Open Measurement SDK

OMID API

Version 1.2 | June 2017

Executive Summary

The Open Measurement Software Development Kit (OM SDK) is designed to facilitate third party viewability and verification measurement for ads served to mobile app environments without requiring multiple Ad Verification Service Providers (Measurement Provider) Software Development Kits (SDKs).

The OM SDK consists of a native library for iOS & Android operating systems (OS) as well as a JavaScript API, named Open Measurement Interface Definition (OMID). This document covers the details of the OMID API.

OMID is an API that enables standard communication of OM SDK data to measurement tags from Measurement Providers used to access information about the state of an advertisement and the environment it’s being served into.

App developers or their Advertising Software Development Kit (Ad SDK) providers must integrate the OM SDK and implement the OM SDK Javascript provided with the OM SDK to ensure that this communication may occur.

OM SDK is developed and managed by the Open Measurement Working Group (OMWG). More informaiton about OMWG is aviable here: https://iabtechlab.com/working-groups/open-measurement-working-group/

Audience

This API document is designed for for Ad SDK developers, App publishers and Measurement Providers to understand the API details

More information on OM SDK available at: https://www.iabtechlab.com/OM SDK

Change Log

Date	Version	Changes
4/10/2018	1.1	Initial Release
6/14/2018	1.2	Added OMID for Web Video support. **General Changes:** Throughout the API spec, documentation has been rewritten or clarified or provided where it was otherwise missing. Usage examples were also added. In general, except where listed below, these should reflect already existing behavior of OMSDK. **Context Object Changes:** The Context object is the only major source of additions or modifications. **API Version:** The apiVersion property was added to the top-level context object to reflect the version of the Verification Client API that has been implemented. **Environment:** The environment property has a new valid value ("web") to support the web case. **Access Mode:** The accessMode property was added to the top-level context object to reflect whether sandboxing is enabled or not ("limited" vs "full" accessMode respectively). **Video/Slot Element:** The videoElement and slotElement properties were added to the top-level context object in order to accomodate passing the rendering elements in non-sandbox mode. These are only provided for non-sandbox mode. **Measuring Video/Slot Element:** The measuringVideoElement and measuringSlotElement properties were added to the top-level context object. **VAST 4.1 Values:** The adServingId, transactionId, podSequence, and adCount properties were added to the top-level context object in order reflect newly defined values from VAST 4.1. These are potentially valuable for tracking purposes and will be available as macro fields in VAST 4.1 documents and so should be made available from OMID in order to prevent an information gap. **OMID Implementer:** The omidImplementer property has been added to the omidJsInfo subobject of the context. This is meant both to provide a clear place to distinguish between OMSDK and non-OMSDK implementations, as well as to provide identification of non-OMSDK providers. **Supports Array:** The documentation of the supports property has removed the "vlid" value in order to reflect the fact that "video lifecycle interface" is in fact not an optional part of the API (the intention of this property is to mark the availability of optional features). The "clid" value is maintained and documented, as the "container lifecycle interface" (e.g. geometryChange event) is considered optional when using non-sandbox mode. For example, a third-party web implementer would not be expected to perform geometry calculations if it was providing

		direct access to rendering elements. Since OMSDK always provides this, it would continue to provide the "clid" value in supports. The "vlid" could be removed in the future, but this is not strictly necessary for this API document.

About IAB Tech Lab

The IAB Technology Laboratory is an independent, international, research and development consortium charged with producing and helping companies implement global industry technical standards. Comprised of digital publishers and ad technology firms, as well as marketers, agencies, and other companies with interests in the interactive marketing arena, the IAB Tech Lab's goal is to reduce friction associated with the digital advertising and marketing supply chain, while contributing to the safe and secure growth of the industry.

Learn more about IAB Tech Lab here: https://www.iabtechlab.com/

Open Measurement Working Group

Commit Group Members

Comscore	IAB Tech Lab
DoubleVerify	Moat
Google	Pandora
Integral Ad Science	Microsoft

Working Group Members

AdColony	eBay	Jun Group	Rubicon Project
Adform	Extreme Reach	Kochava Inc.	SITO Mobile
AerServ	Flipboard	Lucidity	Smaato
AppNexus	FreeWheel	Miaozhen Information Consultancy Co., Ltd	SpringServe
BARC India	Fyber	Microsoft Advertising	Tapjoy

Burt	Gameloft	NBCUniversal	Teads
Coalition for Innovative Media Measurement (CIMM)	Google	Nielsen	TenMax
comScore	InMobi	Oath	TripleLift
Cyber Communications Inc.	Innity	OpenX	Verve
Dailymotion	Innovid	Oracle's Moat	Vidooly
Display.io	Integral Ad Science	Pandora	Weborama
DoubleVerify	Intowow	RTBAsia	White Ops
			YOSPACE

* Working Group membership as of June 7, 2018

Learn more about Open Measurement Working Group here (https://iabtechlab.com/working-groups/open-measurement-working-group/)

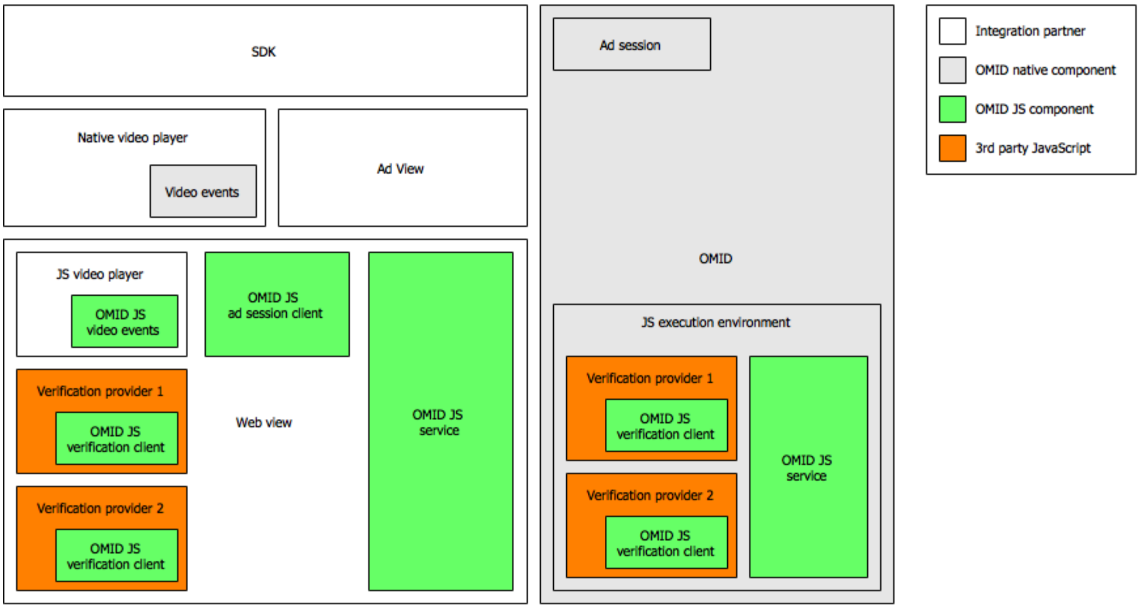
Table of Contents

Introduction

The following documents the new OMID API, which is implemented in the OM SDK. The term “integration partner” has been used throughout this document to cater for both Ad SDK and publisher that implement direct OM SDK integrations.

OM SDK components

The diagram below shows the various Open Measurement SDK components and where each component has been designed to be integrated;



OM SDK supports two ad session types; HTML and native. When creating an HTML ad session OM SDK expects all JS components to be executed within the web view, but for native ad sessions OM SDK will create a JS execution environment enabling verification provider script execution.

Ad session

Central to the OMID API is the ad session which enables the integration partner to manage its lifecycle. This API has been designed to support a number of integration scenarios - these include;

1. HTML display ad sessions where the integration partner manages the lifecycle from the native SDK. This requires that the native SDK will be responsible for starting/finishing the ad session as well as recording the impression event.
2. HTML display ad sessions where the integration partner uses a combination of native SDK and JS SDK to manage the lifecycle. This still requires that the native SDK will be responsible for starting/finishing the ad session but the JS SDK would contribute the ad session by triggering the ad impression event.

3. Native display ad sessions where the integration partner manages the lifecycle from the native SDK. Because this ad session type does not rely on web views for rendering OM SDK creates a JavaScript execution environment for communicating events to all verification providers.
4. HTML video ad sessions where the integration partner uses a combination of native SDK and JS SDK to manage the lifecycle which is very similar to the display scenario mentioned above. Because the HTML video ad session is designed to run with a HTML5 video player this scenario expects the JS SDK to interact with the JS video event API for communicating playback state.
5. Native video ad sessions where the integration partner manages the lifecycle from the native SDK which is very similar to the display scenario mentioned above. This video scenario also requires the native SDK to use the video event API for communicating playback state.

All ad session scenarios mentioned above support two registration API methods; one for ad view registration which enables the native SDK to notify OMID which view should be considered for viewability and another API for friendly obstructions which OMID will exclude from all viewability calculations.

For any integration partners wishing to use the OMID JS ad session API, this has been designed to be shared as source. Each JavaScript ad SDK will include OMID JS ad session client code within their existing script and minified using their existing processes.

OM SDK JS data service

Because OM SDK is designed to support both native only and native + JS ad sessions we have introduced a central OM SDK JS data service which collects all events from both ad session providers and is then responsible for notifying all registered OMID JS clients of any ad session / state changes.

The OM SDK JS data service also provides a detection mechanism which the OMID JS client will use so verification providers can apply the correct measurement strategy.

For HTML ad sessions it is important that integration partners ensure OMID JS data service has been injected prior to starting any ad session and loading verification provider scripts. For native ad sessions we require the integration partner to provide the downloaded OMID JS service content when creating any new ad session context.

OMID JS verification client

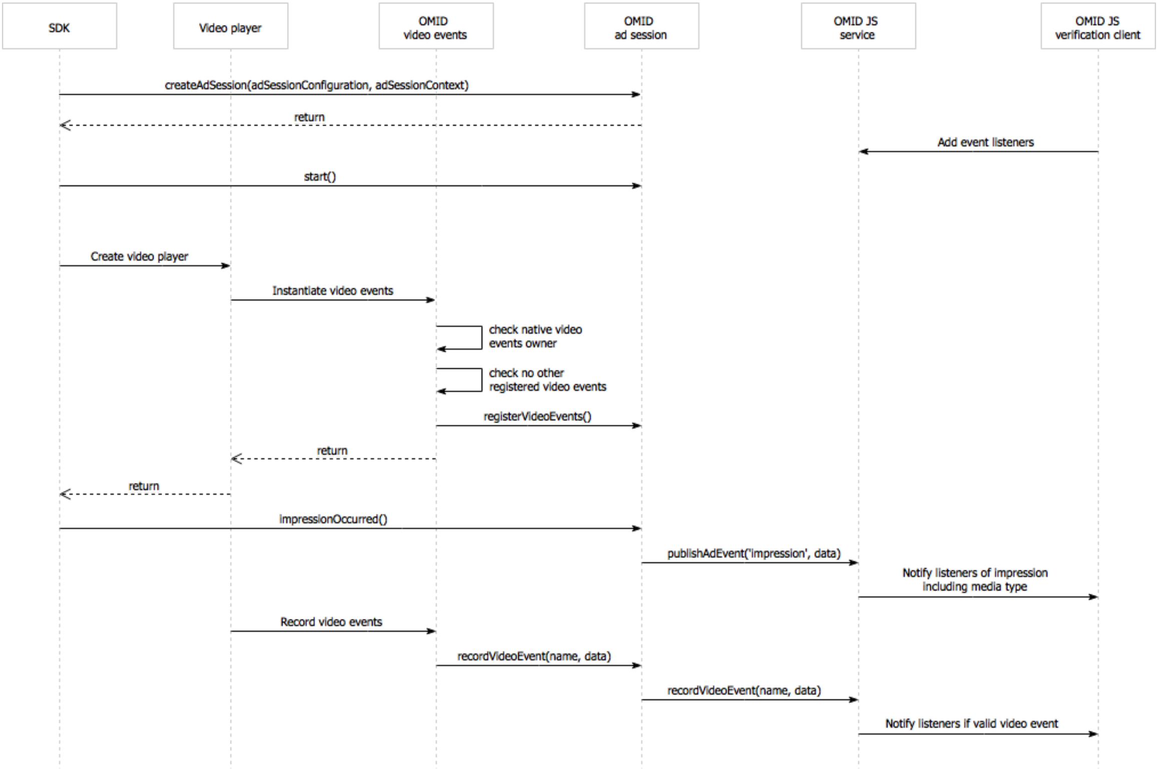
The OMID JS verification client is a utility that interfaces directly with the OMID JS data service both for detection and subscribing to ad session events. This verification client will also handle communication for both friendly and unfriendly placements (i.e. cross-domain iFrames). We recommend that all clients use this API when interested in OMID events.

We have designed the OMID JS verification client to be shared as source and for each verification provider to include this within their existing script and minified using their existing processes.

Video events

Because of the number of video player implementations available across the advertising ecosystem as well as challenges where JS video players may not have direct access to the top level window (i.e. cross-domain iFrames) we have introduced support for video event implementations in OM SDK. Each video player can select the most appropriate video event implementation and this will assume responsibility for publishing video events to the OM SDK JS data service.

Once the event has been received by the OM SDK JS data service these will then be shared with all registered OMID JS clients - see above section for more details. See the below sequence diagram for more information;


SDK
Video player
OMID video events
OMID ad session
OMID JS service
OMID JS verification client
createAdSession(adSessionConfiguration, adSessionContext)
return
Add event listeners
start()
Create video player
Instantiate video events
check native video events owner
check no other registered video events
registerVideoEvents()
return
return
impressionOccurred()
publishAdEvent('impression', data)
Notify listeners of impression including media type
Record video events
recordVideoEvent(name, data)
recordVideoEvent(name, data)
Notify listeners if valid video event

For any integration partners wishing to use the OMID JS video events API this has been designed to be shared as source and for each HTML5 video player to include this within their existing script and minified using their existing processes.

OM SDK namespace builds

The OM SDK build process supports both namespace and generic OM SDK library builds. The generic builds use the class and package names described in this document.

Namespaced builds rename the classes and package names to allow ad SDK integrators to include OM SDK in their SDKs without conflicting with other Ad SDKs. Ad SDKs and Apps must use the namespaced version of OM SDK.

OM SDK JS service injection strategies

For HTML ad sessions each integration partner is responsible for ensuring that the OM SDK JS service has been injected into the webview prior to any additional JS components.

For native ad sessions each integration partner is expected to download and provide the OM SDK JS service content when creating new ad session contexts. Any attempt to create an ad session without a valid OM SDK JS service will result in an error.

