iab. TECH LAB

Video Ad Serving Template (VAST)

Released December 2022

This document is available online at:
https://iabtechlab.com/audio-video/tech-lab-digital-video-suite/

Drafted by the Digital Video Technical Standards Working Group.

The IAB Tech Lab lead on this initiative is Katie Stroud.

About IAB Technology Laboratory

The IAB Technology Laboratory (Tech Lab) is a non-profit consortium that engages a member community globally to develop foundational technology and standards that enable growth and trust in the digital media ecosystem. Comprised of digital publishers, ad technology firms, agencies, marketers, and other member companies, IAB Tech Lab focuses on improving the digital advertising supply chain, measurement, and consumer experiences, while promoting responsible use of data. Its work includes the OpenRTB real-time bidding protocol, ads.txt anti-fraud specification, Open Measurement SDK for viewability and verification, VAST video specification, and DigiTrust identity service. Board members include ExtremeReach, Facebook, Google, GroupM, Hearst Digital Media, Index Exchange, Integral Ad Science, LinkedIn, LiveRamp, MediaMath, Microsoft, Oracle Data Cloud, Pandora, PubMatic, Quantcast, Rakuten Marketing, Telaria, The Trade Desk, Verizon Media Group, Xandr, and Yahoo! Japan. Established in 2014, the IAB Tech Lab is headquartered in New York City with staff in San Francisco, Seattle, and London. Learn more at https://www.iabtechlab.com.

TABLE OF CONTENTS

Executive Summary

The Video Ad Serving Template or VAST is a template for structuring ad tags that serve video and audio ads to media players. Using an XML schema, VAST transfers important metadata about an ad from the ad server to a media player. Initially launched in 2008, VAST has since played an important role in the growth of the digital video and audio marketplace.

The early days of video consisted mostly of shared videos and other user-generated content. Success in monetizing this content with ads has produced the resources to improve the digital video marketplace. However, digital video has met a number of challenges along the way.

One challenge, and a key reason some video publishers avoid using VAST, is a lack of quality control. Along with the IAB Video Player-Ad Interface Definition (VPAID), VAST can deliver ads programmatically or include ads with complex interactions. If a player isn't programmed to accept VPAID ads, the ad cannot be executed. Even when the player does accept VPAID ads, performance may be slow and cause latency in load times. In the meantime, the audience experiences a delay or a malfunction in their viewing experience.

Publishers and ad vendors need a way to separate the video file from its interactive components to ensure that ads play in systems that cannot execute the interactive components. These ads should also execute more efficiently in players that are equipped to handle the interactions.

Another challenge, especially for broadcasters who are moving their content online, is the lack of a consistent identifier for creative that is maintained across systems. VAST offers a creative identifier, but it has been used inconsistently and one creative may use different identifiers for every system it passes through. A system-wide identifier is a requirement for broadcasters trying to maintain control over the ads they play.

VAST 4 has addressed these challenges along with a few others. As players begin to adopt the updates in VAST 4.x, digital video and audio can expect to see smoother operation and the continued growth that results.

VAST 4.0 Updates

The updates made in VAST 4.0 and the challenges they address are summarized here:

- **Separate Video File and Interactive File:** The complexity of digital video has given rise to the need to separate the linear video file from any creative interactive API files. While the VAST media file has accepted a variety of media files in the past, interactive APIs cannot always be executed. A VAST tag that provides the video file separate from APIs can display more successfully across platforms and devices. The Interactive File standard is expected to be Secure Interactive Media Interface Definition (SIMID) - which is a replacement for VPAID focused on interactivity.
- **Server-Side Ad Insertion Support:** While client-side ad execution and tracking has been the recommended way to track ad impressions and other metrics, digital in-stream video and audio ads are often served to devices (clients) that cannot execute

and track ads using traditional display methods. VAST 4 supports the increasingly common “ad-stitching” method for stitching linear ads into a video or audio content stream and sending it to players with limited capabilities.

- **Mezzanine File:** To support advertising across video platforms that include long-form content and high-resolution screens, VAST 4 features include support for the raw, high-quality mezzanine file. The mezzanine file is very large and cannot be used for ad display, but ad-stitching services and other ad vendor use it to generate files at appropriate quality levels for the environment in which they play.
- **Ready-to-Serve Files:** Along with support for including the mezzanine file, VAST 4 provides guidance on providing three ready-to-serve media files, each at different quality levels, to ensure that a linear video/audio ad can always play. The IAB Digital Video Ad Format Guidelines offers guidance on video/audio file specifications for linear ads.
- **Universal Ad ID:** While VAST has offered a creative identifier in the past, it has been used inconsistently. The new Universal Ad ID feature is used specifically for including a creative identifier that is maintained across systems. The existing `adId` attribute for creative can still be used to log creative IDs specific to the server.
- **Ad Verification and Viewability Execution:** Verification vendors have been using VPAID for measurement verification instead of using it for ad interaction as VPAID was intended. VAST 4 offers a designated space (`<AdVerifications>`) for inserting ad verification APIs, enabling a more streamlined process for executing files strictly intended for ad verification. Open Measurement (OM) is expected to be used for this purpose. In addition, a secondary impression element, the `<ViewableImpression>` element, has been added to allow publishers the option to track viewability on their inventory.
- **Support for Categories:** Ad categories help publishers separate competing ad creative and improve brand safety. VAST 4 ad categories support these efforts.
- **Conditional Ad Declaration:** In programmatic environments, a VPAID unit is sometimes used to decide whether or not to place an ad. If this “conditional ad” never results in an ad to display, the publisher may have to forfeit any revenue from the resulting lost inventory. A declaration in VAST for a conditional ad helps publishers prevent and reclaim any potentially lost inventory revenue in programmatic ad delivery. Note - VPAID & this element are being deprecated as of VAST 4.1 since VPAID is being replaced by SIMID (Secure Interactive Media Interface Definition) and OMID (Open Measurement Interface Definition.
- **New Error Codes:** Along with support for the mezzanine file and other new features, added error codes provide additional troubleshooting support.
- **Standardized Timestamp:** Trackers used in VAST often include timestamp macros, but its use has not been consistent. In VAST 4, the [TIMESTAMP] macro and the format for time has been standardized to enable more consistent time-sensitive tracking.

VAST 4.1 Updates

The updates made in VAST 4.1 are summarized here:

- **Verification:** Changes that enable verification to be supported in a non-VPAID architecture (separated from media file). Also includes changes required to work with Open Measurement.
- **Digital Audio Ad Serving Template (DAAST):** With VAST 4.1, DAAST has been merged into VAST. This mostly involves providing direction in places where audio ads might need to be treated differently. The main change is an optional "adType" added to the "Ad" element to support the various audio use cases.
- **Ad Requests:** VAST is a response protocol. 4.1 now includes a basic Ad Request specification, based on macros.
- **Updates to Macros:** With the new Ad Requests proposal, the Macros section has been completely revamped and updated.
- **Server Side Ad Insertion (SSAI) related changes:** VAST 4.1 includes minor changes to how headers should be handled. The "Ad Request" section is also relevant to SSAI use cases.
- **Deprecating Video Player Ad-Serving Interface Definition (VPAID):** While VPAID will likely be in use for some more time, with VAST 4.1 we are taking the first steps to officially deprecate the use of VPAID. The apiFramework attribute on MediaFile, and the conditionalAd attribute on the Ad element are being deprecated.
- **Updates to Tracking Events:** Added "loaded", "closeLinear" (back from VAST 3.0). Removed acceptInvitationLinear and timeSpentViewing
- **AdServingId:** A required field has been added to simplify comparing data about a video impression across the various systems involved with the delivery and tracking of the impression.
- **VAST Interactive Templates:** Recognizing the need for standardizing interactive ads without ad delivered executable code, VAST 4.1 introduces the concept of interactive templates, with End-Cards as an example template.
- **Closed Captioning:** VAST 4.1 enables Closed Captioning by standardizing the delivery of Closed Captioning files.
- **Flash:** Following up on the white paper to transition video ads from flash to HTML5 (https://iabtechlab.com/html5videotransition/) with VAST 4.1, all references to Flash and Flash resources are being removed.
- **Survey:** The survey node is being deprecated as of VAST 4.1.
- **Other Updates:**
    - MediaFile fixes - fixes to UniversalAdId, added support for more types, changed bounding of Mezzanine files, added support for fileSize etc.
    - Added id attribute to "Advertiser" element
    - Added "Expires" element
    - Added variableDuration to InteractiveCreativeFile

Note - based on feedback received during public comment, the group decided not to deprecate nonlinear ads in VAST 4.1. However, we will continue to explore this format further to determine if the use cases are better handled with other options.

