ADOBE CONSULTING

Adobe Analytics Technical Specification

Deploying the AEP SDK via Launch for iOS Swift

July 1, 2020

TABLE OF CONTENTS

REVISION HISTORY

VERSION	DATE	AUTHOR	SUMMARY OF CHANGES
1.0	03/21/2022	Sheetal Raina	Initial version

DOCUMENT OVERVIEW

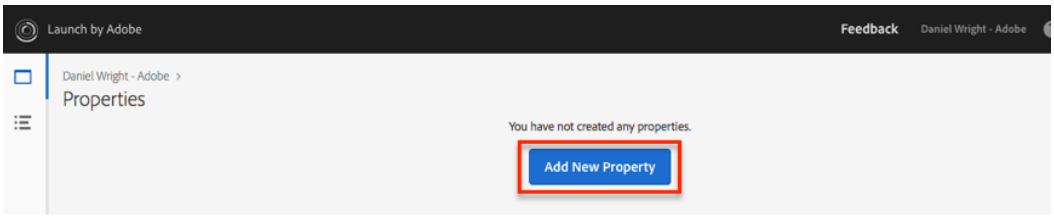
This tech spec contains basic setup best practices for implementing the Adobe Experience Platform SDK on iOS Swift using Launch by Adobe. As a companion to this guide, a reference implementation using these best practices can be found at: https://aep-sdks.gitbook.io/docs/

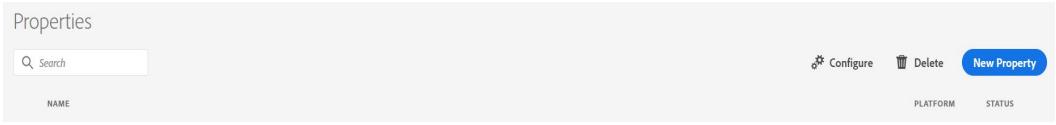
Please note that the below Launch configuration will be done by Adobe Analytics Consultant.

GENERAL LAUNCH CONFIGURATIONS & SETTINGS

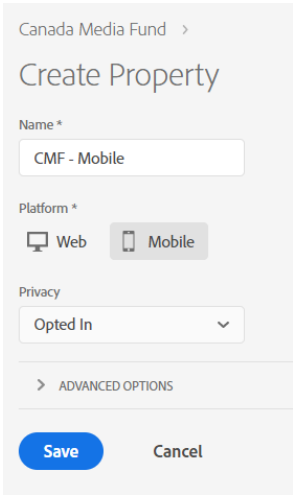
CONFIGURATION 1: CREATE A PROPERTY

1. Click the "Add New Property" or "New Property" button:





2. Name your property and select "Mobile" as the platform.



3. By default, "Opted In" privacy settings. This can be changed later if needed.
4. Don't change any of the Advanced Settings.

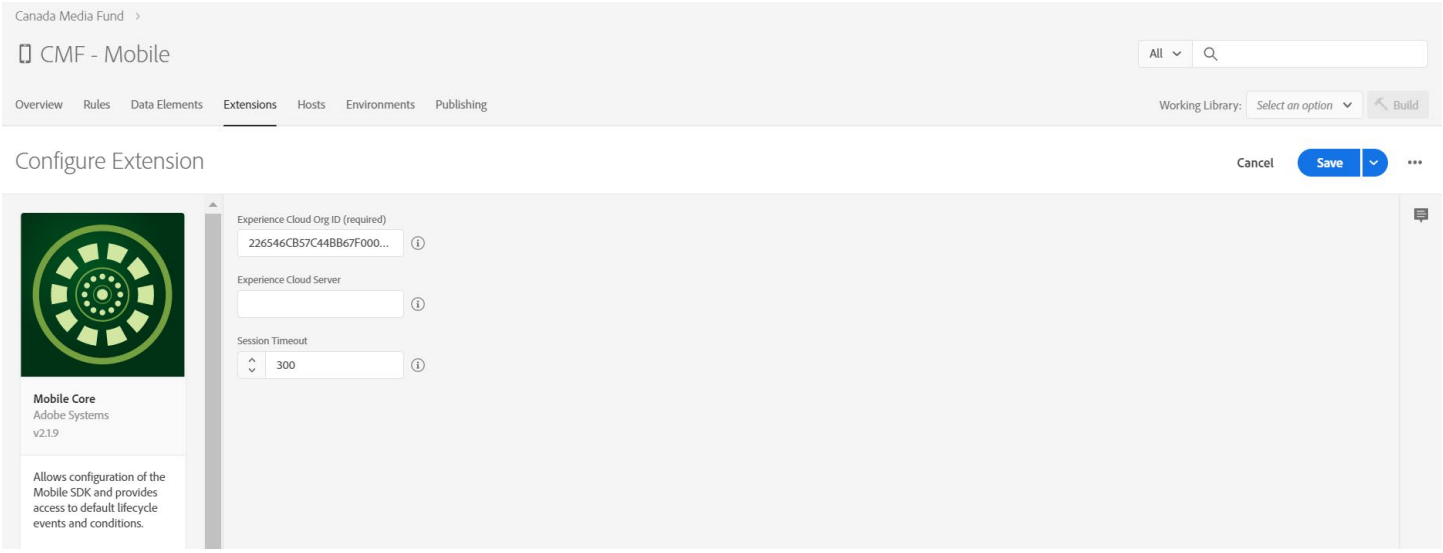
5. Click "Save"

Your new property should display on Property list page. Click on the name of your property to open the Overview screen and then click on the Extensions tab. Note that the "Core" and "Profile" extensions are automatically added when you created the Property.

CONFIGURATION 2: CONFIGURE EXTENSIONS

Start by configuring your "Core" extension that has already been installed by default.

