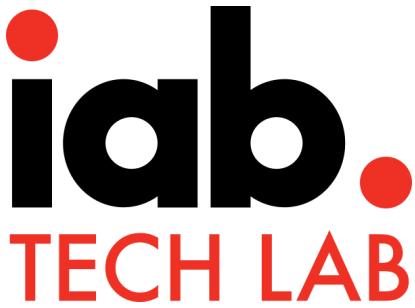

OMID Web Implementation

Guide for Third Parties

Version 1.0 | June 2018

Please note that as of December 2020, OM SDK for Web Video implementations should use the OM SDK for Web Video, available for download at:

https://tools.iabtechlab.com/omsdk

Table of Contents

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) SDK.

The OM SDK consists of a native library for iOS & Android as well as a JavaScript API named Open Measurement Interface Definition (OMID) that is integrated by the Integration Partners (mobile app or Ad SDK developers) . OMID API version 1.2 [INSERT URL] and above supports web browser environments and can be used to measure viewability on the web. Though IAB Tech Lab has not yet developed the code libraries for implementation of OMID version 1.2 for web browsers, the API is being made available to support publishers or video players who may want to implement their own version and offer OMID verification for third party measurement providers.

The collection, processing, analysis, and reporting of information surfaced by the OM SDK is the responsibility of the Measurement Provider using their respective JavaScript tag that is served with the ad creative.

OM SDK and OMID are developed and managed by the Open Measurement Working Group

Background

Overview

The IAB TechLab is facilitating the development of the Open Measurement SDK (OMSDK), which is a library intended to directly integrate with mobile in-app ad SDKs in order to enable viewability measurement and general verification. At the same time, Digital Video Technical Working Group has been seeking a standard to replace VPAID and enable verification on web video for VAST 4+. As OMSDK covers in-app video, an optimal web standard would share the underlying API and allow verification vendors to easily measure both web and in-app video.

While OMSDK is an implementation and not a standard, the interface it exposes to verification code -- a subset of the Open Measurement Interface Definition (OMID) -- can be used as such.

Currently OMSDK is an app-only library, and while a web version is on the roadmap, there is a need for web players to be able to independently develop OMID compatibility in the interim. This document provides guidance for third-parties web players or SDKs to implement OMID support such that they will interoperate with verification code designed for OMSDK. Implementers should also consult the VAST 4.1 specification for the latest features, including clarifications and updates to the verification sections intended to support this work.

VAST 4+ Support with Backwards Compatibility

In VAST 4, measurement code and relevant per-vendor metadata appears within the `<AdVerifications>` node, allowing multiple vendors to measure the same impression. Earlier versions of VAST will be able to support this as well, by including `<AdVerifications>` as an `<Extension>`.

VPAID

VPAID was not originally designed to handle the verification use case, and as such had several unfortunate properties (taking rendering control away from publishers, forcing rendering responsibility onto verifiers, making early loading of verification code almost impossible, putting

verification code in the critical path of creative playback with multiple verifiers all loading in serial, etc). This has caused many negative associations with the name VPAID among publishers, advertisers, and users, as well as confusion as to why support for a spec intended for interactivity was required to make inventory verifiable. The design of VAST 4+OMID aims to have a standard focused purely on verification, solving many of the issues with the VPAID paradigm.

Terminology

Player: Code used by the publisher (including any third-party integrated components) to handle video playback, ad fetching, user interaction, etc. This document uses the term "player" throughout, although this is usually meant to refer more generally to the party implementing OMID and interacting with verification code. In practice, for example, this may more commonly be an advertising SDK which is integrated with the actual web player.

Verification Code/Resource: Executable code provided by the parties wishing to monitor ad playback (generally from a third-party verification company), for the purpose of live measurement at impression time. It collects information regarding video playback, runtime environment, and viewability/audibility.

VAST Integration

Loading Verification Resources

From VAST 4.1+

For video impressions delivered through VAST 4.1+ documents, the `<AdVerifications>` element should be used to pass the resource URLs for the desired verification code. Each `<Verification>` should be loaded (or considered for loading per vendor whitelisting), including those from the `<Wrapper>` of any intermediate VAST.