VAST 4.2 updates

The updates included in VAST 4.2 are summarized here:

- Support for Secure Interactive Media Interface Definition (SIMID). SIMID is the replacement for VPAID to support interactive use cases. More information available at this blog post by the IAB Tech Lab. This includes a new tracking event and an error code 902.
- Allow ClickThrough element to be specified in a wrapper to better enable the use case defined in 2.3.5 (managing assets with an ad cloud)
- Added error code 206 to support use cases where the player might decide not to play an ad (for example during live broadcasts where the break needed to be shortened at the last moment).
- Allow multiple UniversalAdID nodes to be provided
- Updates to Icon and Icon ClickFallbackImages
- New macros for better adbreak info.

VAST 4.3 updates

The updates included in VAST 4.3 are summarized here:

- Macros moved to Github for management independent of VAST updates (Section 6.2)
- Macro value added for [PLAYBACKMETHODS] to indicate continuous play, where content episodes are being played back to back without any user interaction. A value of 7 indicates “continuous play”
- For interactive creative files, an inline data URI may be included such as when you want to place HTML directly instead of a URL for retrieving the file. (section 3.9.3)

Intended Audience

This document was designed for digital video and audio technologists who either develop players that accept digital in-stream ads or for vendors who develop ads to be sent to digital in-stream players.

For engineers, section 3 defines all the VAST XML elements. Section 5 includes a “human-readable” schema for quick reference with links to more details in the document if needed. Section 2 defines VAST compliance and section 4 provides technical tips for migrating to VAST 4.x.

For executives, the executive summary and section 1 provide high-level explanation of how VAST can be used to streamline digital video or audio ad operations.

Resources for Digital In-Stream Video and Audio

In order to improve the interconnectivity of the digital video and audio marketplaces, the IAB has published technical specifications, metric definitions, and best practices developed by members with industry experience. Descriptions for each of these publications are listed below.

- **VAST:** The Video Ad Serving Template is an XML response framework that enables a consistent delivery format for ad across streaming video and audio platforms.
- **VPAID:** The Video Player-Ad Interface Definition specifies the protocol between the ad and the media player required to enable ad interactivity and other advanced video advertising functionality. Deprecated - VPAID is being phased out, to be replaced by OMID (for Verification) and SIMID for interactivity.(http://bit.ly/videoAdVision)

- **SIMID:** The Secure Interactive Media Interface Definition, the API made available to build an interactive experience with the video ad media. A replacement for VPAID for interactivity in the VAST 4 model of separated media & executable files.
- **OMID:** The Open Measurement Interface Definition, the API made available to verification code by OMSDK or equivalent service. A replacement for VPAID for verification in the VAST 4 model of separated media & executable files.
- **VMAP:** The Video Multi-Ad Playlist is an XML response framework that defines where to place ads within the video content.
- **Digital Video Ad Metric Definitions:** An industry-defined list of metrics used in digital video ads.
- **Digital Video Ad Format Guidelines:** An industry-defined list of streaming video creative submission specifications. https://iabtechlab.com/standards/iab-digital-video-in-stream-ad-format-guidelines/
- **Digital Advertising Alliance (DAA) Interest-Based Advertising (IBA) Notice for Digital Video:** Guidelines for implementing the AdChoices program within in-stream ads that are placed using interest-based criteria.
- **Open Measurement SDK (OMSDK):** An IAB-led project developing a common library to collect and expose measurements of ad creatives, including video, at view time, for verification purposes.

1 General Overview

VAST is used to send in-stream ad details to a media player. Historically, the player (client) has received, executed, and tracked streaming video or audio ads. However, with the increase in player devices, the player is often unable to execute anything more than a single stream of content. Players might have compensated for this by using one player for content and loading a secondary player for ad playback. After ad playback, the original player would be reloaded for resuming content playback. This process caused a brief buffering period between player loads.

The solution that has emerged for this challenge is a service that involves inserting ads into a stream of content for the player. The result is a seamless experience for the viewer along with the ability to select ads dynamically for insertion and more sophisticated tracking options.

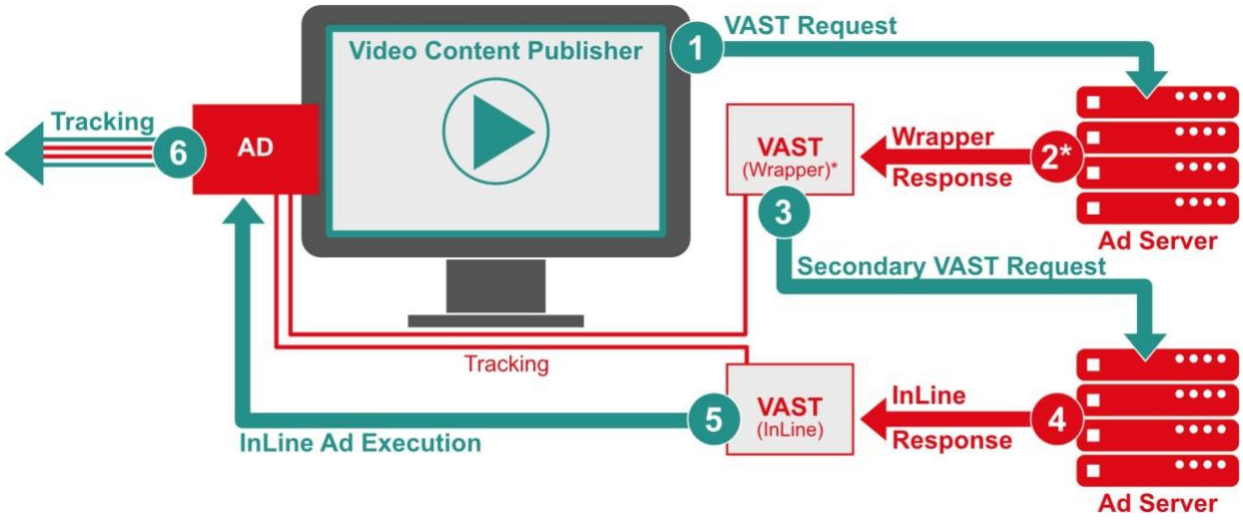
VAST 4.x includes support for high-quality video formats necessary for long-form video content and server-side tracking for use when ad-stitching is leveraged to reach devices that cannot use client-side tracking methods. Version 4.x also allows embedding optional scripts for viewability and ad verification.

1.1 VAST Ad Serving and Tracking

Display advertising uses standardized browser technology to request and execute ads. However, digital in-stream video and audio advertising operates on players, sometimes built with proprietary code. As a template for ads served to a media player, VAST offers a set of instructions for developers on how to program their players to process VAST-formatted ads. Using VAST, ad servers can serve ads to any VAST-compliant player regardless of what code the player uses.

1.1.1 Client-Side Ad Serving

VAST is a unidirectional means of sending ad details to a media player. Built as a layer on top of browser technology, the VAST process that uses client-side execution looks something like this:

