Open Measurement SDK

OMID API

Version 1.4 | October 2023

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 information about OMWG is available 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 the 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 to 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.
6/22/2022	1.3	**Better classification**: New ad session types and creative types more clearly defined for robust creative measurement. **Improved transparency:** New Begin to Render impression definition with the capability to use different impression types in a transparent manner. **Simplified integration:** OM SDK activation method made easier to use **Brand safety support**: added capability to declare the content URL in which the ad is being shown to the user. **Audio ad support:** New creative type added for audio and new rules to support audio measurement. **Friendly obstruction support:** Capability to declare friendly obstructions and the reason for the obstruction in the verification data.
9/30/2023	1.4	**Identifying CTV Traffic:** Know the device category, environment and operating system in which a native app is running. **Last Activity:** Signal that an event occurred indicating someone is "still watching". **Display Connection Status:** Understand when the TV display is off, but applications may still be running.

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

DoubleVerify	HUMAN
Google	Nielsen
Integral Ad Science	Oracle Advertising and Customer Experience
IAB Tech Lab	Pandora

Working Group Members

A full list of working group members may be found here:

https://iabtechlab.com/working-groups/open-measurement-working-group/

Table of Contents

Introduction

The following documents the OMID API, which is implemented in the OM SDK. The term "integration partner" has been used throughout this document to include both Ad SDKs and publishers that integrate OM SDK directly in their apps.

OM SDK components

The diagram below shows the various Open Measurement SDK components and where each component has been designed to be integrated;
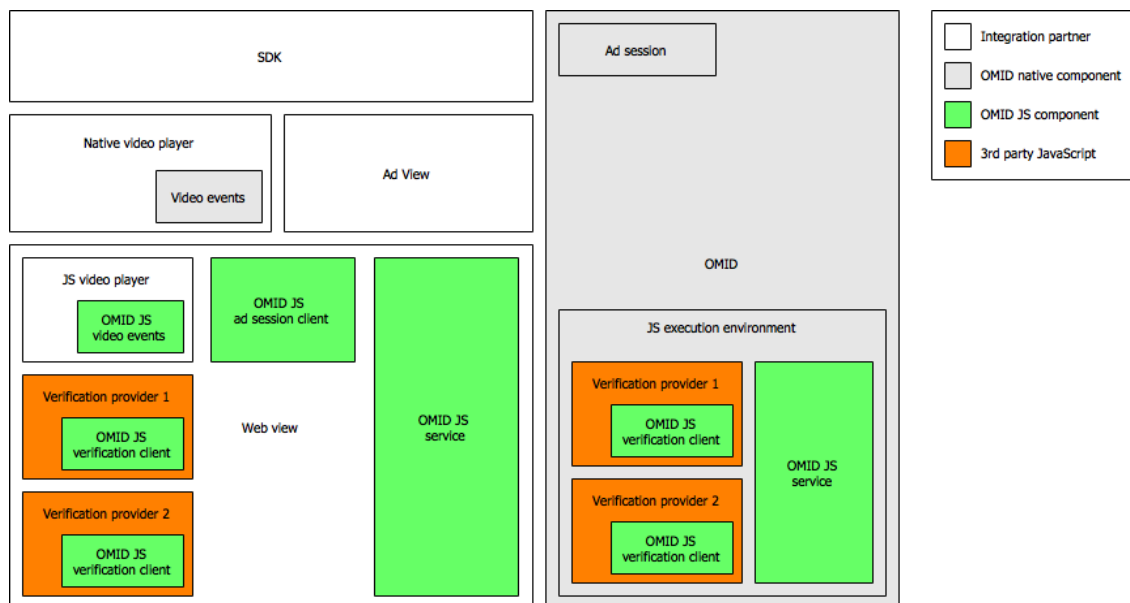
OM SDK supports three ad session types; HTML, JavaScript, and native. When creating HTML or JavaScript ad sessions, OM SDK expects all JS components to be executed within a web view provided by the integration partner, but for native ad sessions OM SDK will create a JS execution environment enabling 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 to 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.