Below we have detailed some possible solutions to OM SDK JS service injection for HTML ad sessions.

Server-side OM SDK JS service script content injection

This injection strategy relies on the ad server being responsible for downloading the OM SDK JS service script content and modifying the original HTML ad response.

The following outlines the steps required to support this injection strategy;

1. Ad server requests and caches the OM SDK JS service script content.
2. Integration partner creates new OMID ad session.
3. OMID enabled ad request is received by the ad server.
4. Ad server modifies the HTML ad response to prepend OM SDK JS service script content - for example; `<script>downloaded/cached OM SDK JS service script content</script><<ORIGINAL TAG HTML CONTENT>>`.
5. Integration partner receives HTML ad response and injects content into the registered web view.
6. Integration partner notifies OM SDK that the ad session can be started.

This solution assumes that the ad server will take responsibility for ensuring that OM SDK JS service script content is correctly injected into the HTML ad response across the variety of supported tags.

Please note that this is the recommended OM SDK JS service injection solution as this provides the most flexibility when it comes to updating any injection rules without impacting the client integration.

Client-side OM SDK JS service script content injection

This injection strategy relies on the integration partner assuming responsibility for downloading and caching the OM SDK JS service from their CDN. Once available the integration can choose between using the OMID script injection API or implement their own injection strategy using the downloaded script content.

The following outlines the steps required to support this injection strategy;

1. Integration partner SDK will download / cache their AVID JS service resource.
2. Integration partner creates new OMID ad session.
3. OMID enabled ad request is received by the ad server and unmodified ad response sent back to integration partner.
4. If OM SDK JS service download is complete then use the OMID script injection API to modify HTML ad content. If OM SDK JS service download is not yet complete then wait for download callback.
5. Integration partner injects the modified content into the registered web view.
6. Integration partner notifies OMID that the ad session can be started.

When it comes to using the OMID script injection API the following rules will apply;

- If the HTML ad response contains no `<html>`, `<head>` or `<body>` then the script content will be prepend to the HTML.
- If the HTML ad response contains a `<html>` element with no `<head>`, but a `<body>` element then the script content will be added as the first child element of the `<body>`.
- If the HTML ad response contains a `<html>` element with both a `<head>` and `<body>` elements then the script content will be added as the first child element of the `<head>`.
- If the HTML ad response contains a `<html>` element with no `<head>` or `<body>` elements then the script content will be added as the first child element of the `<html>`.
- If the HTML ad response contains a `<!DOCTYPE html>` element with no <html>, `<head>` or `<body>` elements then the script content will be added as the first child element of the `<!DOCTYPE html>`.

This implementation will also cater for situations where any element has been commented out - for example, `<html><!-- <head></head> --><body></body></html>`. In this example the script content will be added as the first child element of the `<body>`.

The OMID script injector will also be able to handle self-closing tags - for example; `<html><head/><body>...</body></html>` will be converted into `<html><head>script</head><body>...</body></html>`.

Ad session lifecycle

As highlighted above the OMID API is designed to support a variety of integration styles. The diagrams below cover these off in more detail explain how the OMID API should be used in each scenario.

Note that creating an OMID ad session in the native layer sends a message to the OM SDK JS Service running in the webview. If the OM SDK JS Service has not loaded before the ad session is created, the message is lost, and the verification scripts will not receive any events. To prevent this, the implementation must wait until the webview finishes loading OM SDK JS before starting the OMID ad session.

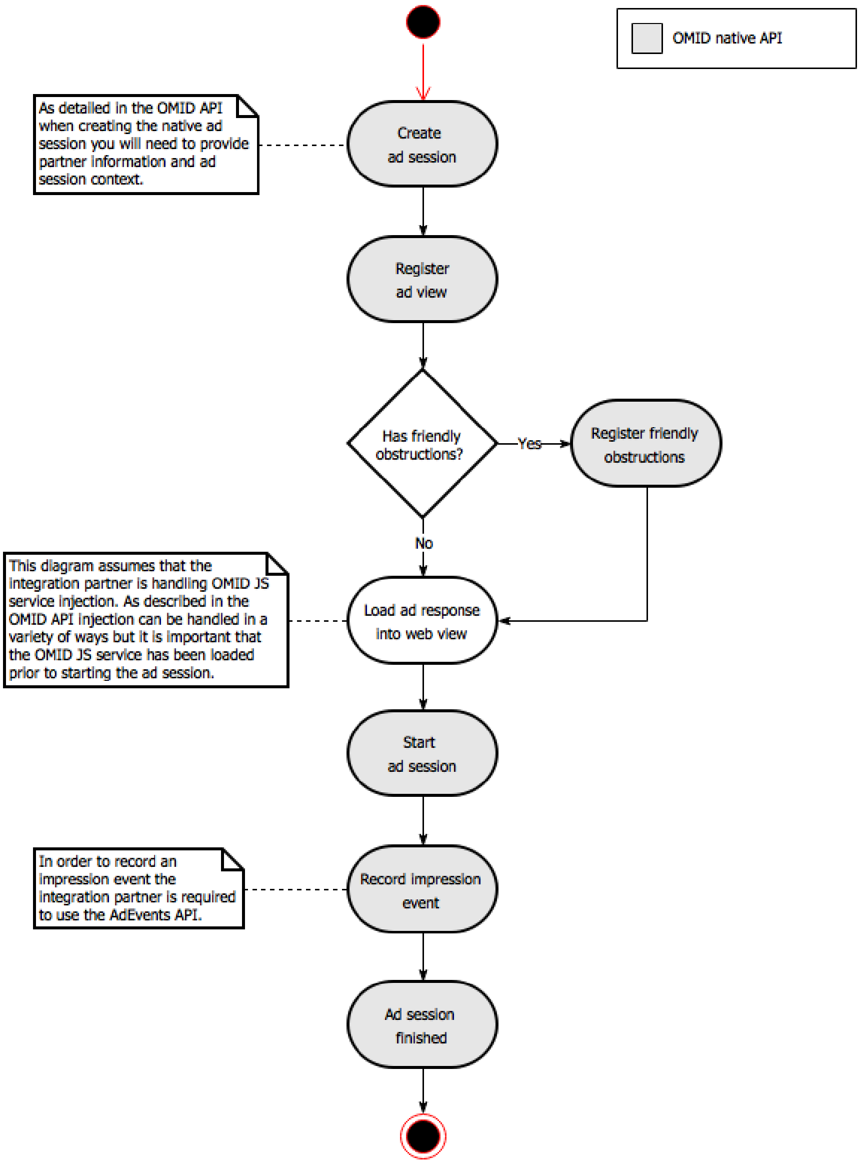
Also note that ending an OMID ad session sends a message to the verification scripts running inside the webview supplied by the integration. So that the verification scripts have enough time to handle the “sessionFinish” event, the integration must maintain a strong reference to the webview for at least 1.0 seconds after ending the session.

In Android, for all ad sessions that are created, finish must be called once the ad session is no longer required, otherwise memory will be leaked.

In iOS, for all ad sessions that are started, finish must be called once the ad session is no longer required, otherwise memory will be leaked.

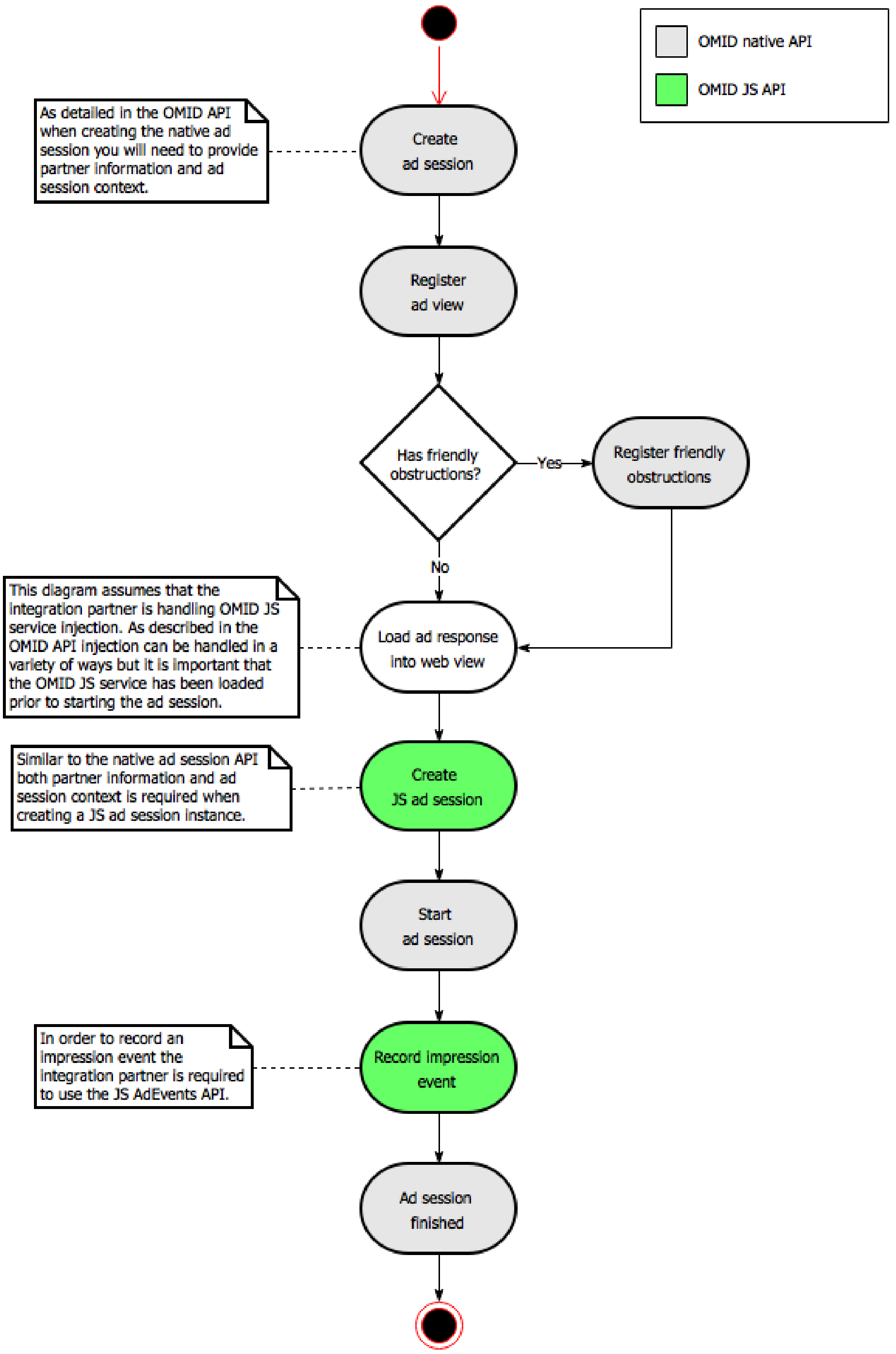
Display ad session with no contributing JS ad session

The below diagram demonstrates the OMID display ad session lifecycle where the integration partner wishes to only use the full native OMID API.


OMID native API
As detailed in the OMID API when creating the native ad session you will need to provide partner information and ad session context.
Create ad session
Register ad view
Has friendly obstructions?
Yes
Register friendly obstructions
No
This diagram assumes that the integration partner is handling OMID JS service injection. As described in the OMID API injection can be handled in a variety of ways but it is important that the OMID JS service has been loaded prior to starting the ad session.
Load ad response into web view
Start ad session
In order to record an impression event the integration partner is required to use the AdEvents API.
Record impression event
Ad session finished

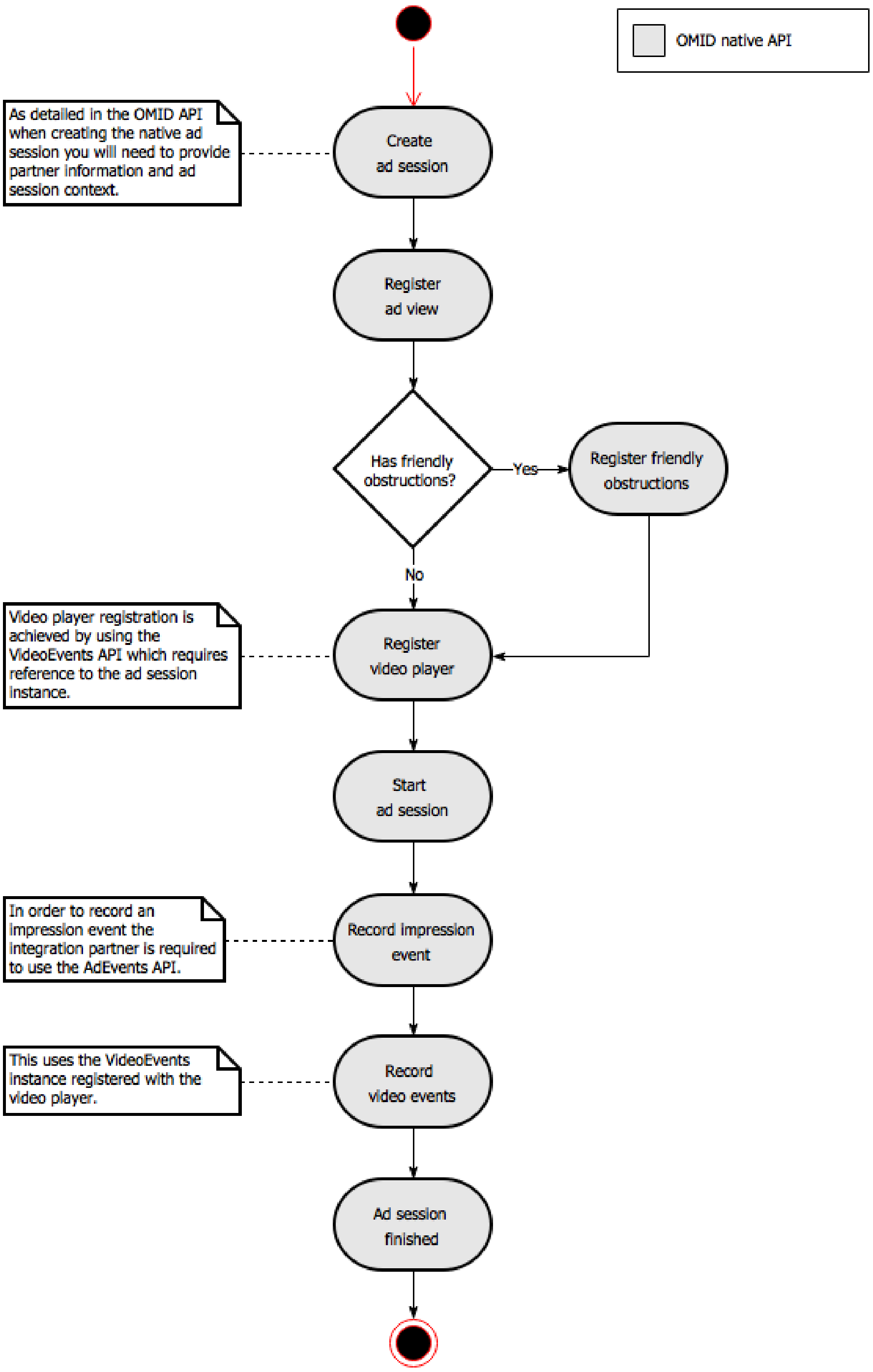
Display ad session with a contributing JS ad session

The below diagram demonstrates the OMID display ad session lifecycle where the integration partner wishes to use both the native and JS OMID API.