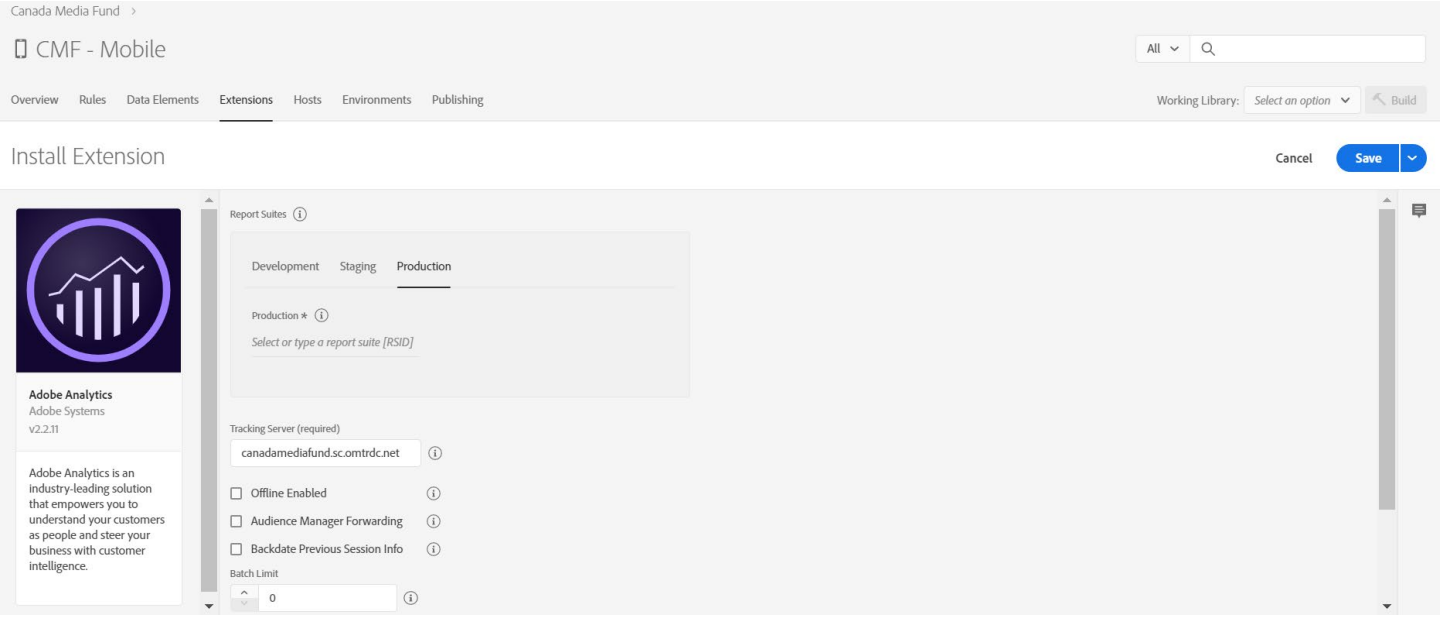
1. Click **Configure** on Mobile Core and you will see the below screen.



2. Provide your **Experience Cloud Org ID**. (by default this value is auto-populated using the currently signed-in Organization ID). *This is a required identifier for your Experience Cloud Organization. This is typically, a 24-character, alphanumeric string followed by @AdobeOrg. If you need help finding it, contact your Adobe CSM or Customer Care.*
3. Optionally, provide your **Experience Cloud ID Server**. This is an optional server value which will be used to send Visitor ID Service network requests to a custom endpoint.
4. Optionally, change the **Session Timeout** value. A default value of 300 seconds is already set. This timeout value indicates the number of seconds that must pass after a user backgrounds the app before we consider a launch a new Lifecycle session.
5. Click **Save** to confirm your settings for **Mobile Core**.

Configure the Analytics Extension

1. On the **Catalog** tab, locate the **Adobe Analytics** extension and click **Install**.



2. Provide extension settings
    a. **Report Suites**: Type one or more report suite identifiers to which the Analytics data should be sent. To add multiple report suite IDs, click **Add Another**; to remove these IDs, click **Remove Circle.** Report suite IDs can also be configured for the Development and Staging environments.
    b. **Tracking Server**: Provide the tracking domain to which all Analytics requests should be made. For help determining this – see this link.
    c. **Offline Tracking**: Offline tracking allows you to store measurement calls when the application is offline. When this box is checked, Analytics 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.
    d. **Audience Manager Forwarding:** If you set up Analytics server-side forwarding to Audience Manager, check this setting. When this setting is enabled, all SDK requests to Analytics servers are sent with an expected response code of 10. This step ensures Analytics traffic is forwarded to Audience Manager and that the Audience Manager User Profile is correctly updated in the SDK.
    e. **Backdate Previous Session Info:** Enabling this setting will cause the SDK to backdate end-of-session lifecycle information so it can be attributed into its correct session. Session information currently consist of crashes and session length. When enabled, the SDK will backdate the session information hit to 1 second after the last hit of the previous session. This means that crashes and session data will correlate with the correct date in which they happened. One hit will be backdated on every new launch of the application. For instance, if this setting is checked, Lifecycle session information or crash events will be backdated to one second after the last hit was sent. If unchecked, Lifecycle data will be attached to the first hit of the subsequent session. When disabled, the Adobe SDK will attach the session info to the current lifecycle.
    f. **Batch Limit:** This setting creates a threshold number of hits to be sent in consecutive calls. For example, this setting is set to 10, each Analytics hit before the 10th hit will be stored in the queue.

When the 10th hit comes in, all 10 hits will be sent to Analytics, consecutively. The default value for this setting is 0, which means that hit batching is disabled and all hits will be immediately sent to Analytics as they are generated. *If you're batching, you need to ensure that Offline Tracking is also enabled.*

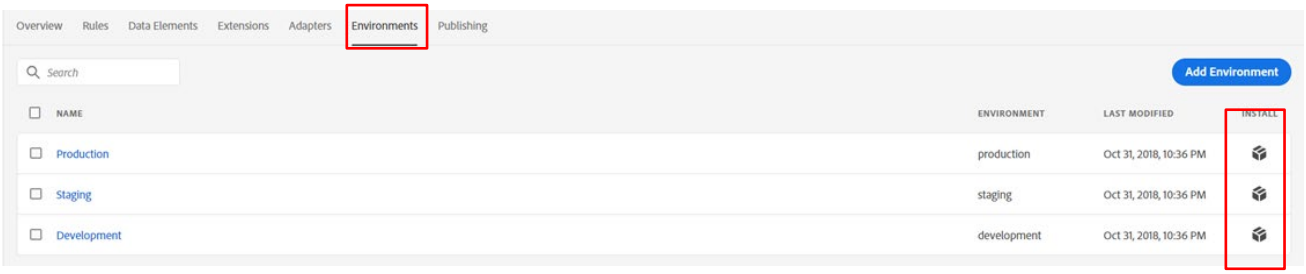
g. **Launch Hit Delay:** Number of seconds to wait before Analytics launch hits are sent from the SDK. Ensure that this setting is set at 5s or greater when using acquisition functionality from the Mobile Services extension.