6. JavaScript video ad sessions where the integration partner uses a combination of native SDK and JS SDK to manage the lifecycle. Similar to the native video ad session, the JavaScript video ad session is designed to work with a native video player, however, the integration partner also provides the webview to execute JS components, allowing use of either native SDK or JS SDK for triggering the ad impression event and 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;
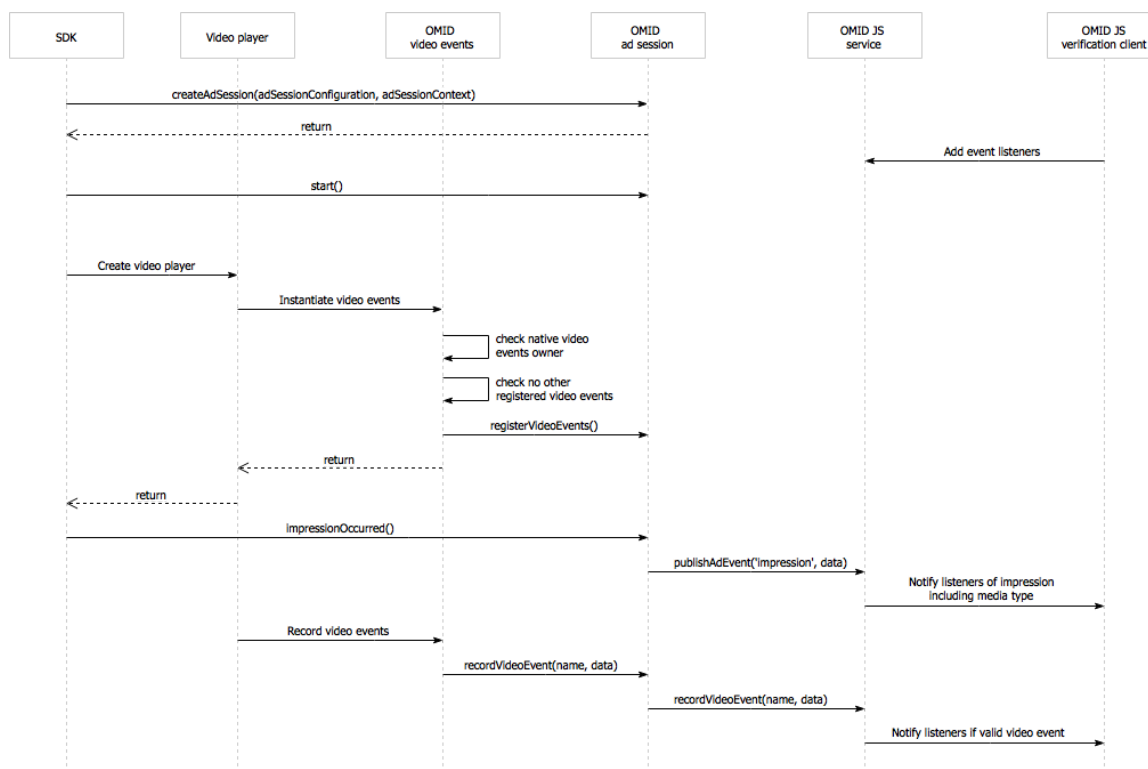
For any integration partners wishing to use the OMID JS video events API this has been designed to be shared as source and for each HTML5 video player to include this within their existing script and minified using their existing processes.