OMID native API
OMID JS API
As detailed in the OMID API when creating the native ad session you will need to provide partner information and ad session context.
Create ad session
Register ad view
Has friendly obstructions?
Yes
Register friendly obstructions
No
This diagram assumes that the integration partner is handling OMID JS service injection. As described in the OMID API injection can be handled in a variety of ways but it is important that the OMID JS service has been loaded prior to starting the ad session.
Load ad response into web view
Similar to the native ad session API both partner information and ad session context is required when creating a JS ad session instance.
Create JS ad session
Start ad session
In order to record an impression event the integration partner is required to use the JS AdEvents API.
Record impression event
Ad session finished

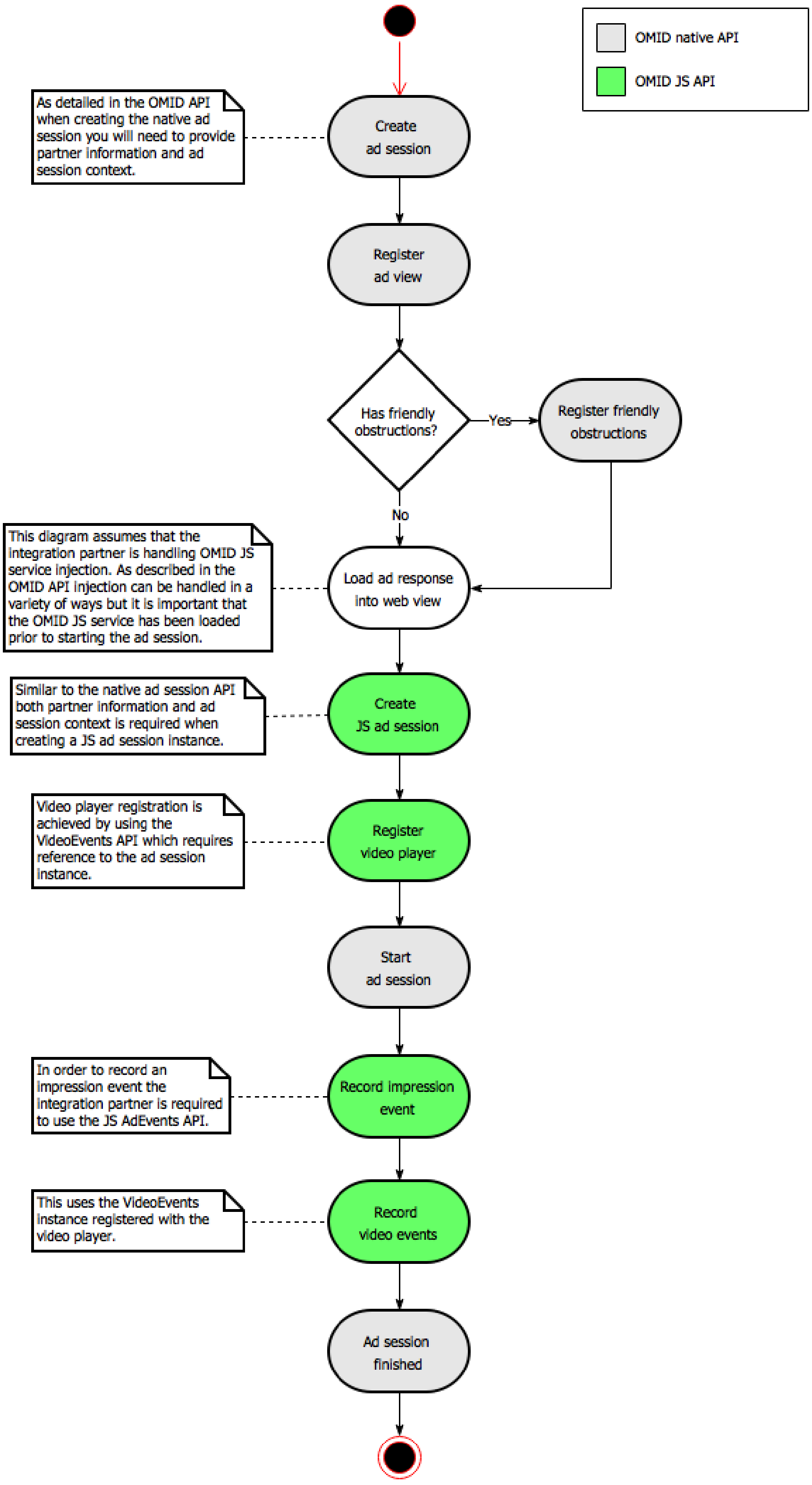
Video ad session with native video player

The below diagram demonstrates the OMID video ad session lifecycle using a native video player.


OMID native API
As detailed in the OMID API when creating the native ad session you will need to provide partner information and ad session context.
Create ad session
Register ad view
Has friendly obstructions?
Yes
Register friendly obstructions
No
Video player registration is achieved by using the VideoEvents API which requires reference to the ad session instance.
Register video player
Start ad session
In order to record an impression event the integration partner is required to use the AdEvents API.
Record impression event
This uses the VideoEvents instance registered with the video player.
Record video events
Ad session finished

Video ad session with HTML video player

The below diagram demonstrates the OMID video ad session lifecycle using a HTML video player.


OMID native API
OMID JS API
As detailed in the OMID API when creating the native ad session you will need to provide partner information and ad session context.
Create ad session
Register ad view
Has friendly obstructions?
Yes
Register friendly obstructions
No
This diagram assumes that the integration partner is handling OMID JS service injection. As described in the OMID API injection can be handled in a variety of ways but it is important that the OMID JS service has been loaded prior to starting the ad session.
Load ad response into web view
Similar to the native ad session API both partner information and ad session context is required when creating a JS ad session instance.
Create JS ad session
Video player registration is achieved by using the VideoEvents API which requires reference to the ad session instance.
Register video player
Start ad session
In order to record an impression event the integration partner is required to use the JS AdEvents API.
Record impression event
This uses the VideoEvents instance registered with the video player.
Record video events
Ad session finished

Supporting verification script resources in VAST

Unlike HTML ad formats where all verification clients will be loaded using the more traditional `<script src=”...”></script>` HTML element for video ad formats we will be using VAST XML ad responses as detailed below;

VAST version 4.1 support via ad verifications node

Below is an example of how to include verification information VAST 4.1 tags. Please refer to VAST 4.1 specification for exact usage of different parameters in <AdVerifications> node.
.

VAST version 4.1 OMID example

```
<AdVerifications>
      <Verification vendor=”company.com-omid”>
        <JavaScriptResource apiFramework="omid" browserOptional=”true”>
                    <![CDATA[https://verification.com/omid_verification.js]]>
        </JavaScriptResource>
        <TrackingEvents>
          <Tracking event="verificationNotExecuted">
             <![CDATA[https://verification.com/trackingur/[REASON]l]]>
              </Tracking>
       </TrackingEvents>
             <VerificationParameters>
             <![CDATA[verification params key value pairs]]>
           </VerificationParameters>
      </Verification>
    </AdVerifications>
```

Pre-VAST version 4.1 support via a custom extension

For older versions of VAST documents namely VAST 2.0, VAST 3.0 or VAST 4.0, verification code should be loaded via Extensions node specifying ‘Extension type’ as ‘AdVerifications’. The root element is ‘AdVerifications’ node with the same schema as the VAST 4.1 ‘AdVerifications’ node.

Pre-VAST version 4.1 OMID example

```
<Extensions>
  <Extension type="AdVerifications">
    <AdVerifications>
      <Verification vendor=”company.com-omid”>
        <JavaScriptResource apiFramework="omid" browserOptional="true">
                    <![CDATA[https://verification.com/omid_verification.js]]>
        </JavaScriptResource>
            <TrackingEvents>
          <Tracking event="verificationNotExecuted">
             <![CDATA[https://verification.com/trackingurl]]>
              </Tracking>
       </TrackingEvents>
       <VerificationParameters>
             <![CDATA[verification params key value pairs]]>
        </VerificationParameters>
      </Verification>
    </AdVerifications>
  </Extension>
</Extensions>
```

Android OMID library API

Usage

Check for OMID compatibility and library activation

1. Verify that `Omid` class exists *(this is important only when the integration partner is using a shared OMID library)*.
2. Execute `Omid.isCompatibleWithOmidApiVersion` and ensure that `true` is returned.
    a. if `false` is returned you must avoid using the integrated OMID library as this will produce errors.
3. Check if OMID has already been activated by calling `Omid.isActive`.
    a. if OMID has not been activated then you will need to execute `Omid.activateWithOmidApiVersion`.

Load and inject OM SDK JS script content into tag response **(optional)**

1. Each integration partner is responsible for downloading and caching the OM SDK JS service ready for use in the OMID ad session.
2. Once the OM SDK script content has been downloaded then OMID JS injection can be performed calling `ScriptInjector.injectScriptContentIntoHtml`.

Using the OMID ad session API

1. Create a new `Partner` object.
2. Create a new `Context` object specifying the `Partner` and either a web view instance or a list of verification script resources.
3. Create a new `AdSession` object specifying the `Context`.
4. Once ready start the ad session executing `AdSession.start`.
5. Once started you can now record ad session events - see ad events and video events detailed below.
6. All ad session errors should be recorded calling `AdSession.error`.
7. Once the ad session has finished execute `AdSession.finish`.

Handling ad session ad events (display and video)

1. Create `AdEvents` object specifying the `AdSession` instance.
2. Notify the ad session when an impression has occurred by calling `AdEvents.impressionOccured`. This step can be ignored if the JS ad session controls when the impression event should be triggered.

Handling ad session video events (video only)

For HTML video ad sessions this will be handled by the JS ad session.

1. Create `VideoEvents` object specifying the `AdSession` instance.
2. Update video player implementation to trigger the appropriate video events during content loading / playback.

Thread Safety

OMID library functions must be called only from the main UI thread of the application. This is the same rule that the Android and iOS UI frameworks require.

API

com.iab.omid.library.Omid

```
package com.iab.omid.library;

/**
* This application level class will be called by all integration partners to ensure OMID
has <br>
* been activated before calling any other API methods. <br>
* Any attempt to use other API methods prior to activation will result in an exception.
* <p>
*/

public final class Omid {

   /**
      * Allows the integration partner to check that they are compatible with the running
OMID
     * library version.
     *
     * @param version of OMID library integrated by the partner.
      * @return true if the version supplied is compatible with the integrated OMID library
version.
      * @throws IllegalArgumentException if the supplied version parameter is null, blank or
an
     * invalid semantic version number.
     */
   public static boolean isCompatibleWithOmidApiVersion(final String version);

   /**
     * Access the running OMID library semantic version.
     *
     * @return the current semantic version of the integrated OMID library.
     */
   public static String getVersion();

   /**
      * Used to determine if the running OMID library is active before attempting to create
any ad
     * sessions.
     *
```

```
    * @return true if the OMID library has already been activated.
    */
   public static boolean isActive();

   /**
      * Enables the integration partner to activate OMID prior to calling any other API
methods.
    *
    * @param version            of OMID library integrated by the partner.
    * @param applicationContext of the running application.
     * @return true if activation was successful when checking the supplied version number
for
    * compatibility.
     * @throws IllegalArgumentException if the supplied version parameter is null, blank or
an
    * invalid semantic version number.
    * @throws IllegalArgumentException if the supplied application context is null.
    */
   public static boolean activateWithOmidApiVersion(final String version,
                                                    final Context applicationContext);
}
```

com.iab.omid.library.ScriptInjector

```
package com.iab.omid.library;

/**
* Utility class which enables integration partners to use a standard approach for injecting
OMID JS
* into the served tag HTML content.
* <p>
*/
public final class ScriptInjector {

   /**
    * Injects the supplied OMID JS content into the served HTML.
    *
     * @param scriptContent containing the OMID JS service content to be injected into the
hidden
    *                      tracking web view.
      * @param html             of the tag content which should be modified to include the
downloaded
    *                      OMID JS content.
    * @return modified HTML to include the supplied OMID JS service.
    * @throws IllegalArgumentException if the supplied HTML is either null or blank.
      * @throws IllegalStateException     if this method has been executed before OMID has
been
    * activated.
     * @throws IllegalStateException     if this method has been executed before OMID JS is
available.
    */
     public static String injectScriptContentIntoHtml(final String scriptContent, final
String html);
}
```

com.iab.omid.library.adsession.Partner

```
package com.iab.omid.library.adsession;

public class Partner {

   /**
    * Create new partner instance providing both name and version.
    * Both name and version are mandatory.
    *
    * @param name    used to uniquely identify the integration partner.
    * @param version used to uniquely identify the integration partner.
    * @return a new partner instance
    * @throws IllegalArgumentException if any of the parameters are either null or blank.
    */
   public static Partner createPartner(final String name, final String version);
}
```

com.iab.omid.library.adsession.AdSessionContext

```
package com.iab.omid.library.adsession;

/**
* This class will provide the ad session both details of the partner and whether this is
considered
* HTML or native.
* <p>
*/
public final class AdSessionContext {

   /**
    * Create a new "html" ad session context.
    *
     * @param partner             details of the integration partner responsible for the ad
session
    * @param webView             responsible for serving the ad content
      * @param customReferenceData containing reference data specific to the integration
partner.
    * @return a new HTML context instance
    * @throws IllegalArgumentException if the supplied partner is null.
    * @throws IllegalArgumentException if the supplied webView is null.
        * @throws IllegalArgumentException if customReferenceData is greater than 256
characters.
    * @see Partner
    * @see WebView
    */
   public static AdSessionContext createHtmlAdSessionContext(final Partner partner,
                                                             final WebView webView,
                                                                   final String
customReferenceData);
```

```
  /**
    * Create a new "native" ad session context.
    *
    * @param partner                    details of the integration partner responsible for
the ad
    *                                   session
     * @param omidJsScriptContent            containing the OMID JS service content to be
injected into
    *                                   the hidden tracking web view
      * @param verificationScriptResources of all verification providers who expect to
receive OMID
    *                                   event data.
      * @param customReferenceData              containing reference data specific to the
integration
    *                                   partner.
    * @return a new native context instance
    * @throws IllegalArgumentException if the supplied partner is null.
      * @throws IllegalArgumentException if the supplied omidJsServiceContent is null or
blank.
     * @throws IllegalArgumentException if the supplied verificationScriptResources is null
or empty.
        * @throws IllegalArgumentException if customReferenceData is greater than 256
characters.
    * @see Partner
    * @see VerificationScriptResource
    */
  public static AdSessionContext createNativeAdSessionContext(final Partner partner,
                                                                              final String
omidJsScriptContent,
                                                                                    final
List<VerificationScriptResource> verificationScriptResources,
                                                                              final String
customReferenceData);
}
```