3. Click **Save**.
4. Follow the publishing process, to update SDK configuration.

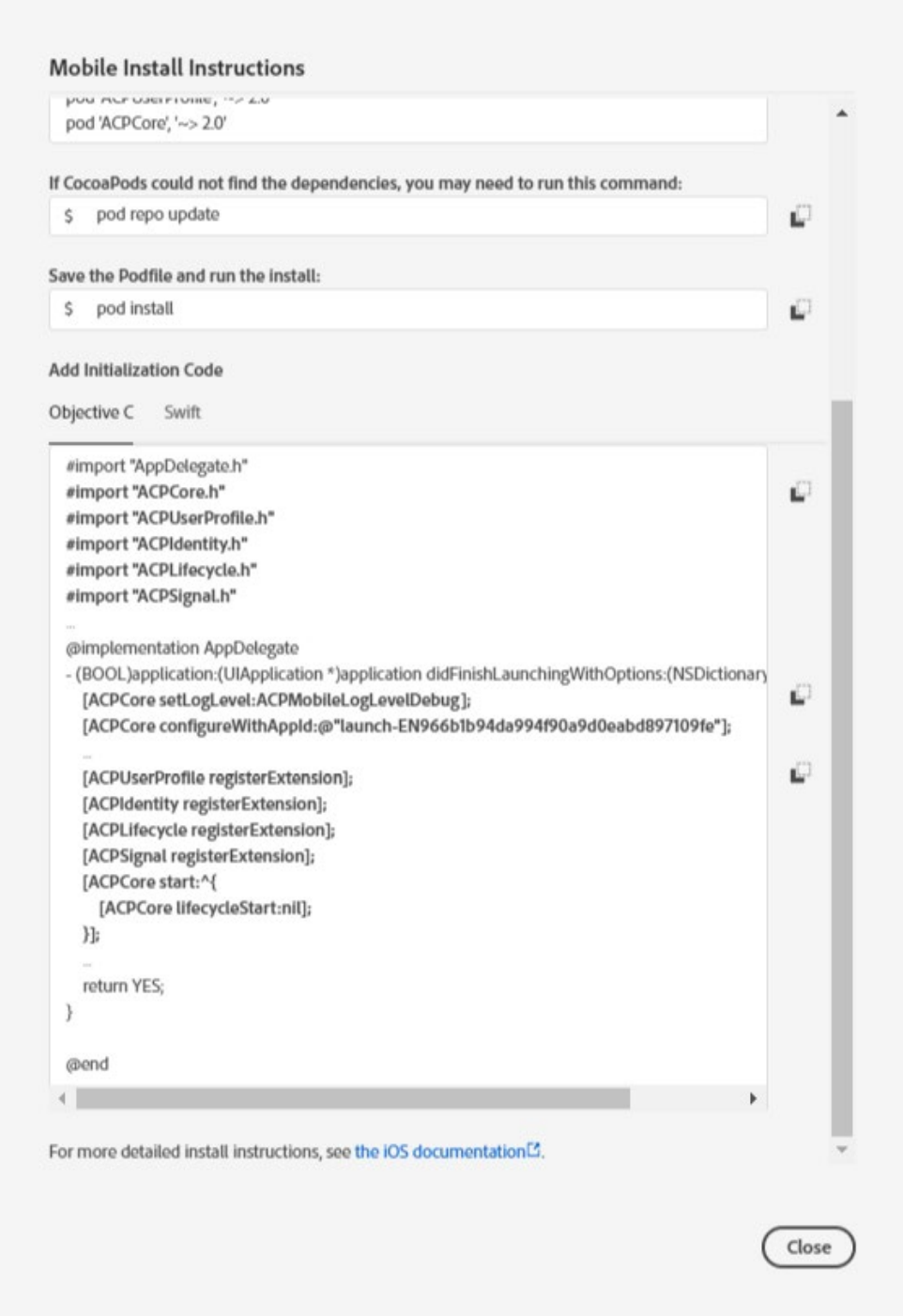
Note: If an extension is added after the SDK has been deployed, you will need to re-install/publish the SDK install code (described in the next step)

CONFIGURATION 3: GET THE SDK

1. Click on the **Environments** tab to view instructions for adding the SDKs to an app.



2. Find the platform needed in the table and click on the box icon under the Install column. You should see a pop up similar to below.



3. On the Mobile Install Instructions pop-up, choose iOS.
4. Follow the instructions for using CocoaPods with iOS.
    a. Cocoapods Documentation: https://guides.cocoapods.org/using/using-cocoapods

Installing the SDK Manually

In order to do a manual installation of the AEP SDK libraries, please complete the following steps:

- Download the extensions needed by your app from the iOS/ directory-

https://github.com/Adobe-Marketing-Cloud/acp-sdks/tree/master/iOS

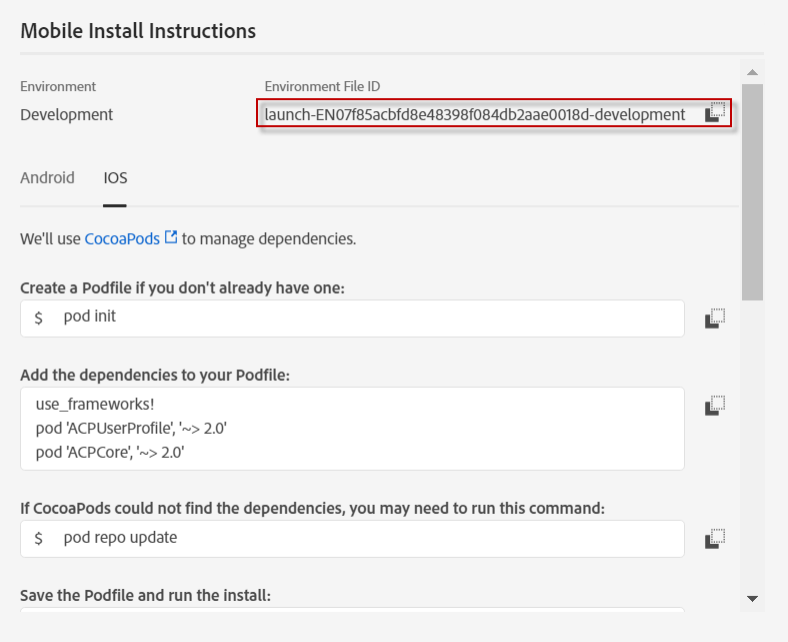
In the Xcode project create a new Group, and then drag all the folders you just download and drop them under the group. And verify the following:

    - The Copy Items if needed checkbox is selected.
    - Create groups is selected.
    - In the Add to targets section select all the targets that need AEP SDKs.
- Select your project from the Project Navigator, select your App from the TARGETS frame, then select the General tab at the top of the window.
- In the Link Binary With Libraries section, click the + link and add the following frameworks and libraries: UIKit, SystemConfiguration, WebKit, UserNotifications, libsqlite3.0, libc++, libz.

Important - note that all AEP SDK iOS libraries depend on libACPCore.a.

INITIALIZE THE SDK & ENABLE DEBUG LOGS

To initialize the SDK, you'll first need to configure the SDK with the Environment ID from Launch. To find your Environment ID, navigate to Environments tab in Launch, and click on the Install box icon corresponding to the environment you are setting up.