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 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 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 prepended 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 `<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 in more detail and 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 completed loading 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.
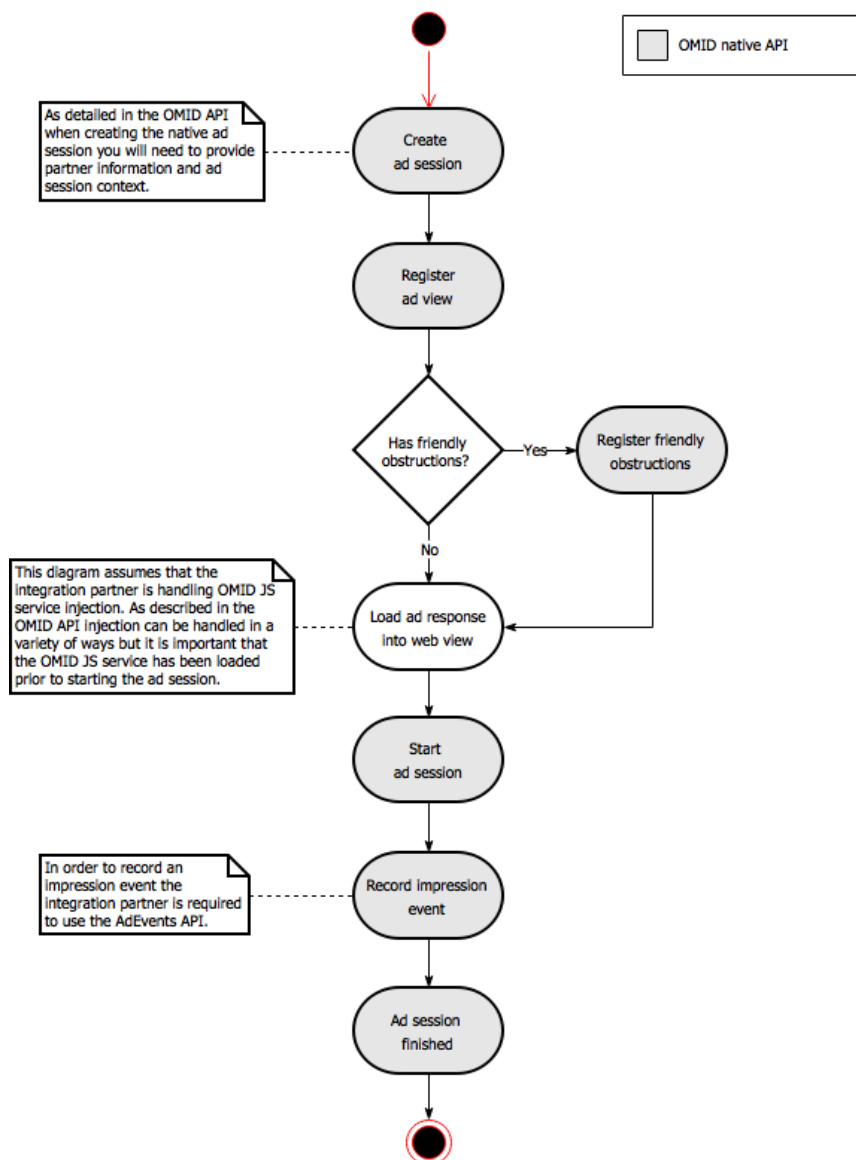
Also note that ending an OMID ad session sends a message to the verification scripts running inside the webview supplied by the integration. So that the verification scripts have enough time to handle the "sessionFinish" event, the integration must maintain a strong reference to the webview for at least 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.
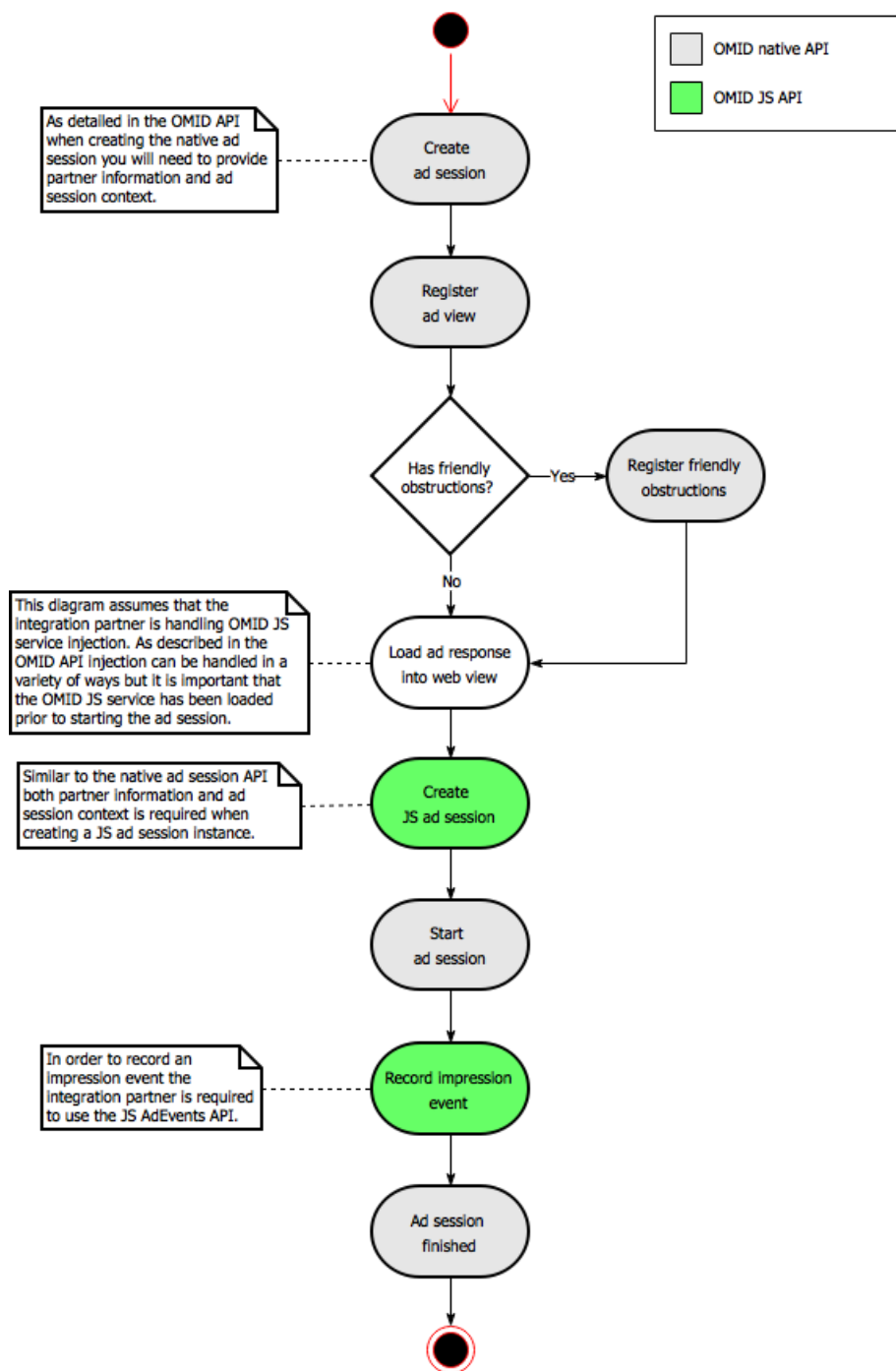
Display ad session with no contributing JS ad session