com.iab.omid.library.adsession.VerificationScriptResource

```
package com.iab.omid.library.adsession;

/**
* Details about the verification provider which will be supplied to the ad session.
* <p>
*/
public final class VerificationScriptResource {

  /**
   * Create new verification script resource instance which requires vendor specific verification
   * parameters.
   * When calling this method all arguments are mandatory.
   *
   * @param vendorKey          used to uniquely identify the verification provider.
   * @param resourceUrl         to be injected into the OMID managed JavaScript execution
   *                              environment.
   * @param verificationParameters which the verification provider script is expecting for the ad
   *                              session.
   * @return a new verification script resource instance
   * @throws IllegalArgumentException if any of the parameters are either null or blank.
   */
  public static VerificationScriptResource createVerificationScriptResourceWithParameters(
      final String vendorKey, final URL resourceUrl, final String verificationParameters);

  /**
   * Create new verification script resource instance which does not require any vendor specific
   * verification parameters.
   * When calling this method all arguments are mandatory.
   *
   * @param resourceUrl to be injected into the OMID managed JavaScript execution environment.
   * @return a new verification script resource instance
   * @throws IllegalArgumentException if any of the parameters are either null or blank.
   */
  public static VerificationScriptResource createVerificationScriptResourceWithoutParameters(
      final URL resourceUrl);
}
```

com.iab.omid.library.adsession.AdSessionConfiguration

```
package com.iab.omid.library.adsession;

public class AdSessionConfiguration {

  /**
   * Create new ad session configuration supplying the owner for both the impression and video
   * events.
   * The OMID JS service will use this information to help identify where the source of these
   * events is expected to be received.
   *
   * @param impressionOwner          of whether the native or JavaScript layer should be responsible for
   *                                 supplying the impression event.
   * @param videoEventsOwner         of whether the native or JavaScript layer should be responsible for
   *                                 supplying video events. This is only required for video ad sessions.
   * @param isolateVerificationScripts When true, verification scripts are in a sandboxed iframe.
   *                                 When false, verification scripts have access to DOM of webview.
   *                                 This setting is only applicable when verification script resources are
   *                                 injected via the javascript session client (typically this would only
   *                                 be relevant for HTML video ad sessions)
   * @return new ad session configuration instance.
   * @throws IllegalArgumentException if the supplied impressionOwner is null.
   */
  public static AdSessionConfiguration createAdSessionConfiguration(final Owner impressionOwner,
                                                                    final Owner videoEventsOwner,
                                                                    final boolean isolateVerificationScripts);

}
```

com.iab.omid.library.adsession.ErrorType

```
package com.iab.omid.library.adsession;

/**
* List of supported error types.
* <ul>
* <li>GENERIC will translate to "generic" string when published to the OMID JS service.</li>
* <li>VIDEO will translate to "video" string when published to the OMID JS service.</li>
* </ul>
* <p>
*/
public enum ErrorType {
  GENERIC("generic"),
  VIDEO("video");
}
```

com.iab.omid.library.adsession.AdSession

```
package com.iab.omid.library.adsession;

/**
 * Ad session API enabling the integration partner to notify OMID of key state relating to
 * viewability calculations.
 * In addition to viewability this API will also notify all verification providers of key
ad session
 * lifecycle events.
 */
public abstract class AdSession {
   /**
    * Create new ad session supplying the adSessionConfiguration and adSessionContext.
    *
     * Note that creating an AdSession sends a message to the OM SDK JS Service running in
the
     * webview.  If the OM SDK JS Service has not loaded before the ad session is created,
the
    * message is lost, and the verification scripts will not receive any events.
    *
     * To prevent this, the implementation must wait until the webview finishes loading OM
SDK
    * JavaScript before creating the AdSession.  The easiest way is to create the AdSession
in a
    * webview callback WebViewClient.onPageFinished().  Alternatively, if an implementation
can
       *  receive  an  HTML5  DOMContentLoaded  event  from  the  webview,  it  can  create  the
OMIDAdSession in
    * a message handler for that event.
    *
     * @param adSessionContext that provides the required information for initialising the
ad session.
    * @return new AdSession instance
    * @throws IllegalArgumentException if the supplied adSessionContext is null.
     * @throws IllegalStateException     if this method has been executed before OMID has
been
    *                                  activated.
    */
            public static AdSession createAdSession(final AdSessionConfiguration
adSessionConfiguration,
                                           final AdSessionContext adSessionContext);

   /**
      * Notify all verification providers that the ad session has started and ad view
tracking will
    * begin.
    * This method will have no effect if called after the ad session has finished.
    */
   @Override
   public void start();

   /**
    * Notify all verification providers that an error has occurred on the ad session.
    *
    * @param errorType of the reported error
    * @param message   containing details of the reported error
    * @throws IllegalArgumentException if the supplied error type is either null.
    * @throws IllegalArgumentException if the supplied message is either null or blank.
    * @throws IllegalStateException    if the ad session has finished.
    */
   @Override
   public void error(final ErrorType errorType, final String message);

   /**
     * Register ad view to be used for tracking viewability. This method should be called
each time
     * the ad view to track changes - i.e. two-part expandable. If an ad view is already
registered
     * for the current session, that ad view will be automatically unregistered and the new
ad view
    * will be registered in its place.
     * If the ad view being registered has been previously registered with a different ad
session,
     * then the ad view will be automatically unregistered from those previously registered
ad sessions.
    * This method will have no effect if called after the ad session has finished.
    *
    * @param adView the native view which should be registered for viewability tracking.
    * @throws IllegalArgumentException if the supplied ad view is null.
    */
   @Override
   public void registerAdView(final View adView);

   /**
     * Notify all verification providers that the ad session has finished and all ad view
tracking
```

```
     * will stop.
     * This method will have no effect if called after the ad session has finished.
     */
    @Override
    public void finish();

    /**
     * Add friendly obstruction which should then be excluded from all ad session viewability
     * calculations.
     * This method will have no effect if called after the ad session has finished.
     *
     * @param friendlyObstruction to be excluded from all ad session viewability calculations.
     * @throws IllegalArgumentException if the supplied friendly obstruction is null.
     */
    @Override
    public void addFriendlyObstruction(final View friendlyObstruction);

    /**
     * Remove registered friendly obstruction.
     * This method will have no effect if called after the ad session has finished.
     *
     * @param friendlyObstruction to be removed from the list of registered friendly obstructions.
     * @throws IllegalArgumentException if the supplied friendly obstruction is null.
     */
    @Override
    public void removeFriendlyObstruction(final View friendlyObstruction);

    /**
     * Utility method to remove all registered friendly obstructions.
     * This method will have no effect if called after the ad session has finished.
     */
    @Override
    public void removeAllFriendlyObstructions();
}
```

com.iab.omid.library.adsession.AdEvents

```
package com.iab.omid.library.adsession;

/**
* Ad event API enabling the integration partner to signal to all verification providers when key
* events have occurred.
* Only one ad events implementation can be associated with the ad session and any attempt to create
* multiple instances will result in an exception.
* <p>
*/
public final class AdEvents {
   /**
    * Create ad events instance associated with the supplied ad session.
    *
    * @param adSession associated with the ad events
    * @return new ad events instance associated with the supplied ad session.
    * @throws IllegalArgumentException if the supplied ad session is null.
    * @throws IllegalStateException    if an ad events instance has already been registered
with
    * the ad session.
    */
   public static AdEvents createAdEvents(final AdSession adSession);

   /**
    * Notify the ad session that an impression event has occurred.
    * When triggered all registered verification providers will be notified of this event.
    *
     * NOTE: the ad session will be automatically started if this method has been called
first.
    *
     * @throws IllegalStateException if the ad session configuration identifies JAVASCRIPT
as the impression owner
    * @throws IllegalStateException if the impression event has already been published
    * @throws IllegalStateException if the session has already been finished
    */
   public void impressionOccurred();

}
```

com.iab.omid.library.adsession.video.InteractionType

```
package com.iab.omid.library.adsession.video;

/**
* List of supported video event user interaction types.
* <p>
*/
public enum InteractionType {
   CLICK("click"),
   INVITATION_ACCEPTED("invitationAccept");
}
```

com.iab.omid.library.adsession.video.PlayerState

```
package com.iab.omid.library.adsession.video;

/**
* List of supported video event player states.
* <p>
*/
public enum PlayerState {
   MINIMIZED("minimized"),
   COLLAPSED("collapsed"),
   NORMAL("normal"),
   EXPANDED("expanded"),
   FULLSCREEN("fullscreen");
}
```

com.iab.omid.library.adsession.video.Position

```
package com.iab.omid.library.adsession.video;

/**
* List of supported video player positions.
* <p>
*/
public enum Position {
   PREROLL("preroll"),
   MIDROLL("midroll"),
   POSTROLL("postroll"),
   STANDALONE("standalone");
}
```

com.iab.omid.library.adsession.video.VastProperties

```
package com.iab.omid.library.adsession.video;

/**
*  This  object  is  used  to  capture  key  VAST  properties  so  this  can  be  shared  with  all
registered verification providers.
* <p>
*/
public final class VastProperties {

   /**
     *  This  method  enables  the  video  player  to  create  a  new  VAST  properties  instance  for
skippable video ad placement.
    *
    * @param skipOffset number of seconds before the skip button is presented
    * @param isAutoPlay whether the video will auto-play content
    * @param position of the video in relation to other content
    * @return new instance of VAST properties
    * @throws IllegalArgumentException if the supplied Position is null.
    */
       public   static   VastProperties   createVastPropertiesForSkippableVideo(final   float
skipOffset,
                                                                          final boolean
isAutoPlay,
                                                                         final Position
position);

   /**
     *  This  method  enables  the  video  player  to  create  a  new  VAST  properties  instance  for
non-skippable video ad placement.
    *
    * @param isAutoPlay whether the video will auto-play content
    * @param position of the video in relation to other content
    * @return new instance of VAST properties
    * @throws IllegalArgumentException if the supplied Position is null.
    */
     public  static  VastProperties  createVastPropertiesForNonSkippableVideo(final  boolean
isAutoPlay,
                                                                              final Position
position);
}
```

com.iab.omid.library.adsession.video.VideoEvents

```
package com.iab.omid.library.adsession.video;

/**
* This provides a complete list of native video events supported by OMID.
* Using this event API assumes the video player is fully responsible for communicating all
video
* events at the appropriate times.
*  Only  one  video  events  implementation  can  be  associated  with  the  ad  session  and  any
attempt to
* create multiple instances will result in an exception.
* <p>
*/
public final class VideoEvents {

   /**
    * Create video events instance for the associated ad session.
     * Any attempt to create a video events instance will fail if the supplied ad session
has
    * already started.
    *
    * @param adSession associated with the ad events.
    * @return new video events instance.
    * @throws IllegalArgumentException if the supplied ad session is null.
       *  @throws  IllegalStateException      if  a  video  events  instance  has  already  been
registered with
    * the ad session.
     * @throws IllegalStateException     if a video events instance has been created after
the ad
    * session has started.
    * @see AdSession
    */
   public static VideoEvents createVideoEvents(final AdSession adSession);

   /**
     *  Notify  all  video  listeners  that  video  content  has  been  loaded  and  ready  to  start
playing.
    *
    * @param vastProperties containing static information about the video placement.
    * @throws IllegalArgumentException if the supplied VAST properties is null.
    * @throws IllegalStateException if the ad session has finished.
```

```
     * @see VastProperties
     */
    public void loaded(final VastProperties vastProperties);

    /**
     * Notify all video listeners that video content has started playing.
     *
     * @param duration          of the selected video media (in seconds).
     * @param videoPlayerVolume from the native video player with a range between 0 and 1.
     * @throws IllegalArgumentException if an invalid duration or videoPlayerVolume has been
supplied.
     * @throws IllegalStateException if the ad session has not been started or has finished.
     */
    public void start(final float duration, final float videoPlayerVolume);

    /**
     * Notify all video listeners that video playback has reached the first quartile.
     *
     * @throws IllegalStateException if the ad session has not been started or has finished.
     */
    public void firstQuartile();

    /**
     * Notify all video listeners that video playback has reached the midpoint.
     *
     * @throws IllegalStateException if the ad session has not been started or has finished.
     */
    public void midpoint();

    /**
     * Notify all video listeners that video playback has reached the third quartile.
     *
     * @throws IllegalStateException if the ad session has not been started or has finished.
     */
    public void thirdQuartile();

    /**
     * Notify all video listeners that video playback is complete.
     *
     * @throws IllegalStateException if the ad session has not been started or has finished.
     */
    public void complete();

    /**
     * Notify all video listeners that video playback has paused after a user interaction.
     *
     * @throws IllegalStateException if the ad session has not been started or has finished.
     */
    public void pause();

    /**
     * Notify all video listeners that video playback has resumed (after being paused) after
a user
```

```
     * interaction.
     *
     * @throws IllegalStateException if the ad session has not been started or has finished.
     */
    public void resume();

    /**
     * Notify all video listeners that video playback has stopped and started buffering.
     *
     * @throws IllegalStateException if the ad session has not been started or has finished.
     */
    public void bufferStart();

    /**
     * Notify all video listeners that buffering has finished and video playback has
resumed.
     *
     * @throws IllegalStateException if the ad session has not been started or has finished.
     */
    public void bufferFinish();

    /**
     * Notify all video listeners that video playback has stopped as a user skip
interaction.
     * Once skipped video it should not be possible for the video to resume playing content.
     *
     * @throws IllegalStateException if the ad session has not been started or has finished.
     */
    public void skipped();

    /**
     * Notify all video listeners that the video player volume has changed.
     *
     * @param videoPlayerVolume from the native video player with a range between 0 and 1.
     * @throws IllegalArgumentException if an invalid videoPlayerVolume has been supplied.
     * @throws IllegalStateException if the ad session has not been started or has finished.
     */
    public void volumeChange(final float videoPlayerVolume);

    /**
     * Notify all video listeners that video player state has changed. See {@link
PlayerState} for
     * list of supported states.
     *
     * @param playerState to signal the latest video player state
     * @throws IllegalArgumentException if the supplied player state is null.
     * @throws IllegalStateException if the ad session has not been started or has finished.
     * @see PlayerState
     */
    public void playerStateChange(final PlayerState playerState);

    /**
     * Notify all video listeners that the user has performed an ad interaction. See
     * {@link InteractionType} for list of supported types.
     *
     * @param interactionType to signal the latest user integration
     * @throws IllegalArgumentException if the supplied interaction type is null.
     * @throws IllegalStateException if the ad session has not been started or has finished.
     * @see InteractionType
     */
    public void adUserInteraction(final InteractionType interactionType);
}
```

iOS OMID Library API

Usage

Set up OMID:

1. Verify that OMIDSDK class exists.
2. Verify that OMIDSDK responds to isCompatibleWithOMIDAPIVersion:
3. Call +[OMIDSDK isCompatibleWithOMIDAPIVersion:]
4. If returns NO, don't call anything else.
5. Activate OMID. Call -[OMIDSDK activateWithOMIDAPIVersion:error:]

After creating ad:

1. Create an OMIDPartner object.
2. If using OMID-managed verification JS, create an OMIDVerificationResource for each verification URL/file.
3. Create an OMIDAdSessionContext object with web view or verification script resources.
4. Create an OMIDAdSession object.
5. Create OMID*Events object(s).

