated in a block <code>callback</code>.</td>
</tr>
<tr>
<td>winJS: loadRequest</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>Syntax:</td>
<td>static Windows::Foundation::IAsyncOperation&lt;Platform::String ^&gt; ^LoadRequest(TargetLocationRequest ^request);</td>
</tr>
<tr>
<td>Example:</td>
<td>var ADB = ADBMobile; ADB.Target.loadRequest(heroBannerRequest).then(function(content) { // do something with content returned from target server });</td>
</tr>
<tr>
<td>CreateRequest</td>
<td>Creates a TargetLocationRequest object with the given parameters.</td>
</tr>
<tr>
<td>Syntax:</td>
<td>static TargetLocationRequest ^CreateRequest(Platform::String ^name, Platform::String ^defaultContent, Windows::Foundation::Collections::IMap&lt;Platform::String^, Platform::Object^&gt; ^parameters);</td>
</tr>
<tr>
<td>Example:</td>
<td>var ADB = ADBMobile; var heroBannerRequest = ADB.Target.createRequest(&quot;heroBanner&quot;, &quot;default.png&quot;, null);</td>
</tr>
<tr>
<td>CreateOrderConfirmRequest</td>
<td>Creates a TargetLocationRequest object with the given parameters.</td>
</tr>
<tr>
<td>Syntax:</td>
<td>static TargetLocationRequest ^CreateOrderConfirmRequest(Platform::String ^name, Platform::String ^orderId, Platform::String ^orderTotal, Platform::String ^productPurchasedId, Windows::Foundation::Collections::IMap&lt;Platform::String^, Platform::Object^&gt; ^parameters);</td>
</tr>
<tr>
<td>Example:</td>
<td>var ADB = ADBMobile; var orderConfirm = ADB.Target.createOrderConfirmRequest(&quot;orderConfirm&quot;, &quot;order&quot;, &quot;47.88&quot;, &quot;3722&quot;, null);</td>
</tr>
<tr>
<td>ClearCookies</td>
<td>Clears Target cookies for the application on current device.</td>
</tr>
<tr>
<td>Syntax:</td>
<td>static void ClearCookies();</td>
</tr>
<tr>
<td>Example:</td>
<td>ADBMobile.Target.clearCookies();</td>
</tr>
<tr>
<td>GetPcId</td>
<td>Returns the PC ID cookie for the current device.</td>
</tr>
<tr>
<td>Syntax:</td>
<td>static Platform::String ^GetPcId();</td>
</tr>
<tr>
<td>Example:</td>
<td>auto pcId = ADBMobile.Target.getPcId();</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>GetSessionId</td>
<td>Returns the Session ID cookie for the current device.</td>
</tr>
<tr>
<td>winJS: getSessionId</td>
<td></td>
</tr>
</tbody>
</table>

**Syntax:**

```
static Platform::String ^GetSessionId();
```

**Example:**

```
auto sessionId = ADBMobile.Target.getSessionId();
```
## Audience Manager

Information to help you send signals and retrieve visitor segments from Audience Manager.

### Audience Manager Methods

List of Audience Manager methods provided by the Universal Windows Platform library.

The SDK currently has support for multiple Adobe Experience Cloud Solutions, including Analytics, Target, and Audience Manager. Methods are prefixed according to the solution. Audience Manager methods are prefixed with "AudienceManager."

*Note: When you consume winmd methods from winJS (JavaScript), all methods automatically have their first letter lowercased.*

If audience manager is configured in your JSON file, a signal containing lifecycle metrics is sent in with your lifecycle hit.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetVisitorProfile</td>
<td>Returns the visitor profile that was most recently obtained. Returns null if no signal has been submitted yet. Visitor profile is saved in SharedPreferences for easy access across multiple launches of your app.</td>
<td>static Windows::Foundation::Collections::IMap&lt;Platform::String^, Platform::Object^&gt; ^GetVisitorProfile();</td>
<td>var ADB = ADBMobile; var profile = ADB.AudienceManager.getVisitorProfile();</td>
</tr>
<tr>
<td>GetDpid</td>
<td>Returns the current DPID.</td>
<td>static Platform::String ^GetDpid();</td>
<td>var ADB = ADBMobile; var dpid = ADB.AudienceManager.getDpid();</td>
</tr>
<tr>
<td>GetDpuuid</td>
<td>Returns the current DPUUID.</td>
<td>static Platform::String ^GetDpuuid();</td>
<td>var ADB = ADBMobile; var dpuuid = ADB.AudienceManager.getDpuuid();</td>
</tr>
<tr>
<td>SetDpidAndDpuuid</td>
<td>Sets the DPID and DPUUID. If DPID and DPUUID are set, they will be sent with each signal.</td>
<td>static void SetDpidAndDpuuid(Platform::String ^dpid, Platform::String ^dpuuid);</td>
<td></td>
</tr>
</tbody>
</table>
### Example:

```javascript
var ADB = ADBMobile;
ADB.AudienceManager.setDpidAndDpuuid("newDpid", "newDpuuid");
```