The below diagram demonstrates the OMID display ad session lifecycle where the integration partner wishes to only use the full native OMID API.
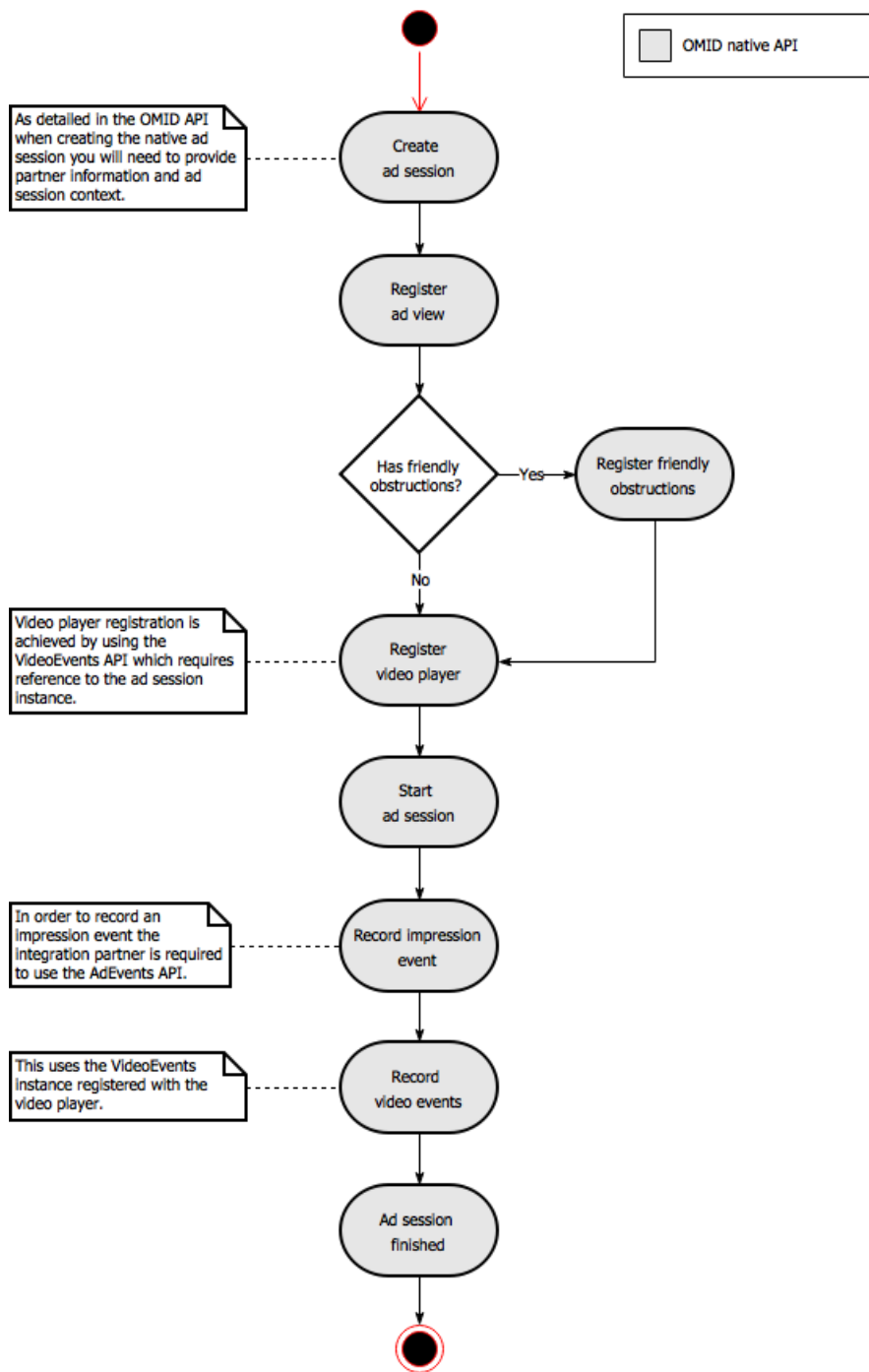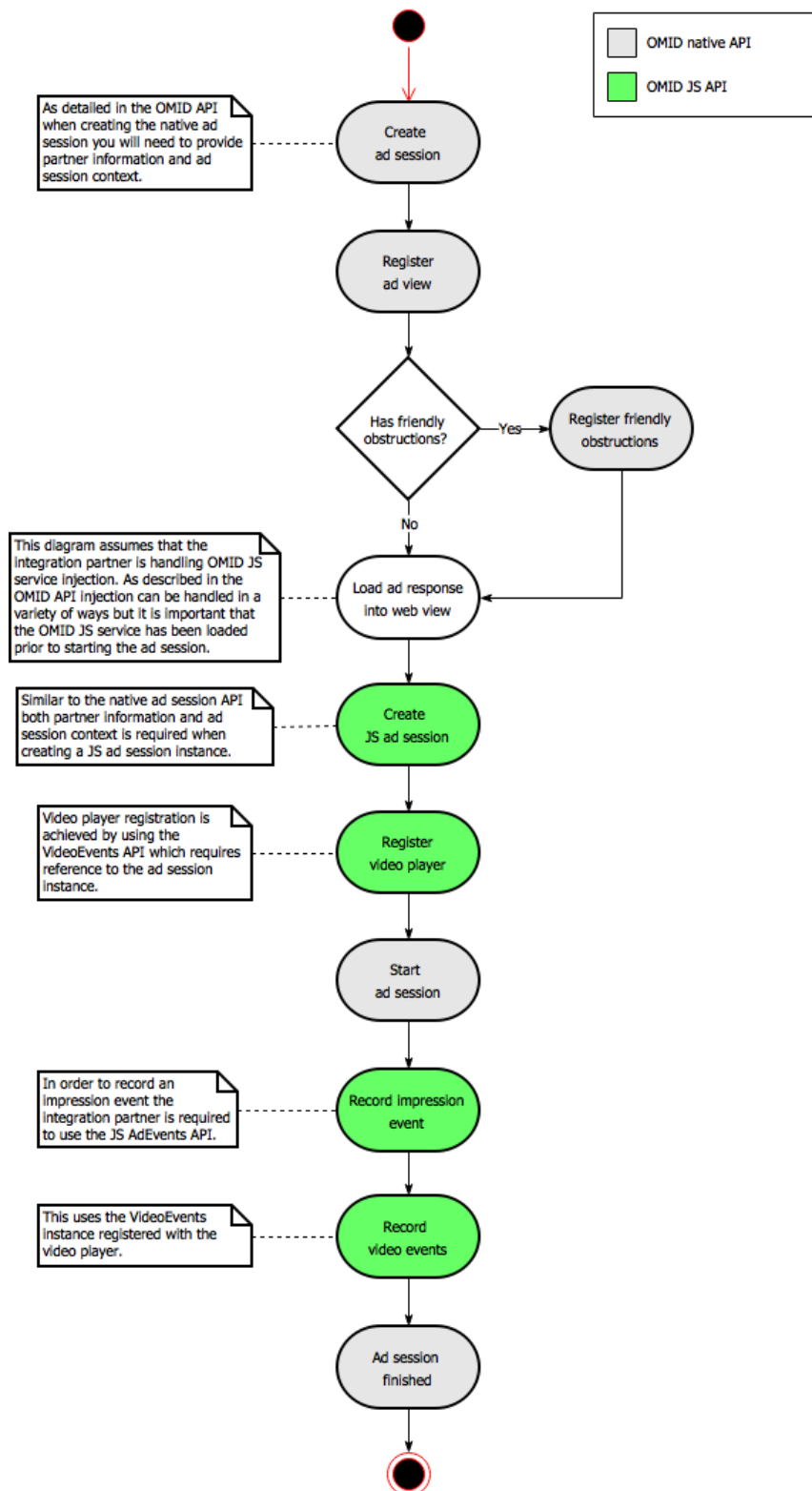
As detailed in the OMID API when creating the native ad session you will need to provide partner information and ad session context.