On ad events:

1. Create OMID*Events objects if required.
2. Call OMID*Events methods.

API

OMIDSDK.h

```
/// API Note: this value must be copied into the ad SDK's binary. It cannot be an extern
defined in
/// the OMID library.
#define OMIDSDKAPIVersionString @"{\"v\":\"1.0.4\",\"a\":\"1\"}"

/*!
 * @discussion This application level class will be called by all integration partners to
ensure OM SDK has been activated before calling any other API methods.
 * Any attempt to use other API methods prior to activation will result in an error.
 */
@interface OMIDSDK : NSObject

/*!
 * @abstract The current semantic version of the integrated OMID library.
 */
+ (nonnull NSString *)versionString;
```

```
/*!
 * @abstract Allows the integration partner to check that they are compatible with the
running OMID library version.
 *
 * @param OMIDAPIVersion The version of OMID library integrated by the partner.
 * @return YES if the version supplied is compatible with the integrated OMID library
version.
 */

+ (BOOL)isCompatibleWithOMIDAPIVersion:(nonnull NSString *)OMIDAPIVersion
NS_SWIFT_NAME(isCompatible(withOMIDAPIVersion:));

/*!
 * @abstract Shared OMIDSDK instance.
 */
@property(class, readonly, nonnull) OMIDSDK *sharedInstance
NS_SWIFT_NAME(shared);

/*!
 * @abstract A Boolean value indicating whether the OMID library has been activated.
 *
 * @discussion The value of this property is YES if the OMID library has already been
activated. Allows the integration partner to check that they are compatible with the
running OMID library version.
 */
@property(atomic, readonly, getter = isActive) BOOL active;

/*!
 * @abstract Enables the integration partner to activate OMID prior to calling any other
API methods.
 *
 * @param OMIDAPIVersion The version of OMID library integrated by the partner.
 * @param error If an error occurs, contains an NSError object that describes the problem.
 * @return YES if activation was successful when checking the supplied version number for
compatibility.
 */
- (BOOL)activateWithOMIDAPIVersion:(nonnull NSString *)OMIDAPIVersion
                             error:(NSError *_Nullable *_Nullable)error;

@end
```

OMIDScriptInjector.h

```
/*!
 * @discussion Utility class which enables integration partners to use a standard approach
for injecting OM SDK JS into the served tag HTML content.
 */
@interface OMIDScriptInjector : NSObject

/*
 Injects the downloaded OMID JS content into the served HTML.
 @param scriptContent containing the OMID JS service content to be injected into the hidden
tracking web view.
```

```
 @param html of the tag content which should be modified to include the downloaded OMID JS
content.
 @param error If an error occurs, contains an NSError object.
 @return modified HTML including OMID JS or nil if an error occurs.
 */
+ (nullable NSString *)injectScriptContent:(nonnull NSString *)scriptContent
                                  intoHTML:(nonnull NSString *)html
                                     error:(NSError *_Nullable *_Nullable)error;

@end
```

OMIDPartner.h

```
/*!
  * @discussion Details about the integration partner which will be supplied to the ad
session.
 */
@interface OMIDPartner : NSObject

/*!
 * @abstract Initializes new partner instance providing both name and versionString.
 *
 * @discussion Both name and version are mandatory.
 *
 * @param name It is used to uniquely identify the integration partner.
 * @param versionString It is used to uniquely identify the integration partner.
 * @return A new partner instance, or nil if any of the parameters are either null or blank
 */
- (nullable instancetype)initWithName:(nonnull NSString *)name
                        versionString:(nonnull NSString *)versionString;

@end
```

OMIDAdSessionConfiguration.h

```
typedef NS_ENUM(NSUInteger, OMIDOwner) {
    OMIDJavaScriptOwner = 1, // will translate into "JAVASCRIPT" when published to the OMID
JS service.
     OMIDNativeOwner = 2, // will translate into "NATIVE" when published to the OMID JS
service.
    OMIDNoneOwner = 3 // will translate into "NONE" when published to the OMID JS service.
};

@interface OMIDAdSessionConfiguration : NSObject

/// Returns nil and sets error if OMID isn't active or arguments are invalid.
/// @param impressionOwner providing details of who is responsible for triggering the
impression event.
/// @param videoEventsOwner providing details of who is responsible for triggering video
events. This is only required for video ad sessions.
```

```
/// @param isolateVerificationScripts determines whether verification scripts will be
placed in a sandboxed environment. This will not have any effect for native sessions.
- (nullable instancetype)initWithImpressionOwner:(OMIDOwner)impressionOwner
                                videoEventsOwner:(OMIDOwner)videoEventsOwner
                      isolateVerificationScripts:(BOOL)isolateVerificationScripts
                                           error:(NSError *_Nullable *_Nullable)error;

@end
```

OMIDAdSessionContext.h

```
/*!
 * @discussion This class will provide the ad session both details of the partner and
whether this is considered HTML or native.
 */
@interface OMIDAdSessionContext : NSObject

- (null_unspecified instancetype)init NS_UNAVAILABLE;

/*!
 * @abstract Initializes a new ad session context providing reference to partner and web
view where OMID JS has been injected.
 *
 * @discussion Calling this method will set the ad session type to “html”.
 * <p>
 * NOTE: any attempt to create a new ad session will fail if OMID has not been activated
(see {@link OMIDSDK} class for more information).
 *
 * @param partner Details of the integration partner responsible for the ad session.
 * @param webView The webView responsible for serving the ad content. Must be a UIWebView
or WKWebView instance. The receiver holds a weak reference only.
 * @return A new HTML context instance. Returns nil if OMID has not been activated or if
any of the parameters are nil.
 * @see OMIDSDK
 */
- (nullable instancetype)initWithPartner:(nonnull OMIDPartner *)partner
                                webView:(nonnull UIView *)webView
              customReferenceIdentifier:(nullable NSString *)customReferenceIdentifier
                                  error:(NSError *_Nullable *_Nullable)error;

/*!
 * @abstract Initializes a new ad session context providing reference to partner and a list
of script resources which should be managed by OMID.
 *
 * @discussion Calling this method will set the ad session type to “native”.
 * <p>
 * NOTE: any attempt to create a new ad session will fail if OMID has not been activated
(see {@link OMIDSDK} class for more information).
 *
 * @param partner Details of the integration partner responsible for the ad session.
 * @param resources The array of all verification providers who expect to receive OMID
event data. Must contain at least one verification script. The receiver creates a deep copy
of the array.
 * @return A new native context instance. Returns nil if OMID has not been activated or if
any of the parameters are invalid.
 * @see OMIDSDK
 */
- (nullable instancetype)initWithPartner:(nonnull OMIDPartner *)partner
                                  script:(nonnull NSString *)script
                               resources:(nonnull NSArray<OMIDVerificationScriptResource *>
*)resources
               customReferenceIdentifier:(nullable NSString *)customReferenceIdentifier
                                   error:(NSError *_Nullable *_Nullable)error;

@end
```

OMIDVerificationScriptResource.h

```
/*!
 * @discussion Details about the verification provider which will be supplied to the ad
session.
 */
@interface OMIDVerificationScriptResource : NSObject

/*!
 * @abstract Initializes new verification script resource instance which requires vendor
specific verification parameters.
 *
 * @discussion When calling this method all arguments are mandatory.
 *
 * @param vendorKey It is used to uniquely identify the verification provider.
 * @param URL The URL to be injected into the OMID managed JavaScript execution
environment.
 * @param parameters The parameters which the verification provider script is expecting for
the ad session.
 * @return A new verification script resource instance, or nil if any of the parameters are
either null or blank.
 */
- (nullable instancetype)initWithURL:(nonnull NSURL *)URL
                           vendorKey:(nonnull NSString *)vendorKey
                          parameters:(nonnull NSString *)parameters;

/*!
 * @abstract Initializes new verification script resource instance which does not require
any vendor specific verification parameters.
 *
 * @discussion When calling this method all arguments are mandatory.
 *
 * @param URL The URL to be injected into the OMID managed JavaScript execution
environment.
 * @return A new verification script resource instance, or nil if URL is nil or blank.
 */
```

```
- (nullable instancetype)initWithURL:(nonnull NSURL *)URL;

@end
```

OMIDAdSession.h

```
typedef NS_ENUM(NSUInteger, OMIDErrorType) {
     OMIDErrorGeneric = 1, // will translate into "GENERIC" when published to the OMID JS
service.
      OMIDErrorVideo = 2 // will translate into "VIDEO" when published to the OMID JS
service.
};

/*!
 * @discussion Ad session API enabling the integration partner to notify OMID of key state
relating to viewability calculations.
 * In addition to viewability this API will also notify all verification providers of key
ad session lifecycle events.
 */
@interface OMIDAdSession : NSObject

/*!
 * @abstract Initializes new ad session supplying the context.
 *
 * Note that creating an OMIDAdSession sends a message to the OM SDK JS Service running in
the
 * webview.  If the OM SDK JS Service has not loaded before the ad session is created, the
 * message is lost, and the verification scripts will not receive any events.
 *
 * To prevent this, the implementation must wait until the webview finishes loading OM SDK
  * JavaScript before creating the OMIDAdSession.  The easiest way is to create the
OMIDAdSession
 * in a webview delegate callback (-[WKNavigationDelegate webView:didFinishNavigation:] or
  * -[UIWebViewDelegate webViewDidFinishLoad:]).  Alternatively, if an implementation can
receive an
  * HTML5 DOMContentLoaded event from the webview, it can create the OMIDAdSession in a
message
 * handler for that event.
 *
 * @param context The context that provides the required information for initialising the
ad session.
 * @return A new OMIDAdSession instance, or nil if the supplied context is nil.
 */
-    (nullable    instancetype)initWithConfiguration:(nonnull    OMIDAdSessionConfiguration
*)configuration
                              adSessionContext:(nonnull OMIDAdSessionContext *)context
                                         error:(NSError *_Nullable *_Nullable)error;


/*!
 * @abstract Notifies all verification providers that the ad session has started and ad
view tracking will begin.
```

```
 *
 * @discussion This method will have no affect if called after the ad session has finished.
 */
- (void)start;

/*!
 * @abstract Notifies all verification providers that the ad session has finished and all
ad view tracking will stop.
 *
 * @discussion This method will have no affect if called after the ad session has finished.
 */
- (void)finish;

/*!
 * @abstract Adds friendly obstruction which should then be excluded from all ad session
viewability calculations.
 *
 * @discussion This method will have no affect if called after the ad session has finished.
 *
 * @param friendlyObstruction The view to be excluded from all ad session viewability
calculations.
 */
- (void)addFriendlyObstruction:(nonnull UIView *)friendlyObstruction;

/*!
 * @abstract Removes registered friendly obstruction.
 *
 * @discussion This method will have no affect if called after the ad session has finished.
 *
 * @param friendlyObstruction The view to be removed from the list of registered friendly
obstructions.
 */
- (void)removeFriendlyObstruction:(nonnull UIView *)friendlyObstruction;

/*!
 * @abstract Utility method to remove all registered friendly obstructions.
 *
 * @discussion This method will have no affect if called after the ad session has finished.
 */
- (void)removeAllFriendlyObstructions;

/*!
 * @abstract Notifies the ad session that an error has occurred.
 *
 * @discussion When triggered all registered verification providers will be notified of
this event.
 *
 * @param errorType The type of error.
 * @param message The message containing details of the error.
 */
- (void)logErrorWithType:(OMIDErrorType)errorType message:(nonnull NSString *)message
NS_SWIFT_NAME(logError(withType:message:));
```

```
@end
```

OMIDAdEvents.h

```
//
//  OMIDAdEvents.h
//  AppVerificationLibrary
//

#import <Foundation/Foundation.h>
#import "OMIDAdSession.h"

/*!
 * @discussion Ad event API enabling the integration partner to signal to all verification providers when key events have occurred.
 * Only one ad events implementation can be associated with the ad session and any attempt to create multiple instances will result in an error.
 */
@interface OMIDAdEvents : NSObject

/*!
 * @abstract Initializes ad events instance associated with the supplied ad session.
 *
 * @param session The ad session associated with the ad events.
 * @return A new ad events instance associated with the supplied ad session. Returns nil if the supplied ad session is nil or if an ad events instance has already been registered with the ad session.
 */
- (nullable instancetype)initWithAdSession:(nonnull OMIDAdSession *)session error:(NSError * _Nullable * _Nullable)error;

/*!
 * @abstract Notifies the ad session that an impression event has occurred.
 *
 * @discussion When triggered all registered verification providers will be notified of this event.
 *
 * NOTE: the ad session will be automatically started if this method has been called first.
 */
- (BOOL)impressionOccurredWithError:(NSError *_Nullable *_Nullable)error;

@end
```