Example: AdVerifications in VAST 4.1

```
...
<AdVerifications>

  <Verification vendor="abc.com-omid">
    <JavaScriptResource apiFramework="omid" browserOptional="true">
      <![CDATA[https://abc.com/omid.js]]>
    </JavaScriptResource>
    <VerificationParameters><![CDATA[...]]></VerificationParameters>
    <TrackingEvents>
      <Tracking event="verificationNotExecuted">
        <![CDATA[https://abc.com/omid_reject?r=[REASON]]]>
      </Tracking>
    </TrackingEvents>
  </Verification>

  <Verification vendor="xyz.com-omidpub">
    <JavaScriptResource apiFramework="omid" browserOptional="true">
      <![CDATA[https://xyz.com/omid-verify.js]]>
    </JavaScriptResource>
    <VerificationParameters><![CDATA[...]]></VerificationParameters>
  </Verification>

</AdVerifications>
...
```

Both `omid.js` and `omid-verify.js` scripts should be picked up by the player.

From VAST pre-Version 4.1

For older version VAST documents, verification code should be loaded via `<Extension>`, where the root is an `<AdVerifications>` with the same schema as the VAST 4.1 element.

Example: AdVerifications in VAST pre-Version 4.1

```
...
<Extensions>
  <Extension type="AdVerifications">
    <AdVerifications>
      <Verification vendor="verification.com-omid">
        <JavaScriptResource apiFramework="omid" browserOptional="false">
          <![CDATA[https://verification.com/omid_verification.js]]>
        </JavaScriptResource>
        <VerificationParameters><![CDATA[...]]></VerificationParameters>
      </Verification>
    </AdVerifications>
  </Extension>
</Extensions>
...
```

The `<AdVerifications>` element under the `<Extension>` is otherwise identical to one found in a VAST 4.1+ document.

Loading Data From VAST

VerificationParameters

The `<VerificationParameters>` element contains a CDATA-encoded string of arbitrary format (i.e. determined by each vendor) and should be passed along verbatim to each script as part of the `sessionStart` event. `<VerificationParameters>` data from a vendor should be matched to the `vendorKey` passed to `registerSessionObserver`.

apiFramework

The `apiFramework` attribute of the `<JavaScriptResource>` for OMID-compatible scripts will be set to "omid".

The vendor Attribute

The vendor attribute on the `<Verification>` element should be unique per verification resource and should match the value passed as the `vendorKey` argument of `registerSessionObserver`.

The browserOptional Attribute

`<JavaScriptResource>` elements marked with `browserOptional="true"` can execute in non-browser environments and do not require the presence of certain web-based JavaScript functionality. In the context of OMID for web, where the existence of a browser is implied, resources with `browserOptional` set to either value are always supported. If a Verification provides both `browserOptional="true"` and `browserOptional="false"` `<JavaScriptResource>` elements, the `browserOptional="false"` version should be preferred, so it can take full advantage of browser resources.

API Implementation

Execution Access

While the OMSDK project supports the ability to optionally execute verification code within a sandbox, third-party OMID web implementations should almost exclusively run verification resources such that the `<video>` element (or in the absence of such, the relevant element where media is rendered) is directly accessible. Implementations where this is not the case will be considered unmeasurable by most verification vendors without prior agreement. Data collected in sandboxed non-OMSDK environments are unusable for reporting on accredited metrics unless certified for implementation and measurement provider has completed MRC audit using the data. Alternatively, publishers may choose to participate in the Open Measurement Working Group to support development of and integrate with OMSDK on web in order to enable sandboxing.

Regardless of choice, OMID implementers should provide the same level of access to all verification resources loaded for a given session.

Verification Vendor Whitelisting

Publishers may choose to employ a vendor whitelist when deciding which verification resources to execute. When this is the case, the player must support the new tracking event `"verificationNotExecuted"`, which appears as a new `<TrackingEvent>` element under `<Verification>` in VAST 4.1.