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.

In order to record an impression event the integration partner is required to use the AdEvents API.

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.

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

OMID native API
OMID JS API

Video ad session with native video player

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

As detailed in the OMID API when creating the native ad session you will need to provide partner information and ad session context.

Video player registration is achieved by using the VideoEvents API which requires reference to the ad session instance.

In order to record an impression event the integration partner is required to use the AdEvents API.

This uses the VideoEvents instance registered with the video player.

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 / Android TV 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. Check if OMID has already been activated by calling `Omid.isActive()`.
3. If not activated, execute `Omid.activate(applicationContext)`. **In Android TV applications, OMID should be activated in** `MyApplication.onCreate()`.

Publish activity events (required for Android TV integrations)

1. After activating Omid in MyApplication.onCreate(), the integration should also call Omid.updateLastActivity().
2. Signal to OM SDK when a user interacts prior viewing an ad by calling Omid.updateLastActivity().

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 by 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

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 media events (video and audio only)

For HTML video ad sessions this will be handled by the JS ad session.
1. Create `MediaEvents` object specifying the `AdSession` instance.
2. Update media player implementation to trigger the appropriate media events during content loading / playback.

Thread Safety

OMID library functions must be called only from the main UI thread of the application.

API Reference

https://docs.iabtechlab.com/omsdk-docs-1.4/android/index.html

iOS / tvOS OMID Library API

Usage

Set up OMID

1. Verify that OMIDSDK class exists.
2. Check `OMIDSDK.isActive` to determine if OM SDK has already been activated.
3. If not, call -`[OMIDSDK activate]`. In tvOS applications, OMID should be activated in `[MyAppDelegate application:didFinishLaunchingWithOptions:]`.

Publish activity events (required for tvOS integrations)

1. Signal to the SDK when a user interacts with the app by calling `[[OMIDSDK sharedInstance] updateLastActivity]`.

Set up an Ad Session

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. Start the ad session.

Report on the Ad Lifecycle and Media Ad Events:

1. Create `OMIDAdEvents` and `OMIDMediaEvents` objects if required.
2. Call AdEvents and MediaEvents methods as needed.

Thread Safety

OMID library functions must be called only from the main UI thread of the application.