Navigate to your app project.

1. **Add your Environment ID:**

In Xcode, find your `didFinishLaunchingWithOptions` located in **AppDelegate.swift** and add:

```
ACPCore.configure(withAppId: "PASTE_ENVIRONMENT_ID_HERE")
```

Environment IDs

Dev: e5b7f45869ff/d78455ec7bc7/launch-c511da7c2da3-development

Staging: e5b7f45869ff/d78455ec7bc7/launch-3848aa4ea353-staging

Production: e5b7f45869ff/d78455ec7bc7/launch-efb05412c8d5

Note: Reach out to adobe consulting to provide you correct Launch App ID value for development, staging and production env.

2. **Enable Debug Logging:**
    a. Debug logging helps you ensure that the SDK is working as intended. Below is a table that explains the levels of logging available and when they may be used.

Note: Reach out to adobe consulting to provide you correct Launch App ID value for development, staging and production env.

Log Level	Description
Error	This log level details unrecoverable errors caused during SDK implementation.
Warning	In addition to detail from Error log level, Warning provides error information during SDK integration. This log level may indicate that a request has been made to the SDK, but the SDK may be unable to perform the requested task. For example, this may be used when catching an unexpected but recoverable exception and printing its message.
Debug	In addition to detail from Warning log level, Debug also provides high-level information on how the SDK processes network requests/responses data.
Verbose	In addition to detail from the Debug level, Verbose provides detailed, low-level information into how SDK processes database interactions and SDK events.

b. Add the below lines of code (uncomment any levels of debugging you want added):

```
ACPCore.setLogLevel(ACPMobileLogLevel.debug)
// ACPCore.setLogLevel(ACPMobileLogLevel.verbose)
// ACPCore.setLogLevel(ACPMobileLogLevel.warning)
// ACPCore.setLogLevel(ACPMobileLogLevel.error)
```

MOBILE INSTALL INSTRUCTIONS

We'll use CocoaPods to manage dependencies.

1. Create a Podfile if you don't already have one:

```
pod init
```

2. Add the dependencies to your Podfile:

```
use_frameworks!
pod 'ACPAnalytics', '~> 2.0'
pod 'ACPUserProfile', '~> 2.0'
pod 'ACPCore', '~> 2.0'
```

3. If CocoaPods could not find the dependencies, you may need to run this command:

```
pod repo update
```

4. Save the Podfile and run the install:

```
pod install
```

5. Add Initialization Code (Change the environment code as per the environment)

```
import ACPCore
import ACPAnalytics
import ACPUserProfile

...
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {
    var window: UIWindow?
    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {

        ACPCore.setLogLevel(.debug)
        ACPCore.configure(withAppId: "e5b7f45869ff/d78455ec7bc7/launch-efb05412c8d5")

        ...

        ACPAnalytics.registerExtension()
        ACPUserProfile.registerExtension()
        ACPIdentity.registerExtension()
        ACPLifecycle.registerExtension()
        ACPSignal.registerExtension()
        ACPCore.start {
            ACPCore.lifecycleStart(nil)
        }
       ...
       return true
       }
}
```

ENABLE THE EXPERIENCE CLOUD ID SERVICE

The Experience Cloud ID service provides a cross-channel notion of identity across Experience Cloud solutions and is pre-requisite for most implementations.

1. In swift, the ACPCore includes ACPIdentity :

```
import ACPCore
```

2. Register the framework with Mobile Core

The `registerExtension()` API registers the Identity extension with the Mobile Core extension. This API allows the extension to send and receive events to and from the Mobile SDK.

Register the Identity extension in your app's `didFinishLaunchingWithOptions` function:

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
  ACPIdentity.registerExtension();
  // Override point for customization after application launch.
  return true;
}
```

After successful configuration, you'll see the Experience Cloud ID generated and included on every network hit sent to Adobe.

****NOTE: If you have installed the Core extension and followed the steps of adding the SDK to your project via the Launch install codes, these steps have already been completed.***

PASSING VISITOR ID FROM NATIVE APP TO WEB VIEW

In some instances, mobile applications will launch mobile web (or "webview") content within the window of the application. If your app opens mobile web content, you need to take extra steps to ensure that visitors are not separately identified as they move between the native and mobile web in a hybrid app.

VISITOR IDS WITHIN APPS

The SDK generates a unique visitor ID when an app is installed. This app visitor ID is stored within persistent memory on the mobile device and is sent with every hit. The app visitor ID is removed only when the user uninstalls the app (app visitor IDs persist through upgrades).

VISITOR IDS WITHIN MOBILE WEB

Typical mobile web implementations use the same standard `AppMeasurement.js` that is used within desktop sites. The JavaScript libraries have their own methods of generating unique visitor IDs, which causes a different visitor ID to be generated when you open mobile web content from your app.

To use the same visitor ID in the app and mobile web, complete the following instructions to pass the app visitor ID to the mobile web.

1. When a webview is opened in your mobile app, call the `appendToURL` method. See the example below for reference:

Note: Method `appendToUrl:withCompletionHandler` was added in ACPCore version 2.5.0 and ACPIdentity version 2.2.0.

Syntax:

```
+ (void) appendToUrl: (nullable NSURL*) baseUrl withCallback: (nullable void (^) (NSURL* __nullable
urlWithVisitorData)) callback;
+ (void) appendToUrl: (nullable NSURL*) baseUrl withCompletionHandler: (nullable void (^) (NSURL*
__nullable urlWithVersionData, NSError* __nullable error)) completionHandler;
```

- *baseUrl* is the URL to which the visitor information needs to be appended. If the visitor information is nil or empty, the URL is returned as is.
- *callback* is invoked after the updated URL is available.
- *completionHandler* is invoked with *urlWithVersionData* after the updated URL is available or *error* if an unexpected exception occurs or the request times out. The returned `NSError` contains the ACPError code of the specific error. The default timeout is 500ms.

Code Sample:

```
ACPIdentity.append(to:URL(string: "https://example.com"), withCallback: {(appendedURL) in
  // handle the appended url here
  if let appendedURL = appendedURL {
    // APIs which update the UI must be called from main thread
    DispatchQueue.main.async {
      self.webView.load(URLRequest(url: appendedURL!))
    }
  } else {
    // handle error, nil appendedURL
  }
});
```