For example, a whitelist implementation might look as follows:

Trusted Verifiers	Owned Domains
Verification Company A	verificationbya.com, static.averification.net
Verification Company B	bverify.com

Given a VAST containing the following `<AdVerifications>`:

```
<AdVerifications>
  <Verification vendor="verificationbya.com-omid">
    <JavaScriptResource apiFramework="omid">
      <![CDATA[https://static.averification.net/omid.js]]>
    </JavaScriptResource>
  </Verification>
  <Verification vendor="unknown-omid">
    <JavaScriptResource apiFramework="omid">
      <![CDATA[https://unknown-domain.com/untrusted.js]]>
    </JavaScriptResource>
    <TrackingEvent>
      <Tracking event="verificationNotExecuted">
        <![CDATA[https://u.com/omid_error?r=[REASON]&i=[ADSERVINGID]
                 &d=[DOMAIN]&p=[PLAYERUA]]]>
      </Tracking>
    </TrackingEvent>
  </Verification>
</AdVerifications>
```

The player would execute the `omid.js` script provided by Verification Company A as normal. The resource from the unrecognized "unknown-domain.com" would not, and the URI under the `"verificationNotExecuted"` `<Tracking>` would be pinged with **[REASON]** set to reason code 1, indicating that the verification resource was rejected by the publisher. All VAST 4.1 macros (including **[ADSERVINGID]**, **[DOMAIN]**, and **[PLAYERUA]**) should be supported in order to provide accounting insight and actionability to such events.

Ad Session ID

The player should generate a GUID for each unique ad session to share with verification code for tracking purposes. This value is provided as the `adSessionId` property on every event sent for that session.

Ad Session Lifecycle

The lifecycle of an ad session proceeds as follows. This process may be started well before video playback begins. In fact, it is recommended to begin loading verification resources as early as is feasible. Note that, unlike VPAID-style verification in which loading verification code would immediately trigger playback, there is little downside to pre-loading OMID verification code. Scripts will simply register handlers and wait for the player to initiate playback. So whenever possible (e.g. mid- or post-roll), players should consider loading verification resources before playback While it is supported to start playing the ad immediately after step 5 regardless

of verification state, the best outcome (which minimizes discrepancies) is one where verification code is fetched early and is ready before the ad begins playing.

1. The player collects the list of `<Verification>` script resources from the `<AdVerifications>` of any VASTs that were loaded for the current impression (including those from intermediate `<Wrapper>` VASTs).
2. The player decides which resource to execute for each `<Verification>`, if any. If there is no compatible or acceptable resource for a given `<Verification>`, any provided `"verificationNotExecuted"` `<Tracking>` events should be fired immediately.
3. For each resource from step 2, the player creates or chooses an iframe (or iframes) and initializes the code which exposes an OMID interface (i.e. `window.omid3p`).
4. The player loads each verification resource accepted from step 2 inside one of these iframes.
5. The player is free to begin creative playback any time after this point. Note that loading resources from step 4 must only be started; the player is not required to wait for each resource to finish loading.
6. Each verification resource initializes by calling `registerSessionObserver`, and by subscribing to any events it's interested in via `addEventListener`.
7. The player sends the `sessionStart` event to any registered observers as soon as all the relevant information is available (this may be well before actual playback starts). This event (as well as all subsequent events) contains the unique Ad Session ID for this session.
8. The player sends any relevant video lifecycle events to registered listeners, in chronological order. This may include cached, timestamped events that occurred before the caller was registered, allowing verification resources that may have loaded late to "catch up" on critical events.
9. The player sends the `sessionFinish` event to indicate session termination.
10. After a delay, the player cleans up verification resources.

Session Cleanup

After the `sessionFinish` event is fired, whenever possible the player should give verification code sufficient time to ensure any end-of-session measurement and reporting is completed before cleaning up containing iframe contexts. The exact value is left to the player, but something on the order of seconds is recommended. This allows time for reporting pings to successfully send without being cancelled by the browser, in order to minimize discrepancies.

Implementation Guidance

The specifics of the OMID JS Verification Client API that need to be implemented can be found in the API document here (https://iabtechlab.com/omsdk). Below is some guidance on implementing that API specifically for third-party web players or SDKs.