<table>
<thead>
<tr>
<th>SignalWithData</th>
<th>Sends audience management a signal with traits and get the matching segments returned in a block callback.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>winJS:</strong> signalWithData</td>
<td></td>
</tr>
<tr>
<td><strong>Syntax:</strong></td>
<td><code>static Windows::Foundation::IAsyncOperation&lt;Windows::Foundation::Collections::IMap&lt;Platform::String^, Platform::Object^&gt;&gt; SignalWithData(Windows::Foundation::Collections::IMap&lt;Platform::String^, Platform::Object^&gt; ^data);</code></td>
</tr>
</tbody>
</table>
| **Example:** | ```javascript
var ADB = ADBMobile;
var traits = new Windows.Foundation.Collections.PropertySet();
traits["trait"] = "b";
ADB.AudienceManager.signalWithData(traits).then(function(visitorProfile)
{
    // segments come back here in "visitorProfile", normally found in the "segs" object of your json
});
``` |
Lifecycle Metrics

Lists the metrics and dimensions that can be measured automatically by the mobile library.

For more information, go to the Knowledge Base at Troubleshoot Lifecycle data.

This topic contains the following information:

- Lifecycle Metrics and Dimensions
- Additional Mobile Metrics and Dimensions

Lifecycle Metrics and Dimensions

When configured, lifecycle metrics are sent in context data parameters to Analytics, in parameters to Target with each mbox call, and as a signal to audience management. Analytics and Target use the same format, while audience management uses a different prefix for each metric.

For Analytics, the context data that is sent with each lifecycle tracking call is automatically captured in and reported on by using the metric or dimension listed in the first column, with the exceptions noted in the Description column.

Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Analytics Context Data/Target Parameter</th>
<th>Audience Manager Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Launches</td>
<td>a.InstallEvent</td>
<td>c_a_InstallEvent</td>
<td>Triggered at the first run after installation or re-installation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Upgrades</td>
<td>a.UpgradeEvent</td>
<td>c_a_UpgradeEvent</td>
<td>Triggered at the first run after an upgrade or when the version number changes.</td>
</tr>
<tr>
<td>Daily Engaged Users</td>
<td>a.DailyEngUserEvent</td>
<td>c_a_DailyEngUserEvent</td>
<td>Triggered when the application is used on a particular day.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Important: This metric is not automatically stored in an Analytics metric. You must create a processing rule that sets a custom event to capture this metric.</td>
</tr>
<tr>
<td>Monthly Engaged Users</td>
<td>a.MonthlyEngUserEvent</td>
<td>c_a_MonthlyEngUserEvent</td>
<td>Triggered when the application is used during a particular month.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Important: This metric is not automatically stored in an Analytics metric. You must create a processing rule that sets a custom event to capture this metric.</td>
</tr>
<tr>
<td>Launches</td>
<td>a.LaunchEvent</td>
<td>c_a_LaunchEvent</td>
<td>Triggered at every run, including crashes and installs. Also triggered on a resume from the</td>
</tr>
</tbody>
</table>
### Lifecycle Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Analytics Context Data/Target Parameter</th>
<th>Audience Manager Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>background when the lifecycle session timeout has been exceeded.</td>
</tr>
<tr>
<td>Crashes</td>
<td>a.CrashEvent</td>
<td>c_a_CrashEvent</td>
<td>Triggered when the application is not backgrounded before being closed. The event is sent when the application is started after the crash. Adobe Mobile crash reporting does not implement a global uncaught exception handler.</td>
</tr>
<tr>
<td>Previous Session Length</td>
<td>a.PrevSessionLength</td>
<td>c_a_PrevSessionLength</td>
<td>Reports the number of seconds that a previous application session lasted, based on how long the application was open and in the foreground.</td>
</tr>
</tbody>
</table>

### Dimensions

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Analytics Context Data/Target</th>
<th>Audience Management</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install Date</td>
<td>a.InstallDate</td>
<td>c_a_InstallDate</td>
<td>Date of first launch after installation. The date format is \textit{MM/DD/YYYY}.</td>
</tr>
<tr>
<td>App ID</td>
<td>a.AppID</td>
<td>c_a_AppID</td>
<td>Stores the application name and version in the following format: \textit{[AppName] [BundleVersion]}. An example of this format is \textit{myapp 1.1}.</td>
</tr>
<tr>
<td>Launch Number</td>
<td>a.Launches</td>
<td>c_a_Launches</td>
<td>Number of times the application was launched or brought out of the background.</td>
</tr>
<tr>
<td>Days since first use</td>
<td>a.DaysSinceFirstUse</td>
<td>c_a_DaysSinceFirstUse</td>
<td>Number of days since the first run.</td>
</tr>
<tr>
<td>Days since last use</td>
<td>a.DaysSinceLastUse</td>
<td>c_a_DaysSinceLastUse</td>
<td>Number of days since the last use.</td>
</tr>
<tr>
<td>Hour of Day</td>
<td>a.HourOfDay</td>
<td>c_a_HourOfDay</td>
<td>Measures the hour that the app was launched. This metric uses the 24-hour numerical format and is used for time parting to determine peak usage times.</td>
</tr>
<tr>
<td>Day of Week</td>
<td>a.DayOfWeek</td>
<td>c_a_DayOfWeek</td>
<td>Number of the day in a week when the app was launched.</td>
</tr>
<tr>
<td>Dimension</td>
<td>Analytics Context Data/Target</td>
<td>Audience Management</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>------------------------------</td>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Operating System Version</td>
<td>a.OSVersion</td>
<td>c_a_OSVersion</td>
<td>OS version.</td>
</tr>
<tr>
<td>Days since last upgrade</td>
<td>a.DaysSinceLastUpgrade</td>
<td>c_a_DaysSinceLastUpgrade</td>
<td>Number of days since the application version number has changed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Important:</strong> This metric is not automatically stored in an Analytics variable. You must create a processing rule to copy this value to an Analytics variable for reporting.</td>