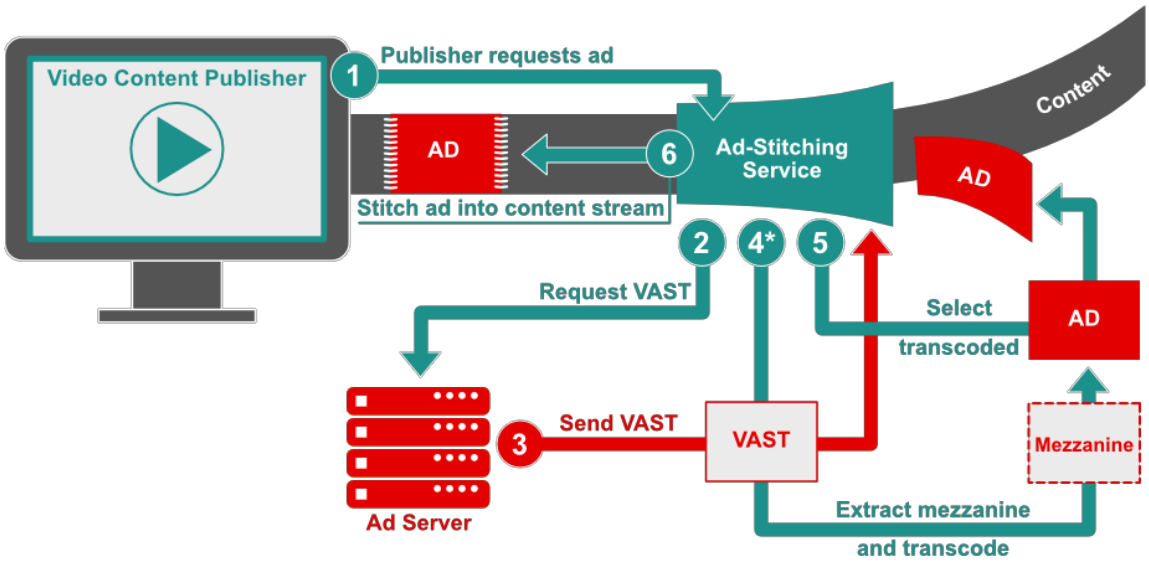


1. **VAST Request:** At some point during content playback, either before (pre-roll), in the middle of (mid-roll), or after (post-roll), the player reaches a cue to insert an ad and uses HTTP to send the request for an ad. See section 1.1.1 on sending an ad request. The request is sent to the primary ad server, which may be the publisher's ad server or a supply-side platform (SSP).
2. *** Wrapper Response:** The primary server responds with VAST. This response is either an InLine response or a Wrapper response. If the server can fill the ad request, it sends an InLine response (step 4). In many cases, the ad server redirects the player to a secondary server using a Wrapper response.
3. **Secondary VAST Request:** If a Wrapper response is received, the player makes a secondary request to another server. The secondary response may be an InLine response or another VAST Wrapper.
4. **InLine Response:** Eventually, after a series of requests and responses, an ad server provides an InLine response.
5. **InLine Execution:** The media player executes the VAST response.
6. **Tracking**: At key points during ad playback, tracking information is sent for both the InLine and Wrapper responses that the player received. In traditional client-side ad serving, cookies are used to track ads and the computers on which they play.

1.1.2 Server-Side Ad Stitching

The example just described the general process for serving an ad directly to a media player, the client, and uses client-side tracking. With client-side tracking, the player sends tracking information. However, in today's wide array of streaming media players the player may not be capable of executing dynamic ad responses or tracking impressions and interactions. In these cases, an intermediary server is needed to insert ads dynamically into the video or audio stream.

Called ad stitching (or stream stitching, ad insertion, etc.), the process looks something like this:



1. **VAST Request:** The publisher sends an ad request to the ad-stitching service.
2. **Request VAST:** The ad-stitching service makes a request to the ad server for a VAST tag.
3. **Send VAST:** The ad server sends a VAST tag with a mezzanine file and ready-to-serve files. If the ad stitching service has already received the creative for a previous request and has transcoded the mezzanine file, or the ready-to-serve files are already in the format required to be stitched into the content stream, then it moves on to step 5. If the VAST tag response is a Wrapper tag then the ad-stitching service should extract the inner InLine response using the same precedence logic as a client-side media player.
4. ***Extract Mezzanine and Transcode:** The ad-stitching service pulls the unique creative identifier from the VAST tag. If the creative has never been used in the system, the mezzanine file is extracted and transcoded. In this scenario, the ad is skipped and the next available ad is played instead. VAST error code 407 is sent.
5. **Select Transcoded:** If the creative in the VAST tag from step 3 matches the unique creative identifier for an ad that has already been transcoded, the ad-stitching service selects the pre-transcoded file already in the system.
6. **Stitch Ad into Content Stream:** The ad-stitching service stitches the ad into the content stream and serves the content and ad to the player in one continuous stream.

Ad-stitching vendors rely on a unique creative identifier for managing the mezzanine source file and its cache of transcoded files for stitching into a video or audio stream. If the ad creative is changed in any way, it should be served with a new creative identifier. In VAST 4.x, the unique creative identifier is provided in the `<UniversalAdId>` element under `<Creative>`. See section 3.7.1 for details.

1.1.3 Headers in Server-to-Server Ad Requests and Ad Tracking

With client-side ad tracking described in section 1.1.2, the player (client) sends tracking included in the VAST tag and uses cookies to determine which computers executed the ad. However, in server-to-server and server-side ad-stitching, the player may not be able to process ad tracking, and the ad-stitching service cannot access cookies used in traditional client-side tracking. Instead, the ad-stitching service must identify devices where ads play by a combination of other methods.

When an ad-stitching service is involved, the ad-stitching server may send tracking on the player's behalf. This server-to-server tracking process is problematic because all the tracking is coming from one IP address. To an ad server that is receiving tracking information, the reports look similar to invalid traffic. In addition, the server is initiating the request on behalf of the client, so it is important to separate information describing the server itself from information describing the client.

Note - For server-side VAST requests prior to version 4.1, namely VAST 2.0, 3.0 and 4.0, the immediate goal is to standardize the ad requests and impression calls from the publisher server (including SSAI platforms) which make the ad request to SSPs, ad exchanges and DSPs. To this end, the publisher server or the SSAI platform may use existing HTTP GET tag requests, on condition that they send the following additional HTTP headers. The goal here is to provide SSAI technology providers the runway to make the transition to the recommended macros-based request model (Section 1.5) which has comprehensive support for this use case, while still allowing for adjustments for the immediate term.

In the future, ad requests will move to a POST based model, which has performance and scaling implications, so the working group recommends that platforms start working on understanding architectural changes required to support POST messages at scale.

To avoid being mistaken as fraudulent traffic, ad stitching providers must include with their ad tracking requests the following HTTP headers:

- `X-Device-IP` set to the IP address of the device on whose behalf this request is being made.
- `X-Device-User-Agent` set to the `User-Agent` of the client on whose behalf the request is being made.

The requester should also include the following headers, if available:

- `X-Device-Referer` set to the `Referer` header value that the client would have used when making a request itself.
- `X-Device-Accept-Language` set to the `Accept-Language` header value that the client would have used when making a request itself.
- X-Device-Requested-With set to the X-Requested-With header value that the client would have used when making a request itself.

The requester may also include the following headers:

- `X-Forwarded-For` for backwards compatibility, although this is now deprecated.
- `X-Device-*` Other HTTP headers that the client would have sent itself may be forwarded as well by the requester using the `X-Device-` prefix.

The information included in these headers must match the information in the original ad request payload, per section 1.1.1.

While these recommendations for tracking support server-side tracking, IAB Impression Measurement Guidelines developed with the Media Rating Council (MRC) favor client-side tracking. "The Measurement Guidelines require ad counting to use a client-initiated approach; server-initiated ad counting methods (the configuration in which impressions are counted at the same time the underlying page content is served) are not acceptable for counting ad impressions because they are the furthest away from the user actually seeing the ad. Measurement counting may happen at the server side as long as it is initiated based

on client-side events and measurement assets. However, pass-through methods (where client-initiated measurement is passed to server-side collection) of signaling interactions detected on the client side from server infrastructure are acceptable." Source: MMTF Public Comment Draft v2 Oct 2017.

In general, an ad-stitching service may have little or no control over ad play once the ad is stitched into the content and streamed to the client. Impression reporting may vary by implementation. For the ad stitching service in situations where the client cannot count impressions, an impression could be reported as the ad is sent on the stitched stream and therefore be as close as possible to the opportunity to play. Alternately, a session-oriented ad-stitching service may report impressions from a given session at session completion. However, any impression measurement beyond the ad-stitched stream is out of the ad-stitching services' control and should be counted by the player whenever possible.

Auditing for compliance with IAB Viewable Ad Impression Measurement Guidelines should focus on disclosing the process by which impressions are counted and any limitations with reporting impressions in certain situations and environments.

See section 2.3.3 for more information on tracking.

1.2 Ad Verification

VPAID, the Video Player-Ad Interface Definition, was originally designed to support interactive ads that controlled the entirety of creative execution. As this was, at the time, the only way to execute code at impression time, ad verification services adopted VPAID in order to run code that verifies and measures playback (including viewability).

An unfortunate side effect of this approach is that, rather than simply enabling monitoring of player-controlled video playback, responsibility for creative rendering is placed on the verification service. In many cases, multiple data-collection VPAID "wrappers" may be used, leading to a fragile chain of intermediaries in the critical path which can significantly delay page rendering and create a negative experience for the viewer.