OMIDVideoEvents.h

```
/*!
 * @abstract List of supported video event player states.
 */
typedef NS_ENUM(NSUInteger, OMIDPlayerState) {
    OMIDPlayerStateMinimized,
    OMIDPlayerStateCollapsed,
    OMIDPlayerStateNormal,
    OMIDPlayerStateExpanded,
    OMIDPlayerStateFullscreen
};

/*!
 * @abstract List of supported video event user interaction types.
 */
typedef NS_ENUM(NSUInteger, OMIDInteractionType) {
    OMIDInteractionTypeClick,
    OMIDInteractionTypeAcceptInvitation
};

/*!
 * @discussion This provides a complete list of native video events supported by OMID.
 * Using this event API assumes the video player is fully responsible for communicating all
video events at the appropriate times.
 * Only one video events implementation can be associated with the ad session and any
attempt to create multiple instances will result in an error.
 */
@interface OMIDVideoEvents : NSObject

/*!
 * @abstract Initializes video events instance for the associated ad session.
 * @discussion Any attempt to create a video events instance will fail if the supplied ad
session has already started.
 *
 * @param session The ad session associated with the ad events.
 * @return A new video events instance. Returns nil if the supplied ad session is nil or if
a video events instance has already been registered with the ad session or if a video
events instance has been created after the ad session has started.
 * @see OMIDAdSession
 */
- (nullable instancetype)initWithAdSession:(nonnull OMIDAdSession *)session error:(NSError
*_Nullable *_Nullable)error;

/*!
 * @abstract Notifies all video listeners that video content has been loaded and ready to
start playing.
 *
 * @param vastProperties The parameters containing static information about the video
placement.
 * @see OMIDVASTProperties
 */
- (void)loadedWithVastProperties:(nonnull OMIDVASTProperties *)vastProperties;

/*!
 * @abstract Notifies all video listeners that video content has started playing.
 *
 * @param duration The duration of the selected video media (in seconds).
 * @param videoPlayerVolume The volume from the native video player with a range between 0
and 1.
 */
```

```
- (void)startWithDuration:(CGFloat)duration
       videoPlayerVolume:(CGFloat)videoPlayerVolume;

/*!
  * @abstract Notifies all video listeners that video playback has reached the first
quartile.
 */
- (void)firstQuartile;

/*!
 * @abstract Notifies all video listeners that video playback has reached the midpoint.
 */
- (void)midpoint;

/*!
  * @abstract Notifies all video listeners that video playback has reached the third
quartile.
 */
- (void)thirdQuartile;

/*!
 * @abstract Notifies all video listeners that video playback is complete.
 */
- (void)complete;

/*!
  * @abstract Notifies all video listeners that video playback has paused after a user
interaction.
 */
- (void)pause;

/*!
  * @abstract Notifies all video listeners that video playback has resumed (after being
paused) after a user interaction.
 */
- (void)resume;

/*!
  * @abstract Notifies all video listeners that video playback has stopped as a user skip
interaction.
 * @discussion Once skipped video it should not be possible for the video to resume playing
content.
 */
- (void)skipped;

/*!
  * @abstract Notifies all video listeners that video playback has stopped and started
buffering.
 */
- (void)bufferStart;

/*!
```

```
 * @abstract Notifies all video listeners that buffering has finished and video playback
has resumed.
 */
- (void)bufferFinish;

/*!
 * @abstract Notifies all video listeners that the video player volume has changed.
 *
 * @param playerVolume The volume from the native video player with a range between 0 and
1.
 */
- (void)volumeChangeTo:(CGFloat)playerVolume;

/*!
 * @abstract Notifies all video listeners that video player state has changed. See {@link
OMIDPlayerState} for list of supported states.
 *
 * @param playerState The latest video player state.
 * @see OMIDPlayerState
 */
- (void)playerStateChangeTo:(OMIDPlayerState)playerState;

/*!
 * @abstract Notifies all video listeners that the user has performed an ad interaction.
See {@link OMIDInteractionType} fro list of supported types.
 *
 * @param interactionType The latest user integration.
 * @see OMIDInteractionType
 */
- (void)adUserInteractionWithType:(OMIDInteractionType)interactionType
NS_SWIFT_NAME(adUserInteraction(withType:));

@end
```

OMID JS ad session client API

The API detailed below should be used where the integration partner relies on JS components when contributing to the ad session state. This API can be used in the following scenarios;

1. Video ad session relying on the HTML5 video player for injecting verification script resources and/or publishing OMID video events.
2. Display ad session relying on a separate JS component to handle the impression event.

Partner

Constructor Summary

Partner(name, version);

Method Summary

No public methods available.

VerificationScriptResource

This object is intended to be used by JavaScript integration partners responsible for parsing the VAST ad response. When the video player discovers <Verification> nodes these should be registered with the OMID JS data service via this API.

Constructor Summary

```
VerificationScriptResource(String resourceUrl, String vendorKey, String verificationParameters)
```

NOTE: the vendorKey is only mandatory when verification parameters have been provided.

Method Summary

No public methods available.

Context

Constructor Summary

```
Context(partner, verificationScriptResources);
```

`partner` is mandatory for all Context instances
`verificationScriptResources` is optional but when supplied must contain a list of resources intended to be handled by OMID JS.

Method Summary

Modifier and type	Method and description
void	setVideoElement(videoElement) videoElement: HTMLVideoElement DOM object. Specifies the video player element in the webview. Causes OM SDK JS Service to include DOM geometry in AdView-providing events (e.g. geometryChange, impression, etc) and media playback state in video events. If the video player is in a cross domain iframe, it won't be accessible to the OM SDK JS Service. The ad session client should use setSlotElement().
void	setSlotElement(slotElement) slotElement: DOM object with ad creative. Specifies the ad creative element in the webview. Causes OM SDK JS Service to include DOM geometry in AdView-providing events (e.g. geometryChange, impression, etc). If the ad creative is in a cross-domain iframe, it won't be accessible to the OM SDK JS Service. The ad session client should pass the iframe element to this method, and should also call Session.setElementBounds() if the ad creative does not fill the iframe.

OmidVersion

Constructor Summary

Omid(String semanticVersion, String apiLevel);

Method Summary

No public methods available.

AdSession

Similar to the OMID JS verification client this provides a JavaScript representation of the ad session enabling JS components to contribute to the overall state and publish events. The OMID JS ad session is responsible for communicating to the OMID JS data service and will also handle scenarios with limited access to the OMID JS data service - i.e. cross-domain iFrames.

Constructor Summary

AdSession(context);

Method Summary

Modifier and type	Method and description
boolean	`isSupported()`
void	`registerSessionObserver(functionToExecute)` Allows ad session clients to observe the ad session lifecycle. Each session observer will be notified of the following three events; - sessionStart - sessionError - sessionFinish Details of each event type have been detailed here. functionToExecute(event) - function to execute when the event has been triggered. All listeners will be required to support a single event parameter - see table below for data structure.
void	error(errorType, message) Allows JS ad session clients to notify verification clients of any errors. Possible errorType values include; "GENERIC" and "VIDEO". When calling this method all verification clients will be notified via the sessionError session observer event.
void	setElementBounds(elementBounds) `elementBounds`: Rectangle `{x, y, width, height}` relative to geometry of slotElement. If `slotElement` is an unfriendly iframe within the webview, the `elementBounds` rectangle specifies the location of the creative within the iframe. The ad session script must call `setElementBounds` whenever the creative geometry changes relative to the slotElement.

AdEvents

Constructor Summary

AdEvents(adSession);

Method Summary

void	impressionOccurred() Notify all verification providers that an impression event should be recorded.

VastProperties

Constructor Summary

VastProperties(boolean isSkippable, float skipOffset, boolean isAutoPlay, string position)

Method Summary

No public methods available.

PlayerState

Constructor Summary

No public constructors available.

Enumeration Summary

Enum	Description
MINIMIZED	The player is collapsed in such a way that the video is hidden. The video may or may not still be progressing in this state, and sound may be audible. This refers specifically to the video player state on the page, and not the state of the browser window.
COLLAPSED	The player has been reduced from its original size. The video is still potentially visible.
NORMAL	The player's default playback size.
EXPANDED	The player has expanded from its original size.
FULLSCREEN	The player has entered fullscreen mode.

InteractionType

Constructor Summary

No public constructors available.

Enumeration Summary

Enum	Description
CLICK	The user clicked to load the ad's landing page.
INVITATION_ACCEPTED	The user engaged with ad content to load a separate experience.

VideoEvents

This will be integrated by video players who wish to maintain full control over the video event lifecycle. The adaptor will also be responsible for handling all communication to the OMID JS ad session instance.

Constructor Summary

VideoEvents(adSession);

Method Summary

Modifier and type	Method and description
void	loaded(VastProperties vastProperties);
void	start(float duration, float videoPlayerVolume); The `videoPlayerVolume` range is between 0 and 1.
void	firstQuartile();
void	midpoint();
void	thirdQuartile();
void	complete();
void	pause();
void	resume();

void	bufferStart();
void	bufferFinish();
void	skipped();
void	volumeChange(float videoPlayerVolume); The `videoPlayerVolume` range is between 0 and 1.
void	playerStateChange(PlayerState playerState);
void	adUserInteraction(InteractionType interactionType);

OMID JS Verification Client API

The Open Measurement SDK (OMSDK) project provides a way for verification providers to measure ad inventory using a common interface across many environments. The OMID JS Verification Client API is the endpoint of that interface: the methods and events that are exposed to verification code. This API may also be adopted by (non-OMSDK-based) third-parties in order to enable OMID verification scripts to measure their inventory.

Verification Client

The OMSDK project publishes the OMID JS Verification Client, a JavaScript library which should be integrated into all OMID verification resources. This utility understands the different underlying mechanisms that might be used to access OMID data and exposes a single consistent interface to verification code. It is designed to work with all direct OMSDK integrations, but will also be compatible with third-party implementations of the OMID JS Verification Client API which follow the implementation guide.

Non-Browser Environments

The OMID JS Verification Client includes several methods essential for verification code (e.g. setTimeout) but that would be unavailable when running in certain non-browser environments (e.g. iOS' JavaScriptCore) where many common functions are not provided. When executed in a standard browser or webview, the library will automatically fallback to built-in functionality.

Integration

The standard process for working with the OMID JS client includes:

1. Copy the OMID JS client source code into your project
2. Create new OMID JS client instance
3. Interface with OMID JS client in order to access OMID state
4. Ensure OMID JS client has been included as part of any minification process

NOTE: the OMID JS client source code is available and has been designed to be minified as part of the JavaScript build process.

VerificationClient

The following methods are available on the `VerificationClient` class provided by the OMID JS Verification Client library.

Constructor Summary

`VerificationClient()`

Example: Client Initialization

```
const omidClient = new VerificationClient();
if (omidClient.isSupported()) {
  startMonitoring(omidClient);
}
```

Method Summary

`boolean isSupported()`

Returns true if an OMID-compatible environment has been detected and is available for use.

`registerSessionObserver(observer, vendorKey)`

Subscribes to all session events (`sessionStart`, `sessionError`, and `sessionFinish`). This method also signals that the verification script has loaded and is ready to receive events, and should be called upon initialization.

Parameter Name	Parameter Type	Description
`observer`	`function(OmidEvent)`	An event handler which will be invoked on session events.
`vendorKey`	`string`	Optional. A string identifying the caller which is used to match with the appropriate verification parameters. In the case of VAST-served video ads, when the `vendorKey` matches the `vendor` attribute of the `<Verification>` element, the value of the `<VerificationParameters>` under that `<Verification>`, if any, should be passed to the matching caller in the event data of the `sessionStart` event. If this value is not provided, no `verificationParameters` will be provided in the data of the `sessionStart` event.

Example: Passing Verification Parameters from VAST

```
<VAST>
  ...
  <AdVerifications>
    <Verification vendor="company.com-omid">
      <VerificationParameters>
        <![CDATA[{'id': 1234, 'option': true}]]>
      </VerificationParameters>
    </Verification>
  </AdVerifications>
  ...
```

```
</VAST>

// Verification code from company.com registers for session events:
omidClient.registerSessionObserver(observer, 'company.com-omid');


// When OMID code invokes the "company.com-omid" observer for the
// sessionStart event, it includes the matching verification
// parameters in the event data.
const compmanyDotComSessionStartEvent = {
  'adSessionId': 'ABC-123',
  'type': 'sessionStart',
  'timestamp': 123456,
  'data': {
    'context': {...},
    'verificationParameters': "{'id': 1234, 'option': true}"
  },
};
observer(companyDotComSessionStartEventData);
```

`addEventListener(type, listener)`

Subscribes to ad lifecycle and metric events.

Parameter Name	Parameter Type	Description
`type`	`string`	The event type to subscribe this listener to.
`listener`	`function(OmidEvent)`	An event handler to be invoked when the given event type is triggered.

`sendUrl(url, successCallback, failureCallback)`

Issues a request to the given URL, selecting the most appropriate method for remote communication based on execution environment.

Parameter Name	Parameter Type	Description
`url`	`string`	The URL to be requested.
`successCallback`	`function()`	Optional. A callback to be executed in the event that the request was successfully received (2xx response code).
`failureCallback`	`function()`	Optional. A callback to be executed in the event that the request was not successfully received (non-success response code or other error).

number setTimeout(callback, timeInMillis)

Invokes a callback after a given delay time. Returns an ID value used to identify the call which can be used to cancel it via `clearTimeout`. Provides behavior equivalent to the `setTimeout` web API method.

Parameter Name	Parameter Type	Description
`callback`	`function()`	The function to invoke after the delay.
`timeInMillis`	`number`	The number of milliseconds to wait before invoking the callback.

clearTimeout(timeoutId)

Clears a callback with the given ID (issued from `setTimeout`) and prevents it from being executed. Provides behavior equivalent to the `clearTimeout` web API method.

Parameter Name	Parameter Type	Description
`timeoutId`	`number`	The ID returned from setTimeout of the callback to cancel.

number setInterval(callback, timeInMillis)

Repeatedly invokes a callback, waiting the given delay time between calls. Returns an ID value used to identify the call which can be used to cancel it via `clearInterval`. Provides behavior equivalent to the `setInterval` web API method.

Parameter Name	Parameter Type	Description
`callback`	`function()`	The function to repeatedly invoke
`timeInMillis`	`number`	The number of milliseconds to wait between function invocations.

clearInterval(intervalId)

Clears a repeated callback with the given ID (issued from `setInterval`) and cancels its further execution. Provides behavior equivalent to the `clearInterval` web API method.

Parameter Name	Parameter Type	Description
`intervalId`	`number`	The ID returned from setInterval of the repeated callback to cancel.

`injectJavaScriptResource(url, successCallback, failureCallback)`

Injects the JavaScript resource from the given URL into the same execution context as caller. For browser-based environments, this will essentially append a new `<script>` element pointing at the url into the DOM of the containing window.

Parameter Name	Parameter Type	Description
`url`	`string`	The URL of the JavaScript resource to load into the environment.
`successCallback`	`function()`	Optional. A callback to be executed in the event that the script was successfully loaded.
`failureCallback`	`function()`	Optional. A callback to be executed in the event that the script failed to load.