</tr>
<tr>
<td>Launches since last upgrade</td>
<td>a.LaunchesSinceUpgrade</td>
<td>c_a_LaunchesSinceUpgrade</td>
<td>Number of launches since the application version number has changed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Important:</strong> This metric is not automatically stored in an Analytics variable. You must create a processing rule to copy this value to an Analytics variable for reporting.</td>
</tr>
<tr>
<td>Device Name</td>
<td>a.DeviceName</td>
<td>c_a_DeviceName</td>
<td>Stores the device name.</td>
</tr>
<tr>
<td>Carrier Name</td>
<td>a.CarrierName</td>
<td>c_a_CarrierName</td>
<td>Stores the name of the mobile service provider as provided by the device.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Important:</strong> This metric is not automatically stored in an Analytics variable. You must create a processing rule to copy this value to an Analytics variable for reporting.</td>
</tr>
<tr>
<td>Resolution</td>
<td>a.Resolution</td>
<td>c_a_Resolution</td>
<td>Width x Height in actual pixels.</td>
</tr>
</tbody>
</table>

**Additional Mobile Metrics and Dimensions**

The following metrics and dimensions are captured in mobile solution variables by the method listed in the **Description** column.

**Metrics**

<table>
<thead>
<tr>
<th>Event</th>
<th>Analytics Context Data/Target</th>
<th>Audience Management Trait</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action Time Total</td>
<td>a.action.time.total</td>
<td>c_a_action_time_total</td>
<td>Populated by trackTimedAction methods.</td>
</tr>
<tr>
<td>Action Time In App</td>
<td>a.action.time.inapp</td>
<td>c_a_action_time_inapp</td>
<td>Populated by trackTimedAction methods.</td>
</tr>
<tr>
<td>Event</td>
<td>Analytics Context Data/Target Parameter</td>
<td>Audience Management Trait</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------------</td>
<td>---------------------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>Lifetime Value (event)</td>
<td>a.ltv.amount</td>
<td>c_a_ltv_amount</td>
<td>Populated by \texttt{trackLifetimeValue} methods.</td>
</tr>
</tbody>
</table>

### Dimensions

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Analytics Context Data/Target Parameter</th>
<th>Audience Management Trait</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location (down to 10 km)</td>
<td>a.loc.lat.a</td>
<td>c_a_loc_lat_a</td>
<td>Populated by \texttt{trackLocation} methods.</td>
</tr>
<tr>
<td></td>
<td>a.loc.lon.a</td>
<td>c_a_loc_lon_a</td>
<td></td>
</tr>
<tr>
<td>Location (down to 100 m)</td>
<td>a.loc.lat.b</td>
<td>c_a_loc_lat_b</td>
<td>Populated by \texttt{trackLocation} methods.</td>
</tr>
<tr>
<td></td>
<td>a.loc.lon.b</td>
<td>c_a_loc_lon_b</td>
<td></td>
</tr>
<tr>
<td>Location (down to 1 m)</td>
<td>a.loc.lat.c</td>
<td>c_a_loc_lat_c</td>
<td>Populated by \texttt{trackLocation} methods.</td>
</tr>
<tr>
<td></td>
<td>a.loc.lon.c</td>
<td>c_a_loc.lon_c</td>
<td></td>
</tr>
<tr>
<td>Point of Interest Name</td>
<td>a.loc.poi</td>
<td>c_a_loc.poi</td>
<td>Populated by \texttt{trackLocation} methods when device is within a defined POI.</td>
</tr>
<tr>
<td>Distance to Point of Interest Center</td>
<td>a.loc.dist</td>
<td>c_a_loc_dist</td>
<td>Populated by \texttt{trackLocation} methods when device is within a defined POI.</td>
</tr>
<tr>
<td>Lifetime Value (conversion variable)</td>
<td>a.ltv.amount</td>
<td>c_a_ltv_amount</td>
<td>Populated by \texttt{trackLifetimeValue} methods.</td>
</tr>
</tbody>
</table>
Extensions

Information to help you use extensions with the Universal Windows Platform SDK for Experience Cloud Solutions.

Windows Visual Studio Extensions for Experience Cloud Solutions 4.x SDK

This Extensions provides you a much easier way to add the reference of Experience Cloud Solutions 4.x Windows SDK in your project.

This section contains the following information:

- Install from GitHub
- Add References to Your Project

Install from GitHub

1. Download the Windows Universal SDK from GitHub.
2. Unzip the downloaded file locally.
3. Double-click ADBMobileUniversalWindowsVSIX.vsix to open the installer.
4. Select Global Location, and then install the lib.

Add References to Your Project

1. Open your Windows 10 project.
2. Open the dialogue of Reference Manager.
3. In the Extensions tab of Windows 10, locate Adobe Mobile SDK, then select it.

4. Click **OK** to save it.

   The Adobe Mobile SDK will be added to your project, together with Microsoft Visual C++ Runtime Package if it has not been added previously.

5. Choose a Platform type in the Configuration Manager, then begin testing your app.
Using Bloodhound to Test Mobile Applications

Lets you send server calls to a local computer to test mobile applications.

During application development, Bloodhound lets you view server calls locally, and optionally forward the data to Adobe collection servers.

Bloodhound is available for Mac and Windows.

Requirements

- An Intel-based Mac computer running Snow Leopard (10.6) or later, or a Windows PC.
- Network connectivity to your mobile devices or simulators.

Using Bloodhound

For more information about Bloodhound, see the Bloodhound User Guide.

⚠️ Important: As of April 30, 2017, Adobe Bloodhound has been sunset. Starting on May 1, 2017, no additional enhancements and no additional Engineering or Adobe Expert Care support will be provided.
4.x Migration Guide

This section describes how to migrate from the 3.x version of a previous Windows mobile SDK to the Universal Windows Platform 4.x SDK for Experience Cloud Solutions.

With the move to version 4.x, all functionality is now accessible through static methods, so no more keeping track of your own objects.

The following sections walk you through migrating from version 3.x to version 4.x.

Remove Unused Properties

You probably noticed a new ADBMobileConfig.json file included with your download. This file contains application-specific, global settings, and replaces most of the configuration variables that were used in previous versions. Here is an example of an ADBMobileConfig.json file:

```json
{
  "version" : "1.0",
  "analytics" : {
    "rsids" : "coolApp",
    "server" : "my.CoolApp.com",
    "charset" : "UTF-8",
    "ssl" : false,
    "offlineEnabled" : true,
    "lifecycleTimeout" : 300,
    "privacyDefault" : "optedin",
    "poi" : [
      ["san francisco",37.757144,-122.44812,7000],
      ["santa cruz",36.972935,-122.01725,600]
    ],
    "target" : {
      "clientCode" : "myTargetClientCode",
      "timeout" : 5
    },
    "audienceManager" : {
      "server" : "myServer.demdex.com"
    }
  }
}
```

The following tables list the configuration variables that you need to move to the configuration file. Move the value set for the variable in the first column to the variable in the second column, and then remove the old configuration variable from your code.

### Migrating from 3.x:

<table>
<thead>
<tr>
<th>Configuration Variable/Method</th>
<th>Variable in ADBMobileConfig.json</th>
</tr>
</thead>
<tbody>
<tr>
<td>offlineTrackingEnabled</td>
<td>&quot;offlineEnabled&quot;</td>
</tr>
<tr>
<td>reportSuiteIDs</td>
<td>&quot;rsids&quot;</td>
</tr>
<tr>
<td>trackingServer</td>
<td>&quot;server&quot;</td>
</tr>
<tr>
<td>charSet</td>
<td>&quot;charset&quot;</td>
</tr>
<tr>
<td>currencyCode</td>
<td>&quot;currency&quot;</td>
</tr>
</tbody>
</table>
### Update Track Calls and Tracking Variables

Instead of using the web-focused `Track` and `TrackLink` calls, the version 4 SDK uses two methods that make a little more sense in the mobile world:

- **TrackState** States are the views that are available in your app, such as "home dashboard", "app settings", "cart", and so on. These states are similar to pages on a website, and `trackState` calls increment page views.
- **TrackAction** Actions are the things that happen in your app that you want to measure, such as "logons", "banner taps", "feed subscriptions", and other metrics. These calls do not increment page views.

The `contextData` parameter for both of these methods contains name-value pairs that are sent as context data.

### Events, Props, eVars

If you’ve looked at the SDK Methods, you are probably wondering where to set events, eVars, props, heirs, and lists. In version 4, you can no longer assign those types of variables directly in your app. Instead, the SDK uses context data and processing rules to map your app data to Analytics variables for reporting.

Processing rules provide you several advantages:

- You can change your data mapping without submitting an update to the App Store.
- You can use meaningful names for data instead of setting variables that are specific to a report suite.
- There is little impact to sending in extra data. These values won’t appear in reports until they are mapped using processing rules.

See **Processing Rules**.

Any values that you were assigning directly to variables should be added to context data instead. This means that calls to `SetProp`, `SetEvar`, and assignments to persistent context data should all be removed and the values added to context data.

### AppSection/Server, GeoZip, Transaction ID, Campaign, and other standard variables

Any other data that you were setting on the measurement object, including the variables listed above, should be added to context data instead.

To put it simply, the only data sent in with a `trackState` or `trackAction` call is the payload in the `data` parameter.

### Replace Tracking Calls

Throughout your code, replace the following methods with a call to `trackState` or `trackAction`:

#### Migrating from 3.x:

- `TrackAppState` (TrackState)
- `TrackEvents` (TrackAction)
- `Track` (TrackAction)
- `TrackLinkURL` (TrackAction)
Custom ID Service
Replace the visitorID variable with a call to setUserIdentifier.

Offline Tracking
Offline tracking is enabled in ADBMobileConfig.json. All other offline configuration is done automatically.
Throughout your code, remove calls to the following methods:

Migrating from 3.x:
- SetOnline
- SetOffline

Products Variable
Since the products variable is not available in processing rules, you can use the following syntax to set products:

```javascript
// create a processing rule to set the corresponding product event.  
// for example, set the Product Views event when context data a.action = "product view"
var cdata = new Windows.Foundation.Collections.PropertySet();
cdata["&&products"] = ";Cool Shoe";
ADB.Analytics.trackAction("product view", cdata);
```

The value of "&&products" (in this example, the value is ";Cool Shoe") should follow the products string syntax for the type of event that you are tracking.

Test the Migration
After you have completed the migration, see Using Bloodhound to Test Mobile Applications to find out how to inspect the data being sent by the mobile SDK.
Contact and Legal Information

Information to help you contact Adobe and to understand the legal issues concerning your use of this product and documentation.

Help & Technical Support

The Adobe Experience Cloud Customer Care team is here to assist you and provides a number of mechanisms by which they can be engaged:

• Check the Experience Cloud help pages for advice, tips, and FAQs
• Ask us a quick question on Twitter @AdobeExpCare
• Log an incident in our customer portal
• Contact the Customer Care team directly
• Check availability and status of Experience Cloud Solutions

Service, Capability & Billing

Dependent on your solution configuration, some options described in this documentation might not be available to you. As each account is unique, please refer to your contract for pricing, due dates, terms, and conditions. If you would like to add to or otherwise change your service level, or if you have questions regarding your current service, please contact your Account Manager.

Feedback

We welcome any suggestions or feedback regarding this solution. Enhancement ideas and suggestions can be added to our Customer Idea Exchange.

Legal

© 2017 Adobe Systems Incorporated. All Rights Reserved.
Published by Adobe Systems Incorporated.

Terms of Use | Privacy Center

Adobe and the Adobe logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. A trademark symbol (®, ™, etc.) denotes an Adobe trademark.

All third-party trademarks are the property of their respective owners. Updated Information/Additional Third Party Code Information available at http://www.adobe.com/go/thirdparty.