1.2.1 Loading Verification Resources

In order to support verification with minimal impact to performance, VAST 4 separates media resources from those intended for measurement.

The `<AdVerifications>` element provides a place for verification vendors to specify their executable resources and related metadata, as described in section 3.16. The IAB Tech Lab strongly recommends using code that supports the Open Measurement Interface Definition (OMID) for this purpose, and strongly against using VPAID (which is being retired). OMID has been designed from the ground up to support the needs of verification efficiently, while providing a level of flexibility and transparency that was unavailable in previous VAST generations.

See section 3.16 for details.

1.3 Long-Form Video Support

Long-form video is the extension of traditional broadcast networks to digital mediums. To enable video advertising across screens that include TV content, the response framework in VAST needs to reduce the challenges faced in this digital video environment. Specifically, VAST needs to support the following features:

- Server-to-server ad stitching
- Availability of the high-quality source (mezzanine) file for the ad
- A unique identifier that follows creative and related data from system to system

1.3.1 High-Quality Video

Previous versions of VAST have allowed for multiple media files so that the media player can poll for the file best suited to the environment where the ad will play. However, the high standards for quality in long-form video need more than a few options.

VAST 4.x includes a media container for the mezzanine file, which is the raw high-quality video file that the publisher can use to produce the best quality file where needed. In addition to the mezzanine file, VAST 4.x requires either an adaptive stream ready-to-serve file or a minimum of three media files at different levels of quality: high, medium, and low. Identifying the quality levels of three media files enables the media player to more quickly find the appropriate file needed for a given environment. A separate document, the IAB Digital Video Ad Format Guidelines, is provided for ad developers and outlines encoding recommendations for each of these files.

See section 3.9 for details.

In order to support additional formats such as 3D, augmented reality (AR), virtual reality (VR), 360-degree video, etc., more than one mezzanine file may be included, with a “type” attribute to help identify the type of mezzanine file.

1.3.2 Unique Creative Identification

As ad creatives move from system to system, they become more difficult to track and impressions involving these creatives become difficult to reconcile in reports from different systems. Historically, VAST has provided a placeholder for a creative id, but its purpose has been unclear and its use varies under different vendors and use cases.

In VAST 4.x, the placeholder for a unique creative ID has been pulled out into a new element to draw more attention to it and provide attributes that more clearly define the id. The new `<UniversalAdId>` element is required for linear ads and consists of an attribute for defining the `idRegistry` and the value of the ID specified in the content of the node.

Using a unique creative identifier enables all data associated with the creative to follow across systems. Unifying data under a unique ID streamlines data collection operations, reporting, and analysis.

See section 3.7.1 for details.

1.4 Audio Ad Support

DAAST, the Digital Audio Ad Serving Template was developed in 2014 as a spin-off of VAST 3.0 to support Audio ads. DAAST is now being merged back into VAST4, though the name of the standard will continue to be VAST for the moment.

The main changes are on the VAST “Ad” element (to include an “adType" attribute), some minor additions (identified below), using the term “media player” instead of “video player”, and instructions on handling various VAST elements/attributes when in an Audio Ad scenario.

1.4.1 Audio Player Use Cases:

Audio players may also support video and display ads but have the additional complexity of being able to run in the background. This means that there are various modes in which ads might be used and need to be well understood.

1. Audio only ads
2. Audio ads with Companion Ads
3. Audio ad provided through a video creative, expected to play as audio only mode
4. Video ad trafficked as audio first, but with the potential to be seen for additional impact.

1.4.2 “Audibility” / Viewability:

Discussions are still ongoing about the concept of “Audible Impression”, and will be added later.

1.5 VAST Ad Requests

VAST is a response protocol. Historically, the request mechanism was not discussed in the VAST specs, even though the request is important since it contains the information needed by the ad server to respond with a VAST response. The protocols used for ad requests vary based on the type of inventory being served.

- For programmatic ad slots, OpenRTB is the standard typically used for ad requests, and the OpenRTB response could include a VAST response.
- For non-programmatic scenarios, the ad requests are currently not based on any standard and are proprietary agreements between the publishers and ad servers. Typically this would consist of an HTTP GET request, with additional data passed in the URL in the path or in query parameters in a key=value format. This is generally a mix of platform-specific parameters, as well as information that is commonly required

across the industry, such as device IDs, contextual information such as domain or app ID, or details of the position of the video in video content.

In spite of many ad servers sharing requirements for this common data, passing this information is extremely difficult, as ad servers often accept these values in different formats. Additionally they are generally passed along VAST Wrapper chains using ad server macros which populate at the time a VAST Wrapper document is generated. This means if data must be passed from ad server A to ad server B to ad server C along a wrapper chain, traffickers in both A and B must traffic proprietary tags properly and have values align, otherwise data will be dropped.

Often necessary common data is missing by the time a final ad server is reached, causing either loss of the opportunity or selection of a suboptimal ad. To mitigate this, some standardization of request values, settable by the VAST client, is needed.

With VAST 4.1, a first step will be made towards a request protocol in VAST. To ensure ease of adoption given the current industry setting of proprietary ad tag parameters on HTTP GET requests, the request protocol will focus entirely on ensuring an agreed-upon set of values that VAST clients can set, and which all ad servers should accept. Future versions of VAST will go further, to define a full request protocol based on AdCOM (reference OpenRTB 3.0) which will define request structure as well as values.

As a further note, this standard will also hold when requests are made for VMAP, as requests for VMAP often result in VAST documents populated in the VMAP response. As such, these values are equally needed at VMAP request time.

To allow VAST clients to set values in ad tags, the VAST macro concept will be expanded to apply not just to tracking URIs, but to AdTagURIs in Wrapper creatives, as well as to initial ad tags URIs passed to a VAST client for the ad initial request.

The list of macros (for both VAST Ad Requests as well as for tracking) has been consolidated in Section 6.

1.6 VAST Interactive Templates

While interactive video ads command a premium, they are not supported on all platforms or by all publishers. While this is partly due to concerns around VPAID, which are being addressed by VAST4 and the replacement for VPAID, the execution of unknown code may never be allowed in many cases. To address this, VAST 4.1 introduces the concept of “VAST Interactive Templates”. These are interactive experiences that only require some visual assets (images, css, etc.) and some instructions/metadata in the VAST tag. The publisher implements the interactive code and uses the metadata to run the interactive ad.

In VAST 4.1 we have included an “end-card” (Section 3.13) based on a use case that has already been informally implemented in the industry. We expect to add more such templates in the future.

Note - The IAB Tech Lab recommends the use of VAST extensions for the industry to experiment with such experiences, and then bring in proposals to the Digital Video Technical Working Group to formalize them as standard templates.

1.7 Flash Support

As indicated in the Flash to HTML5 transition guidance released by the IAB Tech Lab in December 2016, all Flash references are considered deprecated as of January 2017. VAST4 is now officially being updated to reflect that position by removing all Flash related references

1.8 Handling MediaFile Nodes During the Transition from VPAID

One of the goals of VAST4 is to eliminate the practice of using the MediaFile node to deliver executable code (usually VPAID). To achieve this goal, the AdVerifications Node and the InteractiveCreativeFile were added in VAST 4, so that executable code for measurement and for interactivity could be delivered separately from the MediaFile. VPAID is being replaced by OMID for verification/measurement and SIMID for interactivity. (Refer to http://bit.ly/videoAdVision for more information).

However, adoption of new protocols require time and so there will be a transition period where VPAID remains in use. With VAST 4.1, the working group has also deprecated the “apiFramework” attribute on the MediaFile node, which enables the delivery of VPAID. During the transition period, VAST 4.2 tags will likely have both the AdVerifications node for Open Measurement, alongside a VPAID MediaFile element.

This section provides recommendations for both publishers and tag creators to help during this transition period-

For VAST tag creators (both direct as well as wrapped tags):

VAST tag creators should plan to deliver the relevant VAST content based on information in the ad requests (OpenRTB and via VAST ad request macros - refer section 1.5 and 5). Use the `APIFRAMEWORKS` macro.