API Reference

https://docs.iabtechlab.com/omsdk-docs-1.4/ios/index.html

OMID JS Ad Session Client API

https://docs.iabtechlab.com/omsdk-docs-1.4/js/index.html

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

https://docs.iabtechlab.com/omsdk-docs-1.4/js/Partner.html

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.

https://docs.iabtechlab.com/omsdk-docs-1.4/js/VerificationScriptResource.html

Context

https://docs.iabtechlab.com/omsdk-docs-1.4/js/Context.html

OmidVersion

https://docs.iabtechlab.com/omsdk-docs-1.4/js/OmidVersion.html

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.

https://docs.iabtechlab.com/omsdk-docs-1.4/js/AdSession.html

AdEvents

VastProperties

Constructor Summary

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

Method Summary

No public methods available.

VideoPlayerState

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

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.

MediaEvents

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.

https://docs.iabtechlab.com/omsdk-docs-1.4/js/MediaEvents.html

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

https://docs.iabtechlab.com/omsdk-docs-1.4/js/VerificationClient.html

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.
mediaType	string	The media type measured in the ad session.
creativeType	string	The type of ad creative being measured in the ad session.
impressionType	string	Impression type makes it easier to understand discrepancies between measurers of the ad session.

supportsLoadedEvent	boolean	Whether the loaded event is supported.
contentUrl	string	On the web, the URL of the top-level web page. In apps Android deep link or iOS universal link.
lastActivity	object	The most recent user interaction; either provided by the integrator or measured directly by OM SDK. "lastActivity":{"timestamp":1688029344570}

Context Object

Property Name	Property Type	Description
apiVersion	string	The version of the OMID JS Verification Client API provided ("1.0" for this document).
environment	string	<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>
accessMode	string	<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" accessMode implies both that the verification code has access to the creative element and that a reference to that element will be provided via either videoElement or slotElement.</td></tr></table>
videoElement	HTMLVideoElement	Required for all "full" accessMode linear video ads, or any ad 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.

slotElement	Element	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.
adSessionType	string	<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><tr><td>javascript</td><td>Control of the ad session is directed from the native layer but it uses webviews for running JavaScript code.</td></tr></table>
adServingId	string	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.
transactionId	string	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.
podSequence	string	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.
adCount	number	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
omidNativeInfo	Object	Only present when a native layer is involved in the ad session. All properties are required when present.

Property Name	Property Type	Description
`partnerName`	`string`	The name of the native layer OMSDK integration partner.
`partnerVersion`	`string`	The version of the native layer OMSDK integration partner.

```
omidNativeInfo: {
    partnerName: 'exampleNativeSDK',
    partnerVersion: '1.0.0'
}
```

`omidJsInfo`	Object	

Property Name	Property Type	Description
`omidImplementer`	`string`	The name of the OMID provider. For OMSDK integrations this is always "omsdk".
`serviceVersion`	`string`	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`.
`sessionClientVersion`	`string`	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.
`partnerName`	`string`	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.
`partnerVersion`	`string`	The version of the code provided by the party from `partnerName`, if

		partnerName was provided.

```
omidJsInfo: {
    omidImplementer: 'omsdk',
    serviceVersion: '1.0.0',
    sessionClientVersion: '1.0.0',
    partnerName: 'exampleJsSdk',
    partnerVersion: '3.4.2'
}
```

app	Object	Required for OMSDK mobile app integrations, otherwise not provided.
		
		Provides details about the running app and native OMSDK version.
		
		<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>
		`app: {`
		`  libraryVersion: '1.0.0',`
		`  appId: 'com.bundle.app'`
		`}`

deviceInfo	Object	Required for OMSDK mobile app integrations, otherwise not provided. Provides details about the mobile device. <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", "tvOS", "Android").</td></tr><tr><td>osVersion</td><td>string</td><td>Operating system version ("11.1.2").</td></tr></table>
supports	Array \<string\>	A list of optional features which may be implemented in the current environment.

		Value	**Description**
		clid	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.
customReferenceData	string	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.	
friendlyToTop	boolean	Whether the SDK has access to the top window.	
deviceCategory	string	Category of device OM SDK is running on. The possible values are "ctv", "desktop", "mobile", and "other". If the category cannot be determined (such as in the OM SDK for Web) the property will be left undefined.	

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

Property Name	Property Type	Description	
errorType	string	High level error type.	
		Value	**Description**
		video	Video-related rendering or loading errors
		generic	Catch-all for other issues
message	string	Description of the session error.	

sessionFinish

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

Event Data

None.

Lifecycle and Metric Events

Verification code can subscribe to these events using the <u>addEventListener</u> 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

Property Name	Property Type	Description
mediaType	string	The media type measured for this impression. | Value	Description
---	---
display	Used for display ad impressions.
video	Used for video ad impressions.	
creativeType	string	The type of ad creative being measured in the ad session. | Value	Description
---	---
definedByJavaScript	Creative type needs to be set by JavaScript session script.
htmlDisplay	Creatives measured using display methodology that are trafficked as HTML.
nativeDisplay	Creatives measured using display methodology that are trafficked as JSON or other format for structured data.
video	Creatives measured using video methodology
audio	Creatives measured using audio methodology	
impressionType	string	Impression type makes it easier to understand discrepancies

between measurers of the ad session.