Exposing the OMID Interface

The player exposes the OMID interface by providing a global object in each iframe context where verification code is loaded, accessible as **window.omid3p**. See the OMID JS Verification Client API document for details on individual methods. Third-party web implementers are only responsible for providing implementations the following methods, exposed as properties on the **omid3p** object.

```
registerSessionObserver(observer, vendorKey)
addEventListener(type, listener)
```

Example: Boostrapping OMID and Verification Code

```
function createOmidFrames(verificationResources) {
  verificationResources.forEach((verificationResource) => {
    const frame = document.createElement('iframe');
    frame.style.display = 'none';
    document.body.appendChild(frame);
    const frameWin = frame.contentWindow;
    // Expose OMID.
    frameWin.omid3p = {
      'registerSessionObserver': registerSessionObserverImpl,
      'addEventListener': addEventListenerImpl,
    };
    // Load verification script.
    const verificationScript = frameWin.document.createElement('script');
```

```
    verificationScript.src = verificationResource;
    frameWin.document.body.appendChild(verificationScript);
  });
}
```

Event Caching

The player should cache all triggered events after verification scripts have been loaded into their iframes. Upon subscription to an event, the player should invoke the given handler with each previously fired event of that type in chronological order. This allows verification code which may have loaded late to "catch up", and prevents situations where critical events are missed (e.g. start of playback), which may result in reporting discrepancies.

Implementation of this feature may be skipped if the player chooses to wait for each verification vendor to call `registerSessionObserver` before starting ad playback.

Session Context

The context object in the event data of the `sessionStart` event contains static properties of the playback environment. For third-party web implementers, the following properties should be included the context:

Property Name	Property Type	Value to provide
apiVersion	string	"1.0" This is the version of the OMID JS Verification Client API that has been implemented.
environment	string	"web"
accessMode	string	"full"
adSessionType	string	"html"
videoElement	HTMLVideo Element	Required for all linear video ads, or any ad where a video element is the main focal point of the creative. For VAST-served creatives that do not use HTML5 video at all,

<table>
<tr><td></td><td></td><td>slotElement may be used instead.</td></tr>
<tr><td>slotElement</td><td>Element</td><td>Required for non-linear or interactive ads where no HTMLVideoElement is used in the creative. It should not be provided for standard linear video ads; videoElement should be passed instead.</td></tr>
<tr><td>omidJsInfo</td><td>Object</td><td>Information about the provider/implementer of OMID. The provider in this case is the player or SDK that is implementing the OMID interface and interacting with verification code.

This object contains the following properties:
<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 code or library which is implementing and providing OMID.</td></tr>
<tr><td>serviceVersion</td><td>string</td><td>The version of the code from omidImplementer.</td></tr>
<tr><td>partnerName</td><td>string</td><td>If the player or ad SDK does not directly implement OMID, but instead uses a library to provide that functionality, the name of the player or SDK should be listed here.</td></tr>
<tr><td>partnerVersion</td><td>string</td><td>The version of the code from partnerName.</td></tr>
</table>
</td></tr>
</table>

Example: Firing the sessionStart event

```
function registerSessionObserverImpl(observer, vendorKey) {
  registeredSessionObservers.push({
    'observer': observer,
```

```
    'verificationParameters': getParametersFromVastForVendor(vendorKey)
  });
}

function newContext() {
  return {
    'apiVersion': '1.0',
    'environment': 'web',
    'accessMode': 'full',
    'videoElement': document.getElementById('video-player-element'),
    'adSessionType': 'html',
    'adServingId': 'c532d16d-4d7f-4449-bd29-2ec0e693fc86',
    'adCount': 1,
    'omidJsInfo': {
      'omidImplementer': 'example-sdk',
      'serviceVersion': '4.1.1',
    },
  };
}

// Fire the sessionStart event to each registered session observer.
registeredSessionObservers.forEach(function(sessionObserver) {
  const observerCallback = sessionObserver.observer;
  const parameters = sessionObserver.verificationParameters;
  observerCallback({
    'adSessionId': 'ad994cf4-1cf2-4ffc-88aa-d2a6e69881eb',
    'type': 'sessionStart',
    'timestamp': 123456789,
    'data': {
      'verificationParameters': parameters,
      'context': newContext(),
    }
  });
});
```