1. When OMID support is indicated in the request, include the AdVerifications node (or extensions node with type AdVerifications for pre VAST 4.1) to include verification scripts.
2. If VPAID support is indicated in the request and the tag creator requires VPAID support (for interactivity or ad blocking), include the following in the VAST tag
    a. The AdVerifications node (or extensions node with type AdVerifications for pre VAST 4.1)
    b. The VPAID file type MediaFile node (for Blocking or Interactivity)
3. If OMID/VPAID support status is not known and there is no hard requirement on VPAID use from the buyer, ensure that the VAST tag contains all 3 of the following so that the publisher has all resources available:
    a. The AdVerifications node (or extensions node with type AdVerifications for pre VAST 4.1)
    b. The VPAID file type MediaFile node (for Blocking or Interactivity)
    c. One or more non-VPAID (video creative) MediaFile nodes.

For Publishers

For best results, send information in the ad request (OpenRTB and VAST ad request macros) about capabilities of the video player. Use the `APIFRAMEWORKS` macro.

Based on the standards supported, execute in this order for best results

1. If OMID is supported, run the AdVerifications node (for Open Measurement) - and the video creative MediaFile
2. If OMID is not supported but VPAID is supported and VPAID MediaFile node is available in VAST tag, run VPAID
3. If OMID and VPAID are supported, run both AdVerifications node and VPAID
4. If OMID and VPAID not supported, run one of the non-VPAID MediaFile nodes.

The above is only a recommendation and might not work in all cases. Publishers can decide to run any MediaFile they deem appropriate for their use cases, so as always, please ensure that your integrations are working as expected and per your business agreements.

Also, publishers and technology vendors are responsible for testing the various combinations above and ensure that potential issues like double counting or namespace clashes are correctly handled.

Once Open Measurement supports Brand Safety and once the interactivity replacement for VPAID is defined (and is delivered via the InteractiveCreative Node) the use of “VPAID-MediaFile” will be eliminated.

2 VAST Compliance

Compliance is a two-party effort that involves, at a minimum, the media player and the ad server. Both must meet certain expectations so that VAST can be truly interoperable and encourage growth in the marketplace.

General Implementation Note	Open Measurement, VPAID and VMAP specs are excluded from VAST compliance because these specs are independent of each other and of VAST. Compliance with one spec does not imply compliance with any of the other specs. Compliance for either spec must be separately declared.

2.1 Ad Server Expectations

VAST-compliant ad servers must be able to serve ad responses that conform to the VAST XML schema defined in this document. Ad servers must also be able to receive the subsequent tracking and error requests that result from the media player's execution of the VAST ad response. Tables for each VAST XML element define which are required in a VAST response.

2.2 Media Player Expectations

VAST-compliant media players and SSAI systems must be able to play the ad in a VAST response according to the instructions provided by the VAST ad response and according the media player's declared format support, which includes:

- Rendering the ad asset(s) correctly
- Respecting ad server instructions in a VAST response including those of any subsequent ad servers called in a chain of VAST Wrapper responses, providing the responses are VAST-compliant
- Responding to supported user-interactions
- Sending appropriate tracking information back to the ad server
- Supporting XML conventions such as standard comment syntax (i.e. acknowledge VAST comments in the standard XML syntax: `<!--comment-->`)

Details for proper ad display and VAST support are defined throughout this document, including player support requirement notes for each XML element.

2.3 General Compliance

VAST specifies both the format of the ad response and how the media player should handle the response. In order for VAST to be effective, both ad servers and media players must adopt the guidelines outlined in this document.

In general, the video player need only accept ads that it requests and ad server responses should be displayed in the ad format intended.

For example, VAST allows for compliance while only supporting a subset of ad types (described in section 2.3.1). For example, if a standard Linear Ad is requested but a Skippable Linear Ad is received, the media player is not expected to display the Skippable Linear Ad nor should the media player play the Skippable Ad as a Linear Ad (without skip controls).

The following features must be supported for general functionality:

- Declaration of ad types
- XML structure
- Tracking
- Wrappers
- Error reporting
- Macros
- Industry icons
- Verification

These features are described in the following sections.

2.3.1 VAST Ad Types

VAST covers several distinct ad types, but ad serving and publisher organizations may not want to support all formats. For example, some vendors may choose to serve only linear ads with companions.

VAST 3.0 introduced five ad types for compliance so that organizations may be compliant with VAST while only supporting a selected subset of the ad types.

The VAST- compliant ad types are as follows:

1. Linear Ads
2. NonLinear Ads
3. Companion Ads
4. Skippable Linear Ads
5. Ad Pods

A company wishing to display IAB's seal for VAST compliance must declare which of the five ad types their technology supports.

Note – all ad types are relevant for "audio only" ads except NonLinear ads.

2.3.2 XML Structure

A VAST-compliant ad response is a well-formed XML document, compliant with XML 1.0 so that standard XML requirements such as character entities and `<!--XML comments-->` should be honored. It must also pass a schema check against the VAST 4.x XML Schema Definition (XSD) that is distributed in conjunction with this document.

IAB Tech Lab Github URL for the XSD: https://github.com/InteractiveAdvertisingBureau/vast

Ad Server Implementation Note	All URIs or any other free text fields containing potentially dangerous characters contained in the VAST document should be wrapped in CDATA blocks. The VAST response should be carefully tested for appropriate treatment of URI characters that require special handling.

2.3.3 Encoding URIs for VAST

URIs provided in a VAST response must be CDATA-wrapped as in the following example:

```
<Impression id="myserver">
      <![CDATA[
      http://ad.server.com/impression/dot.gif
      ]]>
</Impression>
```

Wrapping the URI in a CDATA section enables most characters to be included as they are. For example, without a CDATA section, the character `&` would need to be encoded as `&`. However, encoding this within a CDATA section double-encodes the URI.

Consider the CDATA wrapping needed for the following URI:

```
http://ad.server.com/impression/dot.gif?v=1&id=abc
```

```
<Impression id="myserver">
      <![CDATA[
      http://ad.server.com/impression/dot.gif?v=1&amp;id=abc
      ]]>
</Impression>
```

The encoding of `&` into `&` is not necessary in this example because the URI is enclosed in a CDATA section. Characters in the URI that are also used to section out the URI with CDATA may need extra encoding.

Consider the CDATA wrapping needed for the following URI:

```
http://ad.server.com/impression/dot.gif?s=x]]>x
```

```
<Impression id="myserver">
      <![CDATA[http://ad.server.com/impression/dot.gif?s=x]]
      ]]>
      <![CDATA[>]]>
      x
</Impression>
```

The `]]>` characters are used to close the CDATA section; therefore, the `>` character must be enclosed in a secondary CDATA section. Since the `x` is harmless character at the end, it can be left outside the CDATA section and will be concatenated with the other two URI components, each closed in their CDATA sections.

Since CDATA-wrapping URIs is a requirement in VAST, the author of the VAST response should carefully edit and test all included URIs, especially when input values require special handling. Incorrect treatment of these characters may cause ad playback to fail or enable content injection attacks.

Some additional examples are offered in the following table.

Impression URL: `http://ad.server.com/impression/dot.gif`

```
<Impression id="myserver">
      http://ad.server.com/impression/dot.gif
</Impression>
```

Not enclosed in a CDATA section

```
<Impression id="myserver">
      <![CDATA[http://ad.server.com/impression/dot.gif]]>
</Impression>
```

Correct and backwards compatible

Impression URL: `http://ad.server.com/impression/dot.gif?v=1&id=abc`

```
<Impression id="myserver">
      http://ad.server.com/impression/dot.gif?v=1&amp;id=abc
      </Impression>
```

The `&` is xml-encoded, but the URI needs to be wrapped in a CDATA block

Invalid -

```
<Impression id="myserver">
      http://ad.server.com/impression/dot.gif?v=1&id=abc
</Impression>
```

Not only is this URI not CDATA enclosed but the `&` is also not XML-encoded

```
<Impression id="myserver">
      <![CDATA[http://ad.server.com/impression/dot.gif?v=1&amp;id=abc]]>
</Impression>
```

This URI is both CDATA-enclosed and the & is XML-encoded. The player will interpret the URI as:

`http://ad.server.com/impression/dot.gif?v=1&id=1234`

```
<Impression id="myserver">
      <![CDATA[http://ad.server.com/impression/dot.gif?v=1&id=abc]]>
</Impression>
```

This URI is properly wrapped in a CDATA block. The `&` doesn't need to be encoded.