```
ACPIdentity.append(to: URL(string: "https://example.com"), withCompletionHandler: { (appendedURL, error) in
  if let error = error {
    // handle error
  } else {
    // handle the appended url here
    if let appendedURL = appendedURL {
      // APIs which update the UI must be called from main thread
      DispatchQueue.main.async {
        self.webView.load(URLRequest(url: appendedURL!))
      }
    } else {
      // handle error, nil appendedURL
    }
  }
})
```

By utilizing the `appendVisitorInfoForURL` you can append Adobe visitor data to a URL string. If the provided URL is null or empty, it is returned as is. Otherwise, the following information is added to the url string that is returned in the AdobeCallback instance:
The `adobe_mc` attribute is an URL encoded list containing:

- Experience Cloud ID (ECID)
- Experience Cloud Org ID
- A timestamp taken when this request was made
- The optional adobe_aa_vid attribute is the URL encoded Analytics Custom Visitor ID, if available

ENABLE LIFECYCLE METRICS

Lifecycle metrics are valuable, out-of-the-box information about your app user. These metrics contain information on the app user's lifecycle such as device information, install or upgrade information, session start and pause times, etc. Here is a complete list of metrics available using Lifecycle.

1. In Swift, importing `ACPCore` also imports the necessary Lifecycle APIs:

```
import ACPCore
```

2. Register the Lifecycle extension with the SDK Core by adding the following to your app's `application:didFinishLaunchingWithOptions:` delegate method:

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
ACPLifecycle.registerExtension();
  // Override point for customization after application launch
  return true;
}
```

NOTE: If you have installed the Core extension and followed the steps of adding the SDK to your project via the Launch install codes, Steps 1 and 2 have already been completed.

3. Start Lifecycle data collection by calling `lifecycleStart:` from within the callback of the `ACPCore::start:` method in your app's `application:didFinishLaunchingWithOptions:` delegate method. If your iOS application supports background capabilities, your `application:didFinishLaunchingWithOptions:` method might be called when iOS launches your app in the background. If you do not want background launches to count towards your lifecycle metrics,

then `lifecycleStart:` should only be called when the application state is not equal to `UIApplicationStateBackground`.

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplicationLaunchOptionsKey: Any]?) -> Bool {
    // register the lifecycle extension
    ACPLifecycle.registerExtension();

    let appState = application.applicationState;
    ACPCore.start {
        // only start lifecycle if the application is not in the background
        if appState != .background {
            ACPCore.lifecycleStart(nil)
        }
    }
}
```

4. When launched, if your app is resuming from a backgrounded state, iOS might call your `applicationWillEnterForeground:` delegate method. You also need to call `lifecycleStart:`, but this time you do not need all of the supporting code that you used in `application:didFinishLaunchingWithOptions::`

```
func applicationWillEnterForeground(_ application: UIApplication) {
    ACPCore.lifecycleStart(nil)
}
```

In iOS 13 and later, for a scene-based application, use the `UISceneDelegate`'s `sceneWillEnterForeground` method as follows:

```
func sceneWillEnterForeground(_ scene: UIScene) {
     ACPCore.lifecycleStart(nil)
}
```

5. When the app enters the background, pause Lifecycle data collection from your app's `applicationDidEnterBackground:` delegate method:

```
func applicationDidEnterBackground(_ application: UIApplication) {
    ACPCore.lifecyclePause()
}
```

In iOS 13 and later, for a scene-based application, use the `UISceneDelegate`'s `sceneDidEnterBackground` method as follows:

```
func sceneDidEnterBackground(_ scene: UIScene) {
     ACPCore.lifecyclePause()
}
```

TRACKING SCREENS AND ACTIONS

You may use the trackState and trackAction APIs in order to start measuring your user's engagement with your app.
States represent screens or views in your app. Each time a new state is displayed in your application, for example, when a user navigates from the home page to the news feed, the trackState API may be called. This method sends an Analytics state tracking hit (equivalent of a page view) with optional context data.
Actions are events that occur in your app. Use the trackAction API to track and measure an action. Each action has one or more corresponding metrics that are incremented each time the event occurs. For example, you might call this API for each new subscription each time an article is viewed, or each time a level is completed.

***The sections detailed below provide general guidelines for using context data to collect custom variable requirements on screens and actions. These sections should be built out according to your client's specified tracking requirements.*

DEPLOYMENT INSTRUCTIONS FOR SCREEN TRACKING

Syntax

```
+ (void) trackState: (nullable NSString*) state data: (nullable NSDictionary*) data;
```

Code Example for Home Page:

```
ACPCore.trackState("home page", data: ["key": "value"])
```

Adding custom tracking dimensions

Code Example for capturing language and country code along with the pageName variable on the home page:

```
ACPCore trackState("homepage", data:[
"my.app.language":"en"
"my.app.countrycode":"us"
]
)
```

DEPLOYMENT INSTRUCTIONS FOR ACTION TRACKING

Syntax

```
+ (void) trackAction: (nullable NSString*) action data: (nullable NSDictionary*) data;
```

Code Example for tracking a click on the Login button:

```
ACPCore.trackAction("loginclicked", data: ["key": "value"])
```

Adding custom tracking dimensions

Code Example for capturing language and country code along with the login clicked action:

```
ACPCore trackAction("loginclicked", data:[
"my.app.language":"en"
"my.app.countrycode":"us"
]
)
```

MAPPING CONTEXT DATA VARIABLES

With the AEP SDK, context data variables must be mapped to the eVars, props and events as defined in the implementation solutions. In the examples above, Language and Country Code would need to be mapped to assigned eVars via the Adobe Analytics Processing Rules interface in order to report on and view the data.

Learn more about Processing Rules at
https://marketing.adobe.com/resources/help/en_US/reference/processing_rules.html

SOLUTION COMPONENTS

SITE CONTENT EFFECTIVENESS

This solution allows the clients to evaluate the effectiveness of the app's pages, including their influence on conversion, landing page, bounce rate and exit rate as well as gives marketers the ability to drill down on each level of your app hierarchy. To do that, it requires that every page/screen of the app be tagged with an effective page name and variables to represent each level of the content hierarchy.

VARIABLES IN THIS SECTION

The below table consists of variables used for this solution.

Analytics Variable	Used For	Context Data Variable
pageName	Pages	N/A (argument of trackState call)
prop5	Language	cmf.app.pageLanguage
prop7	Time Zone	cmf.app.userTimeZone
prop21	Media Name	cmf.app.mediaName
eVar1	Application ID	cmf.app.applicationID
eVar7	Page Name	cmf.app.pageName
eVar21	Media Name	cmf.app.mediaName
eVar35	ECID	cmf.app.ecid
eVar36	User ID	cmf.app.userID
event34	App Page Views	cmf.app.appPageView

DEPLOYMENT INSTRUCTIONS

CONTEXT DATA - ALL SCREENS

Set the following context data variables on every screen and trigger a 'trackState' call (typically on load of every screen):

Context Data Variable	Notes
pageName (argument of trackState call)	Set a clear, descriptive page name. Consider including the type or category of page to start the page name. Continue the page name with each level of the site hierarchy until the final descriptive portion of the page. Use a pipe "|" or ":" as the delimiter between each portion of the pageName.
cmf.app.pageName	Set a clear, descriptive page name. Consider including the type or category of page to start the page name. Continue the page name with each level of the site hierarchy until the final descriptive portion of the page. Use a pipe "|" or ":" as the delimiter between each portion of the pageName.

cmf.app.pageLanguage	Set this variable to the language of the page. Use two letter code for language. Example: "en" for English, "fr" for French etc.
cmf.app.userTimeZone	Set this variable to the time zone of the user browsing the app.
cmf.app.mediaName	Set this variable to media name. The media name should be descriptive and clear and should depict the media/content served by your website. For example: "Maths Experiments", "Multiplayer game" etc.
cmf.app.applicationID	Set this variable to the application ID which will be provided by CMF. For example: "1314.12112.134933"
cmf.app.ecid	Set this variable to the Experience Cloud ID set by the app. Use the "getExperienceCloudId" API as explained below to populate this variable.
cmf.app.userID	Set this variable to a custom visitor ID if the user is logged in or if there is any other way to identify the user uniquely. For example: "US1234", "QA1234" etc. If there is no such ID, then set it as "NA"
cmf.app.appPageView	Set this to 1.

Page Naming Best Practice

- **Context -** Include the directory structure or content hierarchy in the page name to help users understand where the page "lives" within the site. This also allows for simplified report filtering.
- **Clarity -** Ensure the page name is clear and easily identifiable for infrequent users. This will promote faster adoption of Adobe Analytics by business users and also allow for easier, more efficient reporting.
- **Conciseness -** Keep the page name as short as possible to maximize limited character space. This is also important for clean, simple reports but should not take precedence over context and clarity.

getExperienceCloudId

This API retrieves the ECID that was generated when the app was initially launched and is stored in the ECID Service.
This ID is preserved between app upgrades, is saved and restored during the standard application backup process and is removed at uninstall. The values are returned via the callback.
Syntax:

```
+ (void) getExperienceCloudId: (nonnull void (^) (NSString* __nullable experienceCloudId))
callback;
+ (void) getExperienceCloudIdWithCompletionHandler: (nonnull void (^) (NSString* __nullable
experienceCloudId, NSError* __nullable error)) completionHandler;
```

- *callback* is invoked after the ECID is available.
- completionHandler is invoked with *experienceCloudId* after the ECID is available, or *error* if an unexpected error occurs or the request times out. The returned `NSError` contains the ACPError code of the specific error. The default timeout is 500ms.

Example:

```
ACPIdentity.getExperienceCloudId { (retrievedCloudId) in
    // handle the retrieved ID here
}

ACPIdentity.getExperienceCloudId { (retrievedCloudId, error) in
  if let error = error {
    // handle error here
  } else {
    // handle the retrieved ID here
  }
}
```

Context Data Format:

```
ACPCore.trackState("<page Name>",data:[
"cmf.app.pageName":"<Page Name>",
"cmf.app.pageLanguage":"<Page Language>",
"cmf.app.userTimeZone":"<User Time Zone>",
"cmf.app.mediaName","<Media Name>",
"cmf.app.applicationID","<Application ID>",
"cmf.app.ecid":"<ECID>",
"cmf.app.userID":"<User ID>",
"cmf.app.appPageView":"<Set to 1>"])
```

Context Data Example:

```
ACPCore.trackState("xyz|home",data:[
"cmf.app.pageName":"xyz|home",
"cmf.app.pageLanguage":"en",
"cmf.app.userTimeZone":"America/New York",
"cmf.app.mediaName","xyz media abc",
"cmf.app.applicationID","1314.12112.134933",
"cmf.app.ecid":"24343535467474484748484 8",
"cmf.app.userID":"US1234",
"cmf.app.appPageView":"1"])
```

CREATE PROCESSING RULES

Create the following Processing Rules in for below variables to be set with corresponding context data values if corresponding context data exists:

Variable	Name	Context Data
pageName	Pages	N/A (argument of trackState call)
prop5	Language	cmf.app.pageLanguage
prop7	Time Zone	cmf.app.userTimeZone
prop21	Media Name	cmf.app.mediaName
eVar1	Application ID	cmf.app.applicationID
eVar7	Page Name	cmf.app.pageName
eVar21	Media Name	cmf.app.mediaName
eVar35	ECID	cmf.app.ecid
eVar36	User ID	cmf.app.userID
event34	App Page Views	cd.appPageView

MEDIA TRACKING

This section describes the data layer to be added on the certain actions when the Media is engaged by the user.

VARIABLES IN THIS SECTION

The below table consists of variables used for this solution.

Analytics Variable	Used For	Context Data Variable
eVar1	Application ID	cmf.app.applicationID
eVar7	Page Name	cmf.app.pageName
eVar24	Media (Video/ Audio) Name	cmf.app.mediaPlayedName
eVar25	Media Format	cmf.app.mediaFormat
eVar26	Media Type	cmf.app.mediaType
eVar27	Media Genre	cmf.app.mediaGenre
eVar36	User ID	cmf.app.userID
event24	Media Start	cmf.app.mediaStart
event25	Media Complete 25%	cmf.app.media25
event26	Media Complete 50%	cmf.app.media50
event27	Media Complete 75%	cmf.app.media75
event28	Media Complete 100%	cmf.app.mediaComplete
list1	Media Information	cmf.app.mediaInformation

DEPLOYMENT INSTRUCTIONS

CONTEXT DATA

Set the following context data variables on media engagements on play, 25% completion, 50% completion, 75% completion and 100% completion and trigger a 'trackAction' call:

Context Data Variable	Notes
cmf.app.applicationID	Set this variable to the application ID which will be provided by CMF. For example: "1314.12112.134933"
cmf.app.pageName	Set a clear, descriptive page name. Consider including the type or category of page to start the page name. Continue the page name with each level of the site hierarchy until the final descriptive portion of the page. Use a pipe "|" or ":" as the delimiter between each portion of the pageName.
cmf.app.mediaPlayedName	Set this variable with Media (audio/ video) played name. For example: "Education Guidelines"
cmf.app.mediaFormat	Set this variable with Media format. For example: Feature Film, Feature Length Documentary, Mini-Series, MOW, One-off, Pilot, Series etc.

cmf.app.mediaType	Set this variable with Media type. For example: Live Action, Animation, Both Live Action and Animation etc.
cmf.app.mediaGenre	Set this variable with Media genre. For example: Education, Kids, Thriller, Horror etc.
cmf.app.userID	Set this variable to a custom visitor ID if the user is logged in or if there is any other way to identify the user uniquely. For example: "US1234", "QA1234" etc. If there is no such ID, then set it as "NA"
cmf.app.mediaStart	Set this to 1 when media (audio/ video) is started
cmf.app.media25	Set this to 1 when media (audio/ video) completes 25%
cmf.app.media50	Set this to 1 when media (audio/ video) completes 50%
cmf.app.media75	Set this to 1 when media (audio/ video) completes 75%
cmf.app.mediaComplete	Set this to 1 when media (audio/ video) completes 100%
cmf.app.mediaInformation	Set this variable with Media information - season, episode number of the Media, episode name – concatenated by "|". For example: "S02|E11|Education Guidelines"

Context Data Format for media start:

```
ACPCore.trackAction("<Video/ Audio Start>",data:[
"cmf.app.applicationID":"<Set this to application ID>",
"cmf.app.pageName":"<Set this to page name>",
"cmf.app.mediaPlayedName":"<Set this to name of video/audio>",
"cmf.app.mediaFormat":"<Set this to media format>",
"cmf.app.mediaType":"<Set this to media type>",
"cmf.app.mediaGenre":"<Set this to media genre>",
"cmf.app.mediaInformation":"<Set this to media information>",
"cmf.app.userID":"<Set this to User ID>",
"cmf.app.mediaStart":"<Set this to 1>"
])
```

Context Data Example for media start:

```
ACPCore.trackAction("Video Start",data:[
"cmf.app.applicationID":"1314.12112.134933",
"cmf.app.pageName":"home:categories:xyz series:season1",
"cmf.app.mediaPlayedName":"Education Guidelines",
"cmf.app.mediaFormat":"Feature Length Documentary",
"cmf.app.mediaType":"Animation",
"cmf.app.mediaGenre":"<Education",
"cmf.app.mediaInformation":"S02|E11|Education Guidelines",
"cmf.app.userID":"US1234",
"cmf.app.mediaStart":"1"
])
```

Context Data Format for media complete 25%:

```
ACPCore.trackAction("<Video/ Audio Start>",data:[
"cmf.app.applicationID":"<Set this to application ID>",
"cmf.app.pageName":"<Set this to page name>",
"cmf.app.mediaPlayedName":"<Set this to name of video/audio>",
"cmf.app.mediaFormat":"<Set this to media format>",
"cmf.app.mediaType":"<Set this to media type>",
"cmf.app.mediaGenre":"<Set this to media genre>",
"cmf.app.mediaInformation":"<Set this to media information>",
"cmf.app.userID":"<Set this to User ID>",
"cmf.app.media25":"<Set this to 1>"
])
```

Context Data Example for media complete 25%:

```
ACPCore.trackAction("Video Start",data:[
"cmf.app.applicationID":"1314.12112.134933",
"cmf.app.pageName":"home:categories:xyz series:season1",
"cmf.app.mediaPlayedName":"Education Guidelines",
"cmf.app.mediaFormat":"Feature Length Documentary",
"cmf.app.mediaType":"Animation",
"cmf.app.mediaGenre":"<Education",
"cmf.app.mediaInformation":"S02|E11|Education Guidelines",
"cmf.app.userID":"US1234",
"cmf.app.media25":"1"
])
```

Context Data Format for media complete 50%:

```
ACPCore.trackAction("<Video/ Audio Start>",data:[
"cmf.app.applicationID":"<Set this to application ID>",
"cmf.app.pageName":"<Set this to page name>",
"cmf.app.mediaPlayedName":"<Set this to name of video/audio>",
"cmf.app.mediaFormat":"<Set this to media format>",
"cmf.app.mediaType":"<Set this to media type>",
"cmf.app.mediaGenre":"<Set this to media genre>",
"cmf.app.mediaInformation":"<Set this to media information>",
"cmf.app.userID":"<Set this to User ID>",
"cmf.app.media50":"<Set this to 1>"
])
```

Context Data Example for media complete 50%:

```
ACPCore.trackAction("Video Start",data:[
"cmf.app.applicationID":"1314.12112.134933",
"cmf.app.pageName":"home:categories:xyz series:season1",
"cmf.app.mediaPlayedName":"Education Guidelines",
"cmf.app.mediaFormat":"Feature Length Documentary",
"cmf.app.mediaType":"Animation",
"cmf.app.mediaGenre":"<Education",
"cmf.app.mediaInformation":"S02|E11|Education Guidelines",
"cmf.app.userID":"US1234",
"cmf.app.media50":"1"
])
```

Context Data Format for media complete 75%:

```
ACPCore.trackAction("<Video/ Audio Start>",data:[
"cmf.app.applicationID":"<Set this to application ID>",
"cmf.app.pageName":"<Set this to page name>",
"cmf.app.mediaPlayedName":"<Set this to name of video/audio>",
"cmf.app.mediaFormat":"<Set this to media format>",
"cmf.app.mediaType":"<Set this to media type>",
"cmf.app.mediaGenre":"<Set this to media genre>",
"cmf.app.mediaInformation":"<Set this to media information>",
"cmf.app.userID":"<Set this to User ID>",
"cmf.app.media75":"<Set this to 1>"
])
```

Context Data Example for media complete 75%:

```
ACPCore.trackAction("Video Start",data:[
"cmf.app.applicationID":"1314.12112.134933",
"cmf.app.pageName":"home:categories:xyz series:season1",
"cmf.app.mediaPlayedName":"Education Guidelines",
"cmf.app.mediaFormat":"Feature Length Documentary",
"cmf.app.mediaType":"Animation",
"cmf.app.mediaGenre":"<Education",
"cmf.app.mediaInformation":"S02|E11|Education Guidelines",
"cmf.app.userID":"US1234",
"cmf.app.media75":"1"
])
```

Context Data Format for media complete 100%:

```
ACPCore.trackAction("<Video/ Audio Start>",data:[
"cmf.app.applicationID":"<Set this to application ID>",
"cmf.app.pageName":"<Set this to page name>",
"cmf.app.mediaPlayedName":"<Set this to name of video/audio>",
"cmf.app.mediaFormat":"<Set this to media format>",
"cmf.app.mediaType":"<Set this to media type>",
"cmf.app.mediaGenre":"<Set this to media genre>",
"cmf.app.mediaInformation":"<Set this to media information>",
"cmf.app.userID":"<Set this to User ID>",
"cmf.app.mediaComplete":"<Set this to 1>"
])
```

Context Data Example for media complete 100%:

```
ACPCore.trackAction("Video Start",data:[
"cmf.app.applicationID":"1314.12112.134933",
"cmf.app.pageName":"home:categories:xyz series:season1",
"cmf.app.mediaPlayedName":"Education Guidelines",
"cmf.app.mediaFormat":"Feature Length Documentary",
"cmf.app.mediaType":"Animation",
"cmf.app.mediaGenre":"<Education",
"cmf.app.mediaInformation":"S02|E11|Education Guidelines",
"cmf.app.userID":"US1234",
"cmf.app.mediaComplete":"1"
])
```

CREATE PROCESSING RULES

Create the following Processing Rules in for below variables to be set with corresponding context data values if corresponding context data exists:

Variable	Name	Context Data
eVar1	Application ID	cmf.app.applicationID
eVar7	Page Name	cmf.app.pageName
eVar24	Media (Video/ Audio) Name	cmf.app.mediaPlayedName
eVar25	Media Format	cmf.app.mediaFormat
eVar26	Media Type	cmf.app.mediaType
eVar27	Media Genre	cmf.app.mediaGenre
eVar36	User ID	cmf.app.userID
event24	Media Start	cmf.app.mediaStart
event25	Media Complete 25%	cmf.app.media25
event26	Media Complete 50%	cmf.app.media50
event27	Media Complete 75%	cmf.app.media75
event28	Media Complete 100%	cmf.app.mediaComplete
list1	Media Information	cmf.app.mediaInformation

SHOPPING CART

This solution allows the clients to evaluate the effectiveness of the shopping cart.

VARIABLES IN THIS SECTION

The below table consists of variables used for this solution.

Analytics Variable	Used For	Context Data Variable
pageName	Pages	N/A (argument of trackState call)
eVar1	Application ID	cmf.app.applicationID
eVar7	Page Name	cmf.app.pageName
products	Products	Set directly in Products Syntax
purchaseID	Deduplication of Purchases	cmf.app.purchaseID
eVar34	Order ID	cmf.app.purchaseID
evar35	ECID	cmf.app.ecid
eVar36	User ID	cmf.app.userID
purchase	Orders	Set directly in Products Syntax
purchase	Units	Set directly in Products Syntax
purchase	Revenue	Set directly in Products Syntax
event1	Total Revenue	cmf.app.orderRevenue

DEPLOYMENT INSTRUCTIONS

CONTEXT DATA

Set the following context data variables on order confirmation and trigger a 'trackAction' call:

Context Data Variable	Notes
cmf.app.applicationID	Set this variable to the application ID which will be provided by CMF. For example: "1314.12112.134933"
cmf.app.pageName	Set a clear, descriptive page name. Consider including the type or category of page to start the page name. Continue the page name with each level of the site hierarchy until the final descriptive portion of the page. Use a pipe "|" or ":" as the delimiter between each portion of the pageName.
cmf.app.purchaseID	Set the value to the unique order ID for the order (such as the user's confirmation number). The order ID is alphanumeric with a limit of 20 characters, which must be unique for the rest of the life of the report suite
cmf.app.userID	Set this variable to a custom visitor ID if the user is logged in or if there is any other way to identify the user uniquely. For example: "US1234", "QA1234" etc. If there is no such ID, then set it as "NA"

cmf.app.orderRevenue	This should reflect total revenue for the order (before shipping, discount and taxes)
cmf.app.ecid	Set this variable to the Experience Cloud ID set by the app. Use the "getExperienceCloudId" API as explained below to populate this variable.
&&products	It's a special products string which contain multiple information. The format of string is "Category;Product;Quantity;Price[,Category;Product;Quantity;Price]". Category represents the product category. Product represent Product SKU – Product IDs should be unique identifiers of individual products, such as a product SKU. Quantity represents the total number of units purchased for that specific product. Price - This should reflect the revenue (unit price) for that item (before shipping, discount and taxes).

Context Data Format for media start:

```
ACPCore.trackAction("<Order Confirmed>",data:[
"cmf.app.pageName":"<Page Name>",
"cmf.app.applicationID","<Application ID>",
"cmf.app.ecid":"<ECID>",
"cmf.app.userID":"<User ID>",
"&&products":"<Category;Product;Quantity;Price[,Category;Product;Quantity;Price]>",
"cmf.app.purchaseID":"<Purchase ID>",
"cmf.app.orderRevenue":"<Order Revenue>"
])
```

Context Data Example for order confirmation:

```
ACPCore.trackAction("Order Confirmed",data:[
"cmf.app.pageName":"order confirmed",
"cmf.app.applicationID","1314.12112.134933",
"cmf.app.ecid":"24343535467474484748484 8",
"cmf.app.userID":"us1234",
"&&products":";XYZ-100087613;1;69.95,;XYZ-53215652;10;29.99",
"cmf.app.purchaseID":"1234567890",
"cmf.app.orderRevenue":"99.94"
])
```

CREATE PROCESSING RULES

Create the following Processing Rules in for below variables to be set with corresponding context data values if corresponding context data exists:

Variable	Name	Context Data
pageName	Pages	N/A (argument of trackState call)
eVar1	Application ID	cmf.app.applicationID
eVar7	Page Name	cmf.app.pageName
products	Products	Set directly in Products Syntax
purchaseID	Deduplication of Purchases	cmf.app.purchaseID
eVar34	Order ID	cmf.app.purchaseID
evar35	ECID	cmf.app.ecid
eVar36	User ID	cmf.app.userID

purchase	Orders	Set directly in Products Syntax
purchase	Units	Set directly in Products Syntax
purchase	Revenue	Set directly in Products Syntax
event1	Total Revenue	cmf.app.orderRevenue

APPENDIX: VALIDATION

DEBUGGING AND TESTING TOOLS

Adobe Consulting uses a variety of tools to unit test Adobe Analytics code on any given mobile app. We recommend that clients should use at proxying tool as well, like Charles:

- Charles Web Debugging Proxy [http://www.charlesproxy.com]

To test the Mobile SDK implementation, it is recommended to install the mobile app on a mobile device and test Marketing Cloud functionality in real time while performing actions within the app. In order to do this, you must use a proxy monitoring tool and proxy the device to your computer via a proxy.
The following steps indicate how to proxy an iOS device to your machine via Charles Proxy monitoring tool:*
Notes: the mobile device must be on the same network as the computer running the proxy monitoring tool. Be aware that some corporate networks may block the use of such proxying tools. In order to ensure comprehensive testing, it is recommended that SSL is disabled in the ADBMobileConfig.JSON file for your pre-production builds.

- Navigate to Settings > Wi-Fi.
- Click the blue "i" icon to the right of your currently selected Wi-Fi network.
- Scroll to the HTTP Proxy settings.
- Select Manual.
- Enter the IP Address that your laptop/workstation is currently using for Wi-Fi connectivity (in Charles, this is found by clicking Help > Local IP Address)
- Click Home.
- Launch the application and verify that hits from the mobile device are currently appearing in Charles Proxy. The first time you proxy a device, you will be shown a popup in Charles asking to accept the incoming connection. Click yes.