OMID Events

Event Objects

All events passed to verification code session observers or event listeners are objects containing the following properties.

Property Name	Property Type	Description
`adSessionId`	`string`	The Ad Session ID, a unique value provided by the OMID implementer for tracking individual ad lifecycles.
`type`	`string`	The type of event this object represents.
`timestamp`	`number`	The time this event originally occurred. This may not be the current time, as events may be cached and replayed for late loading verification code.
`data`	`Object`	Only available on for particular event types. Certain events include additional data containing more specific details about the triggering event. See individual event definitions for details.

Example: OMID Event Subscription

```
omidClient.addEventListener('volumeChange', function(evt) {
  console.log(
      'Session ' + evt.adSessionId +
      ' changed volume to ' + evt.data.videoPlayerVolume +
      ' at ' + evt.timestamp);
});
```

Event Caching

OMID providers will cache events which may have been missed by late loading verification code. Following event subscription, any previously events that previously occurred will be passed to event handlers in chronological order. The `timestamp` property of the event objects will indicate when the event was originally fired.

Session Events

These events are all subscribed to implicitly when calling [`registerSessionObserver`](). **These events should not be explicitly subscribed to via the addEventListener method.**

Example: Subscribing to session events

```
const vendorKey = 'verify.com-omid';

function observeSession(evt) {
  const sessionId = evt.adSessionId;
  if (evt.type == 'sessionFinish') {
    handleSessionEnd(sessionId);
  } else if (evt.type == 'sessionError') {
    logError(sessionId, evt.data.errorType, evt.data.message);
  } else {
    // Handle sessionStart event.
    const vendorData = parseParams(evt.data.verificationParameters);
    startMonitoring(sessionId, evt.data.context, vendorData);
  }
}

omidClient.registerSessionObserver(observeSession, vendorKey);
```

sessionStart

This event fires as soon as the OMID provider has initialized and has the necessary data to fill in the `context` and `verificationParameters` of the event data, following a call to [`registerSessionObserver`](). It does not imply that the ad has rendered or the video has started playing; it only marks the initialization of the of the ad session. This is always the first event fired for a session.

Event Data

Property Name	Property Type	Description
`context`	[`Context`]()	An object describing the static properties of the playback environment. See [Context Object]() for

		details.
`verificationParameters`	`string`	The per-vendor initialization parameters for this session observer. This value is only provided if `registerSessionObserver` was called with a `vendorKey` argument matching a known vendor+parameters pair. In the case of VAST-served video ads, these pairs come from the `<Verification>` element. If the `vendor` attribute of the `<Verification>` matches `vendorKey`, the value of the `<VerificationParameters>` under that `<Verification>`, if any, is provided here.

Context Object

<table>
<tr><th>Property Name</th><th>Property Type</th><th>Description</th></tr>
<tr><td><code>apiVersion</code></td><td><code>string</code></td><td>The version of the OMID JS Verification Client API provided ("1.0" for this document).</td></tr>
<tr><td><code>environment</code></td><td><code>string</code></td><td><table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>app</td><td>Mobile app environment (i.e. any integration involving a native layer)</td></tr>
<tr><td>web</td><td>Web-only environment (no native layer)</td></tr>
</table></td></tr>
<tr><td><code>accessMode</code></td><td><code>string</code></td><td><table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>limited</td><td>Verification code is executed in a sandbox with only indirect information about the ad.
In this case, geometry information is provided through OMID events (e.g. geometryChange).</td></tr>
<tr><td>full</td><td>Verification code is executed with direct access to the video or rendering element (i.e. is not sandboxed).
Specifically, a "full" <code>accessMode</code> implies both that the verification code has access to the creative element and that a reference to that element will be provided via either <code>videoElement</code> or <code>slotElement</code>.</td></tr>
</table></td></tr>
<tr><td><code>videoElement</code></td><td><code>HTMLVideoElement</code></td><td>Required for all "full" <code>accessMode</code> linear video ads, or any ad</td></tr>
</table>

<table>
<tr><th></th><th></th><th></th></tr>
<tr><td></td><td></td><td>where a <video> element is the main focal point of the creative, otherwise not provided. For video creatives that do not use HTML5 video at all, slotElement may be used instead.
This is the <video> element where the creative is played.</td></tr>
<tr><td>slotElement</td><td>Element</td><td>Required for "full" accessMode display ads, or for any video ad where no <video> element is used or if used does not provide a complete picture of where the creative is rendering. It should not be provided for standard linear video ads; videoElement should be passed instead.
This is the HTML element inside which the creative is rendered.</td></tr>
<tr><td>adSessionType</td><td>string</td><td><table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>native</td><td>Control of the ad session is directed from the native layer.</td></tr>
<tr><td>html</td><td>Control of the ad session is directed from a web layer.</td></tr>
</table></td></tr>
<tr><td>adServingId</td><td>string</td><td>Only provided when available.
The <AdServingId> value of the current ad from the VAST, if one is available. Only provided if a value was available in the VAST.</td></tr>
<tr><td>transactionId</td><td>string</td><td>Only provided when available.
The [TRANSACTIONID] value of the ad request chain, if one is available, as defined in VAST 4.1. Only provided when a value is available and known to the OMID provider.</td></tr>
<tr><td>podSequence</td><td>string</td><td>Only provided when available.
The value of the sequence attribute from the <Ad> of the current session. Only provided if the attribute was present (i.e. this is an ad in a pod). This matches the value of the [PODSEQUENCE] macro described in VAST 4.1.</td></tr>
<tr><td>adCount</td><td>number</td><td>Only provided when available.
The number of <InLine> ads played within the current chain or tree of VASTs, including the executing one. That is, this value starts at 1 and increments for each video played, whether it was pulled from a pod, buffet, nested pod, etc. In standard non-pod VAST responses with a single <InLine> ad, this value is always 1. This matches the value of the [ADCOUNT] macro described in VAST 4.1</td></tr>
<tr><td>omidNativeInfo</td><td>Object</td><td>Only present when a native layer is involved in the ad session. All properties are required when present.</td></tr>
</table>

<table>
<tr><td></td><td></td><td>
<table>
<tr><th>Property Name</th><th>Property Type</th><th>Description</th></tr>
<tr><td>partnerName</td><td>string</td><td>The name of the native layer OMSDK integration partner.</td></tr>
<tr><td>partnerVersion</td><td>string</td><td>The version of the native layer OMSDK integration partner.</td></tr>
</table>
<pre>
omidNativeInfo: {
    partnerName: 'exampleNativeSDK',
    partnerVersion: '1.0.0'
}
</pre>
</td></tr>
<tr><td>omidJsInfo</td><td>Object</td><td>
<table>
<tr><th>Property Name</th><th>Property Type</th><th>Description</th></tr>
<tr><td>omidImplementer</td><td>string</td><td>The name of the OMID provider. For OMSDK integrations this is always "omsdk".</td></tr>
<tr><td>serviceVersion</td><td>string</td><td>For OMSDK integrations, this is the version of the OMSDK JS service used.
For third-party implementations, this is the version of the OMID provider code, and should match the party named in omidImplementer.</td></tr>
<tr><td>sessionClientVersion</td><td>string</td><td>The version of OMSDK JS ad session client used. This is required for OMSDK integrations where a JavaScript SDK contributes to the ad session, and is otherwise not provided.</td></tr>
<tr><td>partnerName</td><td>string</td><td>The name of the JS-level integration partner, if one exists. This will be the name of any JavaScript SDK that is involved in executing the ad session.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr>
<td></td>
<td></td>
<td>
<table>
<tr><td>partnerVersion</td><td>string</td><td>The version of the code provided by the party from partnerName, if partnerName was provided.</td></tr>
</table>
<pre>
omidJsInfo: {
    omidImplementer: 'omsdk',
    serviceVersion: '1.0.0',
    sessionClientVersion: '1.0.0',
    partnerName: 'exampleJsSdk',
    partnerVersion: '3.4.2'
}
</pre>
</td>
</tr>
<tr>
<td>app</td>
<td>Object</td>
<td>
<p>Required for OMSDK mobile app integrations, otherwise not provided.</p>
<p>Provides details about the running app and native OMSDK version.</p>
<table>
<tr><th>Property Name</th><th>Property Type</th><th>Description</th></tr>
<tr><td>libraryVersion</td><td>string</td><td>The version of the compiled native OMSDK library used.</td></tr>
<tr><td>appId</td><td>string</td><td>The bundle or package name of the mobile application in which the ad is rendered.</td></tr>
</table>
<pre>
app: {
  libraryVersion: '1.0.0',
  appId: 'com.bundle.app'
}
</pre>
</td>
</tr>
<tr>
<td>deviceInfo</td>
<td>Object</td>
<td>
<p>Required for OMSDK mobile app integrations, otherwise not provided.</p>
<p>Provides details about the mobile device.</p>
<table>
<tr><th>Property Name</th><th>Property Type</th><th>Description</th></tr>
<tr><td>deviceType</td><td>string</td><td>Name of the device (e.g. "iPhoneX").</td></tr>
<tr><td>os</td><td>string</td><td>Name of the operating system (e.g. "iOS", "Android").</td></tr>
<tr><td>osVersion</td><td>string</td><td>Operating system version ("11.1.2").</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr><th></th><th></th><th></th></tr>
<tr><td><code>supports</code></td><td><code>Array &lt;string&gt;</code></td><td>A list of optional features which may be implemented in the current environment.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>clid</td><td>Indicates that this implementation always provides the geometryChange event and geometry data on the impression event, even when "full" accessMode is used. Note that geometry is always provided in "limited" accessMode cases, even if "clid" is not set.</td></tr>
</table></td></tr>
<tr><td><code>customReferenceData</code></td><td><code>string</code></td><td>Optional. Provides key reference data related to the ad session. There is no formal structure to the reference data, but enables integration partners to share key data with verification providers.</td></tr>
</table>

sessionError

This event is fired following a playback, rendering, or other ad-related error, which may be session terminal or recoverable. In the case of non-recoverable errors, this event **does not** replace `sessionFinish`, which still must be fired following the `sessionError` event.

Event Data

<table>
<tr><th>Property Name</th><th>Property Type</th><th>Description</th></tr>
<tr><td><code>errorType</code></td><td><code>string</code></td><td>High level error type.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>video</td><td>Video-related rendering or loading errors</td></tr>
<tr><td>generic</td><td>Catch-all for other issues</td></tr>
</table></td></tr>
<tr><td><code>message</code></td><td><code>string</code></td><td>Description of the session error.</td></tr>
</table>

sessionFinish

This event is fired when the ad session has terminated and indicates that verification resources should start clean up and handle end-of-session reporting. This is the always the last event sent for a session.

Event Data

None.

Lifecycle and Metric Events

Verification code can subscribe to these events using the `addEventListener` method.

impression

The OMID provider has recorded an impression for this ad. For video ads, this corresponds to the VAST `<Impression>` and should be fired simultaneously with that event.

Event Data

<table>
<tr><th>Property Name</th><th>Property Type</th><th>Description</th></tr>
<tr><td>mediaType</td><td>string</td><td>The media type measured for this impression.
<table><tr><th>Value</th><th>Description</th></tr><tr><td>display</td><td>Used for display ad impressions.</td></tr><tr><td>video</td><td>Used for video ad impressions.</td></tr></table></td></tr>
<tr><td>videoEventAdaptorType</td><td>string</td><td>Provided only for OMSDK integrations on impressions where the mediaType is "video".
<table><tr><th>Value</th><th>Description</th></tr><tr><td>jsCustom</td><td>Used when a JS event adaptor is used.</td></tr><tr><td>nativeCustom</td><td>Used when a native event adaptor is used.</td></tr></table></td></tr>
<tr><td>videoEventAdaptorVersion</td><td>string</td><td>Provided only for OMSDK integrations where videoEventAdaptorType is also provided.
This is the version of the video event adaptor code used.</td></tr>
<tr><td>viewport</td><td>Object</td><td>Only required when accessMode "limited" is used.
The state of the viewport (the mobile device screen or the browser window) at impression time.
<table><tr><th>Property Name</th><th>Property Type</th><th>Description</th></tr><tr><td>width</td><td>number</td><td>The viewport width.</td></tr><tr><td>height</td><td>number</td><td>The viewport height.</td></tr></table>
viewport: {
    width: 320,</td></tr>
</table>

		`    height: 480`
		`}`

`adView`	[`AdView`]()	Only required when `accessMode` "limited" is used. The ad geometry at impression time.

video

This is a special event type which is shorthand to allow subscription to many video playback-related events with a single call to `addEventListener`. Triggered events will never contain "`video`" as the event type, but rather the actual underlying type (e.g. "`start`").

The following events are all subscribed to with a single call to `addEventListener` with the "`video`" event type. **Verifiers should take care not to unintentionally double subscribe to these events.**

- loaded
- start
- firstQuartile
- midpoint
- thirdQuartile
- complete
- pause
- resume
- bufferStart
- bufferFinish
- skipped
- volumeChange
- playerStateChange
- adUserInteraction

Example: Subscribing to the video event

```
omidClient.addEventListener('video', function(evt) {
  switch (evt.type) {
    case 'start':
      handleVideoStart(evt);
      break;
    case 'firstQuartile':
    case 'midpoint':
    case 'thirdQuartile':
    case 'complete':
      handlePlaybackProgress(evt);
      break;
    case 'pause':
    case 'resume':
      handlePlayPause(evt);
      break;
  }
});
```

Event Data

See individual event descriptions.

loaded

Video-only event. The player has loaded and buffered the creative's media and assets either fully or to the extent that it is ready to play the media. Corresponds to the VAST `loaded` event.

Event Data

<table>
<tr><th>Property Name</th><th>Property Type</th><th>Description</th></tr>
<tr><td><code>skippable</code></td><td><code>boolean</code></td><td>Whether the ad can be skipped by the user.</td></tr>
<tr><td><code>skipOffset</code></td><td><code>number</code></td><td>Required when <code>skippable</code> is true. Otherwise should not be provided.
The number of seconds after which the player makes the UI to skip the ad available to the user. Corresponds to the <code>skipoffset</code> attribute from VAST.</td></tr>
<tr><td><code>autoPlay</code></td><td><code>boolean</code></td><td>Whether the ad playback will be automatically started without input from the user.</td></tr>
<tr><td><code>position</code></td><td><code>string</code></td><td><table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>preroll</td><td>The ad plays preceding video content.</td></tr>
<tr><td>midroll</td><td>The ad plays in the middle of video content, or between two separate content videos.</td></tr>
<tr><td>postroll</td><td>The ad plays following video content.</td></tr>
<tr><td>standalone</td><td>The ad plays independently of any video content.</td></tr>
</table></td></tr>
</table>

start

Video-only event. The player began playback of the video ad creative. Corresponds to the VAST `start` event.