Value	Description
definedByJavaScript	Impression type needs to be set by JavaScript session script.
unspecified	The integration is not declaring the criteria for the OMID impression.
loaded	The integration is using count-on-download criteria for the OMID impression.
beginToRender	The integration is using begin-to-render criteria for the OMID impression.
onePixel	The integration is using one-pixel criteria for the OMID impression.
viewable	The integration is using viewable criteria (1 second for display, 2 seconds for video) for the OMID impression.
audible	The integration is using audible criteria (2 seconds of playback with non-zero volume) for the OMID impression.
other	The integration's criteria uses none of the above for the OMID impression.

`videoEventAdaptorType`	`string`	Provided only for OMSDK integrations on impressions where the `mediaType` is "video".

Value	Description
jsCustom	Used when a JS event adaptor is used.
nativeCustom	Used when a native event adaptor is used.

`videoEventAdaptorVersion`	`string`	Provided only for OMSDK integrations where videoEventAdaptorType is also provided. This is the version of the video event adaptor code used.
`viewport`	`Object`	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,`
		`    height: 480`
		`}`

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

loaded

For display ads dispatch the loaded event to signal that the ad has loaded and is ready to display and for video and audio ads dispatch the loaded event when 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

Property Name	Property Type	Description
skippable	boolean	Whether the ad can be skipped by the user.
skipOffset	number	Required when `skippable` 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 `skipoffset` attribute from VAST.
autoPlay	boolean	Whether the ad playback will be automatically started without input from the user.
position	string	<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></table>

		standalone	The ad plays independently of any video content.
mediaType	string	The media type measured in the ad session.	
creativeType	string	The type of ad creative being measured in the ad session.	
impressionType	string	Impression type makes it easier to understand discrepancies between measurers of the ad session.	

media

This is a special event type which is shorthand to allow subscription to many media playback-related events with a single call to addEventListener. The video event type is also accepted as an alias of media. Triggered events will never contain media or 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 media event type. **Verifiers should take care not to unintentionally double subscribe to these events.**