Impression URL: `http://ad.server.com/impression/dot.gif?s=x]]>x`

```
<Impression id="myserver">
      http://ad.server.com/impression/dot.gif?s=x]]&gt;x
</Impression>
```

Not enclosed in a CDATA section even though > is encoded

```
<Impression id="myserver">
      http://ad.server.com/impression/dot.gif?s=x]]>x
</Impression>
```

Not enclosed in a CDATA section

```
<Impression id="myserver">
      <![CDATA[http://ad.server.com/impression/dot.gif?s=x]]>x]]>
</Impression>
```

CDATA section used but appears to end early because of the]]> characters before the x, potentially allowing content injection attacks

```
<Impression id="myserver">
      <![CDATA[http://ad.server.com/impression/dot.gif?s=x]]]]>
      <![CDATA[>]]>x
</Impression>
```

Correct and backwards compatible

Finally, a VAST-compliant ad response must conform to certain additional dependencies that cannot be expressed in the VAST 4 XSD. For example, one ad type of either `<InLine>` or `<Wrapper>` is allowed but not both. Another example is the protocol for providing 3 ready-to-serve media files, ideally separate from any interactive components (see section 3.9). The XSD can only validate for whether you've provided a URI under `<MediaFile>`; it cannot validate whether the appropriate files have been provided. Such dependencies are further described throughout this document.

2.3.4 Tracking

VAST tracking is implemented using a number of individual tracking elements that map to video events, such as video start or video completion. Each of these elements contains a reference to a server-side resource, which historically has been an 1x1 pixel image, but may also be a script or document reference. Calls to these resources are counted by the ad server or other measurement vendor to tally up the total for a specific video event.

Publisher Implementation Note	The publisher is responsible for making the server-side request associated with a specific video event when that event occurs during video playback. These events may originate from a client-side player, or (in some SSAI cases) from the publisher server. In the event the request comes from the publisher server extra care must be taken to make sure that the calls are made concurrently with the corresponding playback events, and any missing client-side information (user agent, etc) should be passed along via headers or other mechanism.

The media player is required to request the resource file for any included tracking elements from the URI provided at the appropriate times, or "fire" the tracking element. Advertisers and publishers depend on accurate tracking records for billing, campaign effectiveness, market analysis, and other important business intelligence and accounting. Good tracking practices throughout the industry are important to the success and growth of digital video and audio advertising.

General Implementation Note	The publisher must send requests to the URIs provided in tracking elements; however, the publisher is not required to do anything with the response that is returned. The response is only to acknowledge an event and to comply with the HTTP protocol. This response is typically a 200 with a 1x1 pixel image in the response body (although the response could be of any other type).

The use of multiple impression URIs enables the ad server to share impression-tracking information with other ad serving systems, such as a vendor or partner ad server employed by the advertiser. When multiple impression elements are included in a VAST response, the

media player is required to request all impressions at the same time or as close as possible to the same time. Any significant delay between impression requests may result in count discrepancies between ad serving systems.

Publisher Implementation Note	If multiple `<Impression>` elements are provided, they must be requested at the same moment in time or as close in time as possible. In particular for a VAST response containing a `<Linear>` element, compliance with the IAB Digital Video Measurement Guidelines. If any of the requests are delayed significantly, discrepancies may result in the counts of participating ad serving system.

2.3.5 VAST Wrappers

Wrappers provide a way for one ad server to redirect a media player to another, secondary ad server to retrieve an ad, multiple ads, or yet another VAST Wrapper.

One ad server may redirect to another for a variety of reasons:

- The first ad server has selected a specific advertiser campaign to fill the inventory. In this case the redirect instructs the secondary ad server to return specific ads from a particular ad campaign.
- The first ad server is delegating a specific piece of inventory for either a single ad or an entire Pod of ads to the secondary ad server to fill with any ads that are within an established agreement between the two parties.
- An ad server may wish to delegate delivery of the specific ad creative file(s) to a separate asset repository/host (ad cloud).
- An ad server may have no ad to return and may return a redirect to a backfill provider.

2.3.5.1 Infinite Loops and Dead Ends

When serving an ad involves a chain of Wrappers, an infinite loop is possible where a chain of Wrappers never results in a final InLine VAST response. Another case involves a finite number of VAST Wrappers in which the resulting InLine response is used as a decisioning mechanism to find an ad instead of delivering the ad as required. In these cases, the decisioning mechanism may never return an ad or may take too long to return the ad.

In general, VAST Wrappers should be limited to five before resulting in an InLine response. If the player detects more than five Wrappers, the player may reject any subsequent responses in the chain, replace the [ERRORCODE] macro in the VAST/Ad/Wrapper/Error URI if provided to indicate that the Wrapper limit was reached, and move on to the next option for an ad. Error codes should be sent for all wrappers in the chain where provided.

When an InLine response fails to produce an ad within the timeframe identified in VPAID or other ad framework, the player may reject the ad, send error code 304 to indicate that no ad was produced in the given timeframe, and move on to the next option for an ad. Error codes should also be sent to any wrappers preceding the InLine response.

2.3.5.2 Wrapper Conflict Management and Precedence

When creative elements, such as Companion creative or click-throughs, are included directly in the Wrapper response, conflict may occur. In a VAST ad, whether served with multiple Wrappers or in one Inline response, all creative offered is intended to be part of the same creative concept, and the media player should attempt to display all creative presented in the response (or in a chain of responses). However, when a conflict occurs, the media player should favor creative elements offered closest to the InLine response.

For example, if a wrapper contains companion creative and the InLine response also contains companion creative, the companion creative in the Inline response should be selected (unless both creative can be displayed without conflict).

In another example, if the InLine response is absent of any companion creative but two or more Wrappers contain companion creative, then creative for the Wrapper served closest to the InLine response should be favored. However, if multiple creative can be served without conflict, the media player should attempt to display whatever creative it can.

With respect to ClickThrough handling, if the InLine response doesn't include a ClickThrough element and one or more of the calling Wrappers includes a ClickThrough element, then the ClickThrough element closest to the InLine response must be favored. Similarly, if the inLine response includes a ClickThrough element then this must be favored over ClickThrough element(s) specified in calling Wrappers.

2.3.6 Error Reporting

The `<Error>` element enables the media player to provide feedback to ad servers when an Ad cannot be served. In VAST 3.0, detailed error codes and specifications for format are provided to enable detailed error logging for better ad serving diagnostics.

Providing more detailed error codes enables stronger diagnostics and enables better technology development over time. If ad servers can collect more detailed information about why their ads or specific creative couldn't be served, they can improve their systems to produce fewer errors.

The `<Error>` element is an optional element nested within the `<InLine>` or `<Wrapper>` element. It is used to track errors for an Ad. An error for an Inline Ad that is part of a chain of Wrappers will produce an error for each of the Wrappers used to serve the Inline Ad.

An `<Error>` element is also provided at the root VAST level and is primarily used to report a "No Ad" response. See section 2.3.6.4) for more information.

2.3.6.1 Ad Server Details: <Error> Element

An `<Error>` element includes a URI that provides a tracking resource for the error. This error-tracking resource is called when the media player is unable to display the Ad.

The following example is a sample VAST response that includes the `<Error>` element for an Inline Ad.

```
<InLine>
      …
      <Error>
            <![CDATA[http://adserver.com/error.gif]]>
```

```
        </Error>
        …
</InLine>
```

If the ad server wants to collect more specific details about the error from the media player (as listed in section2.3.6.3), an [ERRORCODE] macro can be included in the URI.

2.3.6.2 Media Player Details

If an error occurs while trying to load an Ad and the `<Error>` element is provided, the media player must:

- Request the error source file using the URI provided.

Replace the `[ERRORCODE]` macro, if provided, with the appropriate error code listed in the table in section2.3.6.3. At a minimum, error code `900 (Unidentified error)` can be used, but a more specific error code benefits all parties involved.

If the Ad was served after a chain of Wrapper ad responses, the media player must also return error details as listed above for each Wrapper response that also includes error parameters. Macro responses must be correctly percent-encoded per RFC 3986.

The following table lists VAST error codes and their descriptions.

2.3.6.3 VAST Error Codes Table

Code	Description
100	XML parsing error.
101	VAST schema validation error.
102	VAST version of response not supported.
200	Trafficking error. Media player received an Ad type that it was not expecting and/or cannot play.
201	Media player expecting different linearity.
202	Media player expecting different duration.
203	Media player expecting different size.
204	Ad category was required but not provided.
205	Inline Category violates Wrapper BlockedAdCategories (refer 3.19.2).
206	Ad Break shortened. Ad was not served.
300	General Wrapper error.
301	Timeout of VAST URI provided in Wrapper element, or of VAST URI provided in a subsequent Wrapper element. (URI was either unavailable or reached a timeout as defined by the media player.)
302	Wrapper limit reached, as defined by the media player. Too many Wrapper responses have been received with no InLine response.