Event Data

Property Name	Property Type	Description
`duration`	`number`	The duration of the video ad, in seconds.
`videoPlayerVolume`	`number`	The initial player volume level at playback start, scaled to a 0 to 1 range. The player is muted when the level is 0, and at full volume when the level is 1.
`deviceVolume`	`number`	Only provided for mobile app environments when device volume is available. The initial device volume level at playback start, scaled to a 0 to 1 range. The device is muted when the level is 0, and at full volume when the level is 1.

firstQuartile

Video-only event. The creative played continuously for at least 25% of the total duration. Corresponds to the VAST `firstQuartile` event.

Event Data

None.

midpoint

Video-only event. The creative played continuously for at least 50% of the total duration. Corresponds to the VAST `midpoint` event.

Event Data

None.

thirdQuartile

Video-only event. The creative played continuously for at least 75% of the total duration. Corresponds to the VAST `thirdQuartile` event.

Event Data

None.

complete

Video-only event. The creative played to the end for 100% of the total duration. Corresponds to the VAST `complete` event.

Event Data

None.

pause

Video-only event. Playback was stopped in a way from which it may later be resumed, due to user interaction.

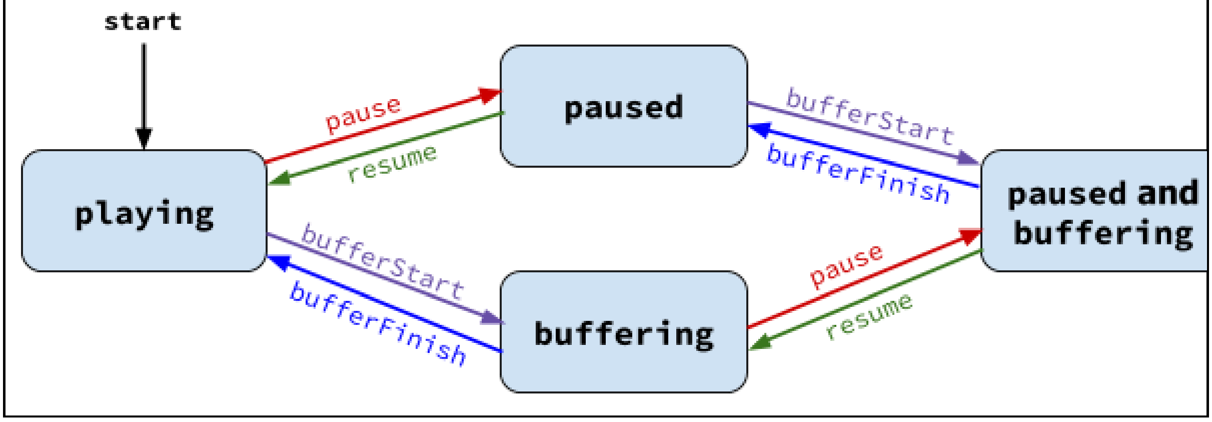
Event Data

None.

NOTE: Semantics of pause/resume and bufferStart/bufferFinish

The `pause` and `resume` events, and the `bufferStart` and `bufferFinish` events, are meant to communicate the entering and exiting of the **`paused`** and **`buffering`** states respectively. These states are implicit, and the player and verification code should track them internally, according to the following rules.

- The player is initially in the `playing` state following the start event
- Video playback is progressing only when the player is in the `playing` state
- The `pause` event should only be fired when the player is in a non-`paused` state
- The `resume` event should only be fired when the player is in a `paused` state
- The `bufferStart` event should only be fired when the player is in a non-`buffering` state
- The `bufferFinish` event should only be fired when the player is in a `buffering` state



resume

Video-only event. Playback resumed following a user-originated pause.

Event Data

None.

bufferStart

Video-only event. Playback was stopped in a way from which it may later be resumed, due to a cause other than user interaction (generally buffering from insufficient available video data).

Event Data

None.

bufferFinish

Video-only event. Playback has resumed following a non-user-originated pause.

Event Data

None.

skipped

Video-only event. The user activated a control which caused ad playback to terminate. Corresponds to the VAST `skip` event.

Event Data

None.

volumeChange

Video-only event. The player and/or device volume has changed.

Event Data

Property Name	Property Type	Description
`videoPlayerVolume`	`number`	The current player volume, scaled to a 0 to 1 range. The player is muted when the level is 0, and at full volume when the level is 1.
`deviceVolume`	`number`	Only provided for mobile app environments when device volume is available. The current device volume, scaled to a 0 to 1 range. The device is muted when the level is 0, and at full volume when the level is 1.

playerStateChange

Video-only event. The player has changed playback states, generally to resize. This includes moving from non-fullscreen to fullscreen state. The assumption is that at start time the video is in the "normal" state. If playback begins when the player is in a "minimized" or "fullscreen" state, then this event is fired immediately following `start` in order to reflect the current state.

Event Data

<table>
<tr><th>Property Name</th><th>Property Type</th><th>Description</th></tr>
<tr><td><code>state</code></td><td><code>string</code></td><td>The new playback state of the player.
Suggested values are as follows, roughly in order of increasing size:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>minimized</td><td>The player is collapsed in such a way that the video is hidden. The video may or may not still be progressing in this state, and sound may be audible. This refers specifically to the video player state, and <b>not</b> the state of the app or browser window.</td></tr>
<tr><td>collapsed</td><td>The player has been reduced from its original size. The video is still potentially visible.</td></tr>
<tr><td>normal</td><td>The player's default playback size.</td></tr>
<tr><td>expanded</td><td>The player has expanded from its original size.</td></tr>
<tr><td>fullscreen</td><td>The player has entered fullscreen mode.</td></tr>
</table></td></tr>
</table>

adUserInteraction

The user has interacted with the ad outside of any standard playback controls (e.g. clicked the ad to load an ad landing page).

NOTE: If this interaction causes playback to pause, then this event should be followed by a separate `pause` event.

Event Data

<table>
<tr><th>Property Name</th><th>Property Type</th><th>Description</th></tr>
<tr><td><code>interactionType</code></td><td><code>string</code></td><td>The type of interaction which triggered the event.
Possible interaction types are as follows:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>click</td><td>The user clicked to load the ad's landing page.</td></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td></td><td><table><tr><td>invitationAccept</td><td>The user engaged with ad content to load a separate experience.</td></tr></table></td></tr>
</table>

geometryChange

The geometry state has changed. Specifically, this event is fired every time the ad container state changes such that any field of the viewport or adView would change value from the previous report.

This event is only required to be provided in `accessMode` "limited" environments. It may optionally still be provided in "full" `accessMode`. If the OMID implementer does provide the `geometryChange` event even when not required, it should include the value "clid" in the in the `supports` array in the [`sessionStart` context](), so that this can be detected.

All size and location units reported in the event data of `geometryChange` are in density-independent pixels with all coordinates are relative to the screen coordinates.

Event Data

<table>
<tr><th>Property Name</th><th>Property Type</th><th>Description</th></tr>
<tr><td><code>viewport</code></td><td><code>Object</code></td><td>The state of the viewport (the mobile device screen or the browser window) at impression time.<table><tr><th>Property Name</th><th>Property Type</th><th>Description</th></tr><tr><td><code>width</code></td><td><code>number</code></td><td>The viewport width.</td></tr><tr><td><code>height</code></td><td><code>number</code></td><td>The viewport height.</td></tr></table></td></tr>
<tr><td><code>adView</code></td><td><code>AdView</code></td><td>Provides full geometry data of the registered ad view including obstructions along with any detected reason codes.</td></tr>
</table>

Rectangle Object

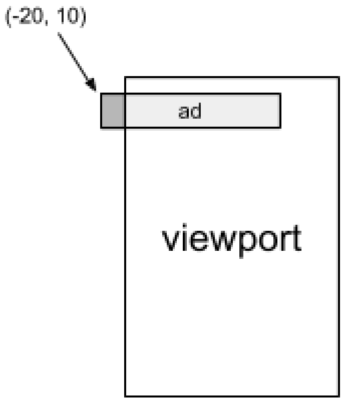
An object representing the size and location of a rectangular area.

Property Name	Property Type	Description
`x`	`number`	The x-coordinate of the top left corner of the rectangle, relative to the viewport, in density-independent pixels.
`y`	`number`	The y-coordinate of the top left corner of the rectangle, relative to the viewport, in density-independent pixels.
`width`	`number`	The width of the rectangle in density-independent pixels.

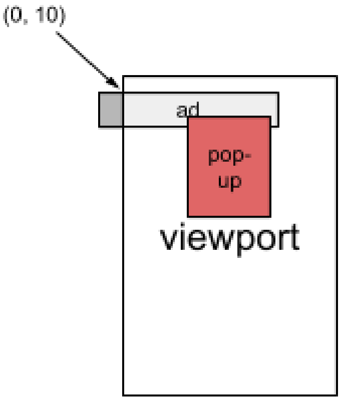
`height`	`number`	The height of the rectangle in density-independent pixels.

`AdView object`

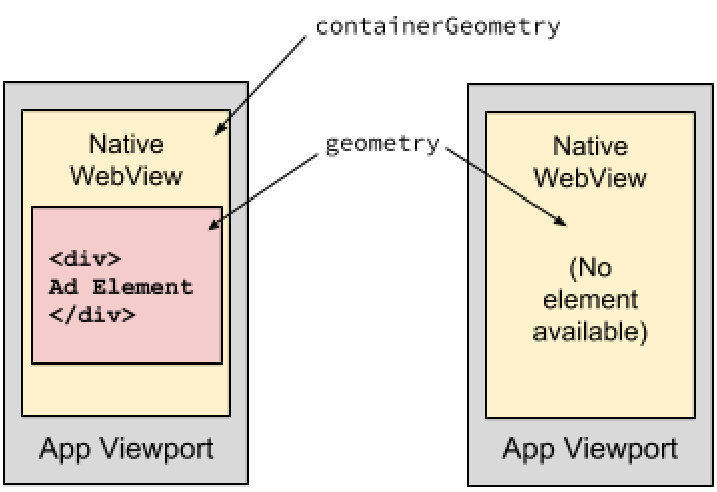
Property Name	Property Type	Description
`percentageInView`	`number`	Value between 0-100 representing the percentage in view of the registered ad view.

`geometry`	[`Rectangle`]()	The rectangle representing the current size and location of the ad. In the case that no creative element at the web-layer level exists or is available to measure, this will measure the geometry of the native-layer webview container. Otherwise this will be the creative element geometry, and the native-layer webview geometry will be available via `containerGeometry`.
		`geometry: {`
		`    x: -20,`
		`    y: 10,`
		`    width: 320,`
		`    height: 50`
		`}`
		

`onScreenGeometry`	[`Rectangle`]()
	`with`
	`Obstructions`
		The rectangle representing the area of the ad that is currently within the viewport, if any, and a list of rectangles which are covering it. This rectangle, after subtracting the list of obstructions, represents the viewable area of the ad.
		If the ad is completely out of viewport (the onscreen area is empty), the `x`, `y`, `width`, and `height` properties should all be set to 0.
		In the case that no creative element at the web-layer level exists or is available to measure, this will measure the geometry of the native-layer webview container. Otherwise this will be the creative element geometry, and the native-layer webview geometry will be available via `onScreenContainerGeometry`.

<table>
<tr>
<td></td>
<td></td>
<td>
<table>
<tr><th>Property Name</th><th>Property Type</th><th>Description</th></tr>
<tr><td><code>x</code></td><td><code>number</code></td><td>The x-coordinate of the top left corner of the within-viewport rectangle.</td></tr>
<tr><td><code>y</code></td><td><code>number</code></td><td>The y-coordinate of the top left corner of the within-viewport rectangle.</td></tr>
<tr><td><code>width</code></td><td><code>number</code></td><td>The width of the within-viewport rectangle.</td></tr>
<tr><td><code>height</code></td><td><code>number</code></td><td>The height of the within-viewport rectangle.</td></tr>
<tr><td><code>obstructions</code></td><td><code>Array &lt;Rectangle&gt;</code></td><td>A list of rectangles which are at least partially covering the onscreen area of the ad.</td></tr>
</table>
<pre>
onScreenGeometry: {
    x: 0,
    y: 10,
    width: 300,
    height: 50,
    obstructions: [
      {
        x: 120,
        y: 40,
        width: 200,
        height: 450
      }
    ]
}
</pre>


</td>
</tr>
<tr>
<td><code>measuringElement</code></td>
<td><code>boolean</code></td>
<td>If true, the geometry of both the creative element inside of the webview and of the native view reprenting that webview are being measured. The geometry of the native-layer webview is provided, in this case, as the <code>containerGeometry</code> and <code>onScreenContainerGeometry</code> properties.</td>
</tr>
</table>

		 containerGeometry geometry Native WebView <div> Ad Element </div> App Viewport Native WebView (No element available) App Viewport
containerGeometry	Rectangle	Only provided if both the native-layer ad view and web-layer ad element exist and are available for measurement. The rectangle representing the current size and location of the native WebView relative to the viewport. In the case that no creative element at the web-layer level exists or is available to measure, this information will instead be provided by the `geometry` property.
onScreenContainerGeometry	Rectangle with Obstructions	Only provided if both the native-layer ad view and web-layer ad element exist and are available for measurement. The rectangle representing the area of the native WebView that is currently within the viewport, if any, and a list of views which are covering it. This rectangle, after subtracting the list of obstructions, represents the viewable area of the ad container. If the ad is completely out of viewport (the onscreen area is empty), the `x`, `y`, `width`, and `height` properties should all be set to 0. In the case that no creative element at the web-layer level exists or is available to measure, this information is instead provided as the `onScreenGeometry` property.
reasons	Array <string>	A set of reasons why the ad is not or only partially viewable. In the majority of cases it is possible to have multiple reasons returned (for example, "obstructed" and "clipped") however only a single reason will be provided for "notFound", "hidden" and "backgrounded". If the ad view is fully in view then the list of reasons will be empty.

<table>
<tr>
<td></td>
<td></td>
<td>
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>notFound</td><td>This indicates an error in which the ad view has not been found within the app view hierarchy.</td></tr>
<tr><td>hidden</td><td>The ad is not viewable because it is currently hidden.</td></tr>
<tr><td>backgrounded</td><td>The application or window has been backgrounded.</td></tr>
<tr><td>viewport</td><td>The ad area is partially or fully outside the viewport (i.e. offscreen).</td></tr>
<tr><td>obstructed</td><td>The ad area is covered by other elements (a list of <code>obstructions</code> should be included in the <code>onScreenGeometry</code>).</td></tr>
<tr><td>clipped</td><td>The ad area has been clipped by a smaller containing parent.</td></tr>
</table>
</td>
</tr>
</table>