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

Example: Subscribing to the video event

```
omidClient.addEventListener('media', function(evt) {
  switch (evt.type) {
    case 'start':
      handleMediaStart(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.

start

Media-only event. The player began playback of the media creative. Corresponds to the VAST `start` event.

Event Data

Property Name	Property Type	Description
duration	number	The duration of the media 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. Maintained for compatibility with versions before 1.3; prefer `mediaPlayerVolume`, which is an alias with the same value.
mediaPlayerVolume	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

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

Event Data

None.

midpoint

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

Event Data

None.

thirdQuartile

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

Event Data

None.

complete

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

Event Data

None.

pause

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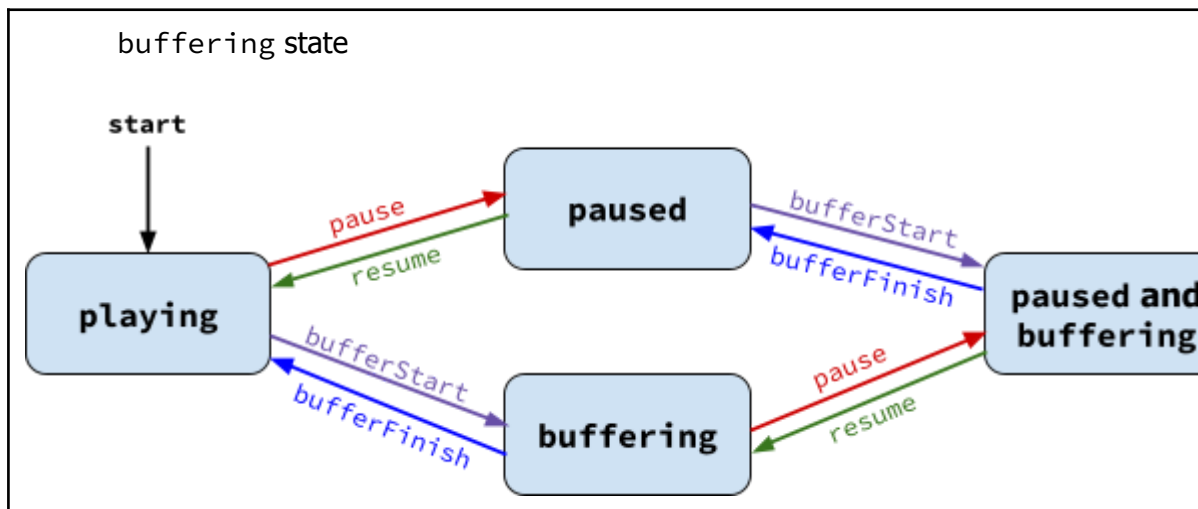
Event Data

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

---

resume

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

Event Data

None.

bufferStart

Media-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

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

Event Data

None.

skipped

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

Event Data

None.

volumeChange

Media-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. Maintained for compatibility with versions before 1.3; prefer mediaPlayerVolume, which is an alias with the same value.
mediaPlayerVolume	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

Media-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

Property Name	Property Type	Description
state	string	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 **not** the state of the app or browser window.</td></tr></table>

		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.

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

Property Name	Property Type	Description
interactionType	string	The type of interaction which triggered the event. Possible interaction types are as follows:

Value	Description
click	The user clicked to load the ad's landing page.
invitationAccept	The user engaged with ad content to load a separate experience.

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

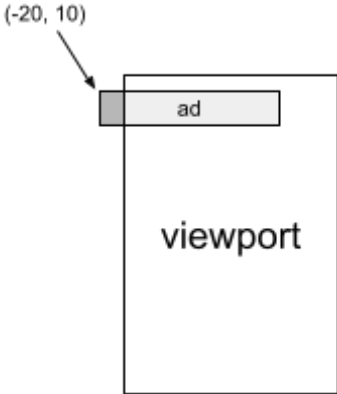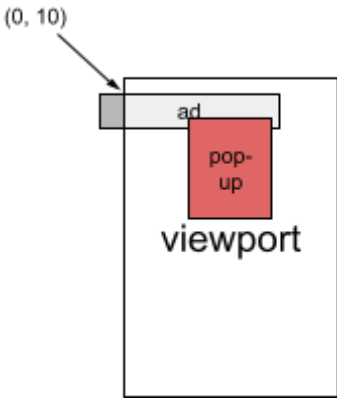| Property Name	Property Type	Description
`viewport`	`Object`	The state of the viewport (the mobile device screen or the browser window) at impression time. <table><tr><th>Property Name</th><th>Property Type</th><th>Description</th></tr><tr><td>`width`</td><td>`number`</td><td>The viewport width.</td>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

Rectangle Object

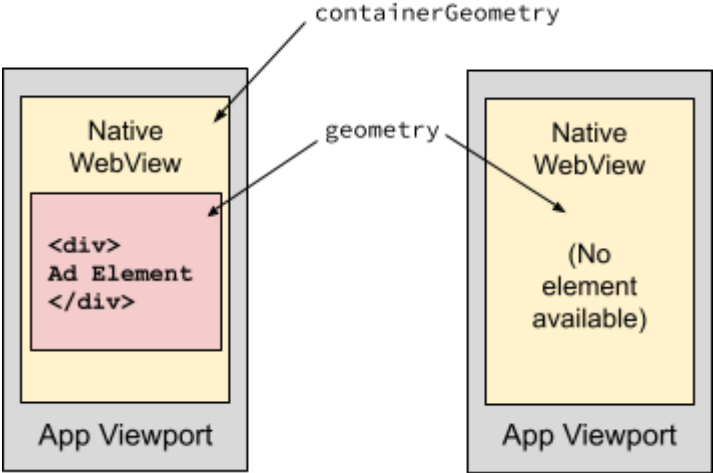An object representing the size and location of a rectangular area.

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`.

<table>
<tr><td colspan="3">

```
geometry: {
    x: -20,
    y: 10,
    width: 320,
    height: 50
}
```


</td></tr>
</table>

onScreenGeometry	Rectangle with Obstructions and friendlyObstructions	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`.

Property Name	Property Type	Description
x	number	The x-coordinate of the top left corner of the within-viewport rectangle.
y	number	The y-coordinate of the top left corner of the within-viewport rectangle.
width	number	The width of the within-viewport rectangle.
height	number	The height of the within-viewport rectangle.
obstructions	Array<Rectangle>	A list of rectangles which are at least partially covering the

		onscreen area of the ad.
`friendlyObstructions`	Array [Rectangle](#) with `friendlyObstructions`	Provides an array using the rectangle object values along with friendly obstructions metadata.

```
onScreenGeometry: {
    x: 0,
    y: 10,
    width: 300,
    height: 50,
    obstructions: [
      {
        x: 120,
        y: 40,
        width: 200,
        height: 450
      }
    ]
}
```


`pixelsInView`	number	The viewable pixels of the registered ad view.
`measuringElement`	boolean	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 `containerGeometry` and `onScreenContainerGeometry` properties.

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.
onScreenContainer Geometry	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", "backgrounded" and "noWindowFocus", "backgrounded" and "noOutputDevice") however only a single reason will be provided for "notFound", "viewport" and "hidden". If the ad view is fully in view then the list of reasons will be empty.

		<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 `obstructions` should be included in the `onScreenGeometry`).</td></tr><tr><td>clipped</td><td>The ad area has been clipped by a smaller containing parent.</td></tr><tr><td>noWindowFocus</td><td>The ad views without window focus.</td></tr><tr><td>noOutputDevice</td><td>This indicates that there is no display connection or audio route present.</td></tr></table>
`declaredFriendly Obstructions`	`number`	The number of friendly obstructions declared by the integrator for the ad session in progress.