303	No VAST response after one or more Wrappers.
304	InLine response returned ad unit that failed to result in ad display within defined time limit.
400	General Linear error. Media player is unable to display the Linear Ad.
401	File not found. Unable to find Linear/MediaFile from URI.
402	Timeout of MediaFile URI.
403	Couldn't find MediaFile that is supported by this media player, based on the attributes of the MediaFile element.
405	Problem displaying MediaFile. Media player found a MediaFile with supported type but couldn't display it. MediaFile may include: unsupported codecs, different MIME type than `MediaFile@type`, unsupported delivery method, etc.
406	Mezzanine was required but not provided. Ad not served.
407	Mezzanine is in the process of being downloaded for the first time. Download may take several hours. Ad will not be served until mezzanine is downloaded and transcoded.
408	Conditional ad rejected. (deprecated along with conditionalAd)
409	Interactive unit in the InteractiveCreativeFile node was not executed.
410	Verification unit in the Verification node was not executed.
411	Mezzanine was provided as required, but file did not meet required specification. Ad not served.
500	General NonLinearAds error.
501	Unable to display NonLinearAd because creative dimensions do not align with creative display area (i.e. creative dimension too large).
502	Unable to fetch NonLinearAds/NonLinear resource.
503	Couldn't find NonLinear resource with supported type.
600	General CompanionAds error.
601	Unable to display Companion because creative dimensions do not fit within Companion display area (i.e., no available space).
602	Unable to display required Companion.
603	Unable to fetch CompanionAds/Companion resource.
604	Couldn't find Companion resource with supported type.
900	Undefined Error.
901	General VPAID error.
902	General InteractiveCreativeFile error code

2.3.6.4 No Ad Response

When the ad server does not or cannot return an Ad, the VAST response should contain only the root `<VAST>` element with optional `<Error>` element, as shown below:

```
<VAST version="4.1">
      <Error>
            <![CDATA[http://adserver.com/noad.gif]]>
      </Error>
</VAST>
```

The VAST `<Error>` element is optional but if included, the media player must send a request to the URI provided when the VAST response returns an empty InLine response after a chain of one or more wrappers. If an `[ERRORCODE]` macro is included, the media player should substitute with error code `303`.

Besides the VAST level `<Error>` resource file, no other tracking resource requests are required of the media player in a no-ad response in either the Inline Ad or any Wrappers.

2.3.7 Industry Icon Support

Several initiatives in the advertising industry involve using an icon that overlays on top of an Ad creative to provide some extended functionality such as to communicate with consumers or otherwise fulfill requirements of a specific initiative. Often this icon and its functionality may be provided by a vendor, and is not necessarily served by the ad server or included in the creative itself.

One example of icon use is for compliance to certain Digital Advertising Alliance (DAA) self-regulatory principles for interest-based advertising (IBA). This section provides an overview of how media players can support the use of icons in a general manner while using the DAA's AdChoices program, as a specific example.

Icons are optional for audio, and can be used in the context of a companion banner. But since audio ad players are not required to have a rendering engine, icons are not a requirement when the adType is "audio" or "hybrid".

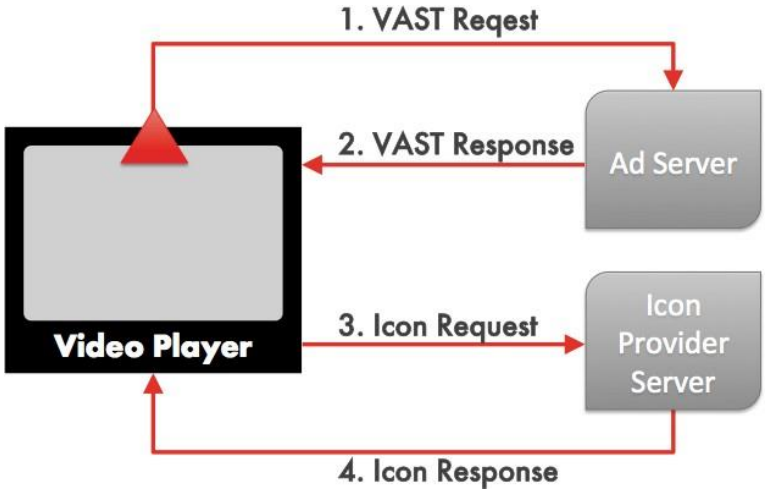
2.3.7.1 Icon Use Case: AdChoices for Interest-Based Advertising (IBA)

The Digital Advertising Alliance (DAA) sets forth principles that endeavor to give consumers a better understanding of and greater control over ads that are customized based on the consumer's online behavior. This control is made available to the consumer in the form of the AdChoices icon, which is displayed in a prominent location in or around the Ad creative. When a consumer clicks the icon, they may be offered: information about the ad server and data providers used to select the Ad, options to learn more about online behavioral advertising (OBA), and the ability for consumers to opt out from receiving OBA ads in the future.

2.3.7.2 The <Icons> Element

VAST 3.0 introduced the `<Icons>` element, which is offered under the `<Linear>` creative element for both Inline and Wrapper ad elements.

The following diagram illustrates the general process for how the `<Icons>` element is represented in a VAST response.



The Icon Provider Server represented in this diagram may be the same server that serves the VAST response but more commonly, is a vendor that serves the icon from its own systems.

When the `<Icons>` element is included in the VAST response, the media player must display the object as an overlay on top of the Linear Ad with which the icon is served and after the ad has started (i.e. first frame of video is displayed in the player).

Media Player Implementation Note	Since a vendor often serves icons and may charge advertising parties for each icon served, the media player should not pre-fetch the icon resource until the resource can be displayed. Pre-fetching the icon resource may cause the icon provider to falsely record an icon view when the icon may not have been displayed.

2.3.7.3 Precedence and Conflict Management:

As an Ad goes through a delivery chain, companies may include their own Icon element in their Wrapper responses. Sometimes these multiple icon elements are all for the same program and the media player must decide on only one icon to display. When icon elements represent more than one program, one icon from each program should be displayed.

The media player can use its own business rules to decide which icon to display, along with any specific program recommendations. For example, when multiple AdChoices icons are offered, the DAA program recommendation is to select the icon that is closest to the creative. To comply with the AdChoices program when multiple AdChoices icons are served, the media player must choose the icon closest to the creative.

If no other rules govern the selection of which icon to display, the media player should choose the one closest to the creative. That is, if the `<Icon>` element is included within the Inline Ad, then that icon is the closest to the creative. However, if the Inline Ad contains no

<Icon> element, but the last Wrapper in a chain of Wrappers did contain the <Icon> element, then the icon from that last Wrapper is the one closest to the creative.

When multiple icons from more than one icon program are included in a chain of wrappers, the media player must decide which icon from each program should be displayed. Again, the media player can use its own business rules; however, the icons must not overlap each other. If all program icons use the same xPosition and yPosition values, the media player can use width and height attribute values to offset coordinates relative to the display area of the Ad creative.

Media Player Implementation Note	A media player may not be able to display an Icon but should make every attempt to do so.

2.4 Viewability Verification and Interactive Linear Creative

VAST 4 adds new sections in the Linear file for viewability, ad verification, and interactive creative files. These new sections offer performance and measurement benefits but also add a level of complexity.

Player compliance with VAST 4 requires appropriate execution of these files.

The player should execute the ad in the following order:

1. Start loading verification resources.
2. Start loading video assets and interactive resources.
3. Initialize interactive resources.
4. Start ad playback.

Player expectations on these added features are summarized here and further defined in their corresponding sections.

2.4.1 Publisher Viewability

Publishers have the option to offer viewable impression tracking on the ad using the <ViewableImpression> feature added in VAST 4. Three URIs may be provided to track whether the ad was <Viewable>, <NotViewable>, or <ViewUndetermined> (see section 3.5).

Note that this feature is specific to (the likely uncommon) situation where the publisher is the party monitoring ad geometry and making the viewability determination. As such, it will very likely be limited to situations where the buyer and seller have some prior relationship and agreement around measurement mechanics and the viewability standard used. It is not a general replacement for, nor should it be confused with, measurement and reporting of viewability by third-party verification services (as in section 2.4.2). These URIs are not intended for reporting viewability determinations from such parties.

This feature is not applicable to audio ads.

2.4.2 Viewability with Ad Verification Services

Ad Verification services can be requested for measurement by adding a `<Verification>` element under `<AdVerifications>` including their executable resources and associated per-impression metadata. Multiple vendors may use this feature to measure the same ad session. All verification resources listed in the VAST should be executed, including those from any intermediary <Wrapper> VASTs, barring exceptions based on a whitelist or other pre-defined rules as outlined below.

A VAST 4 player must check for these elements, however, players may optionally either: refuse to execute unknown resources or declare it as not supported. The recommended process is to consult an IAB TechLab-provided/certified list of known verification vendors and domains, although the exact mechanism is left to the publisher/player. The player must request each associated verificationNotExecuted tracking event URI (with the [REASON] macro filled with reason code 1) in the case that it refuses to execute one or more verification script (see section 3.17.4 for details).

Verification resources should be executed as required by the OMID specification. If the code cannot be executed as provided, any included verificationNotExecuted tracking events URIs must be sent with the appropriate reason code (e.g. not supported, error, etc.).

2.4.3 Interactive Linear Creative Files

In VAST 4, the `<MediaFile>` should only be used to include video or audio files. For Linear files that require an API framework to be executed, the new `<InteractiveCreativeFile>` should be used to include these files. Once the `<AdVerifications>` element has been checked for verification code, the `<InteractiveCreativeFile>` element should be checked for code in order to execute the ad. When the `<InteractiveCreativeFile>` cannot be executed, the `<Error>` should be sent and the player may check the `<MediaFiles>` element for any media files that are available to be played.

This script asset should only be used to enable interactive, dynamic or other creative capabilities and not used for viewability, client-side arbitration, or other non-creative uses.

3 VAST Implementation

VAST is an XML schema for providing metadata about an ad for in-stream video or audio that is parsed by a player or by a server on the player's behalf. This section provides the details for forming the VAST.

Beginning with section 3.1, each element available in VAST is described and a table summarizes information about hierarchy, requirements, and attributes. Each VAST element that includes nested elements is defined under a second-level heading in this document. Third-level headings represent nested elements that have no additional nested elements under them.

Links to the table of contents (TOC) and the schema are provided under each heading to aid in navigation. The human-readable schema in section summarizes VAST elements and provides a link to the chapter that describes the element in more detail.

Before forming a VAST document, considerations for the XML namespace and browser security for JavaScript or other scripting languages should be established as follows.

XML Namespace
Whenever VAST is used in conjunction with any other XML template, such as with VMAP or VAST extensions, a namespace should be declared for each so that the elements of one are not confused with the elements of another.

For more information, visit: http://www.w3.org/TR/REC-xml-names/

Browser Security
Modern browsers restrict Adobe Flash and JavaScript runtime environments from retrieving data from other servers. Since typical VAST responses come from other servers, measures must be taken for each.

Cross Origin Resource Sharing (CORS) for JavaScript
In order for JavaScript media players to accept a VAST response, ad servers must include a CORS header in the http file that wraps the VAST response. The CORS header must be formatted as follows:

```
Access-Control-Allow-Origin: <origin header value>
Access-Control-Allow-Credentials: true
```

These HTTP headers allow an ads player on any origin to read the VAST response from the ad server origin. The value of `Access-Control-Allow-Origin` should be the value of the `Origin` header sent with the ad request.

Setting the `Access-Control-Allow-Credentials` header to `true` will ensure that cookies will be sent and received properly.

***Note**: For requests where the Origin header is null, ad servers should respond with only `Access-Control-Allow-Origin: *` (and no `Access-Control-Allow-Credentials` header) to prevent breaking on originless requests, such as those from iOS wkwebviews.*

For more information, visit http://www.w3.org/TR/cors

3.1 Declaring the VAST Response

All VAST responses share the same general structure. Each VAST response is declared with `<VAST>` as its topmost element along with the `version` attribute indicating the official version with which the response is compliant. For example, a VAST 4.1 response is declared as follows:

```
<VAST version="4.1">
```

As with all XML documents, each element must be closed after details nested within the element are provided. The following example is a VAST response with one nested `<Ad>` element.

```
<VAST version="4.1">
      <Ad>
            <!--ad details go here-->
      </Ad>
```

```
</VAST>
```

3.2 VAST

TOC Schema

VAST is the root node for a VAST-compliant ad response and is used to declare the VAST response as described in section 3.1.

Player Support	Required
Required in Response	Yes
Parent	None (root)
Bounded	1
Sub-elements	Error* Ad*
Attributes	Description
version	A float (number with decimal) to indicate the VAST version being used.

*either `<Error>` or `<Ad>` may be provided but not both

3.2.1 Error (VAST)

TOC Schema

Used to report a no-ad response. When the ad server does not or cannot return an Ad, the VAST response should contain only the root `<VAST>` element with one or more `<Error>` elements, as shown below:

```
<VAST version="4.1">
      <Error>
            <![CDATA[http://adserver.com/noad.gif]]>
      </Error>
</VAST>
```

The VAST `<Error>` element is optional but if included, the media player must use the URI provided to notify the server that no ad was returned. Multiple <Error> elements may be provided to notify multiple parties of the no-ad response.

Player Support	Required
Required in Response	No, but if supplied no other elements are allowed
Parent	VAST
Bounded	0+
Contents	A URI supplied by the ad server and used to report the no ad response

3.3 Ad

TOC Schema

The `<Ad>` element may contain an `<InLine>` ad or a `<Wrapper>`. The wrapper points to a secondary server for another VAST response, which may be another wrapper or an InLine response. An InLine response contains the ad creative necessary to execute ad playback.

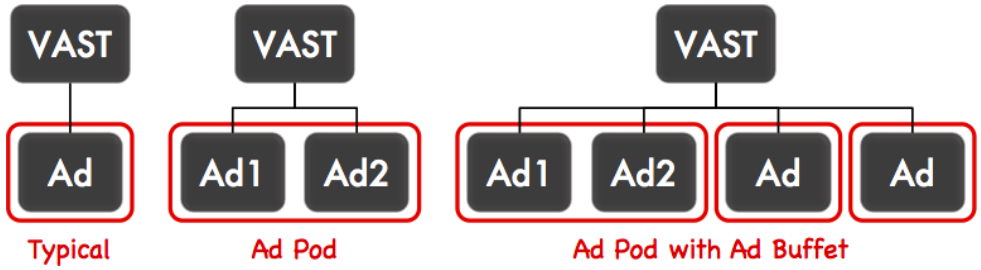
3.3.1 Ad Pods and Stand-Alone Ads

TOC Schema

While a single `<Ad>` element represents the most common VAST response, multiple ads may be included as either stand-alone ads or a Pod of ads, or a mix of both. Ads in a Pod are distinguished by using the `sequence` attribute for an `<Ad>`, denoting which ad plays first, second, and so on. If the player supports Ad Pods, sequenced ads are played in numerical order and all ads in the Pod should be played to the best of the player's ability. All `sequence` values in a VAST response must be unique.

Non-sequenced ads, are stand-alone ads and considered part of an "ad buffet" from which the player may select one or more ad to play in any order. Stand-alone ads may be included in a VAST response with an Ad Pod and may be used to substitute an ad in the Pod when an ad cannot be played.

The following diagram illustrates some options for how the `<Ad>` element may be represented in a VAST response.



If the media player cannot display an entire Ad Pod or any stand-alone ads, it can decline from loading the ad resources and use the error URI, if provided, to notify the server.

Playing a Pod of Ads

When electing to play a Pod of ads returned by the ad server, the media player must play the ads in the Pod in the prescribed sequence and should play as many of the ads as possible. The player may elect to truncate any ads at the end of an Ad Pod if either: the ads cannot be played because they cannot physically fit into the ad break in the stream (such as when time is limited in a live stream) or if the Pod violated any limits specified by the media player request (for example: number of ads to return, or maximum pod duration).

When an Ad Pod is the result of following a VAST `<Wrapper>` the same impression and tracking URIs in the VAST `<Wrapper>` are called as each ad in the Pod is played.

Should an ad in the Pod fail to play, the media player should substitute an un-played stand-alone ad from the response. If stand-alone ads are unavailable, the player should move on to the next ad in the Ad Pod.

If a Wrapper is used to provide an ad in the Ad Pod, the Wrapper can use attributes, `allowMultipleAds=false` and `followAdditionalWrappers=false` to prevent performance issues that result from an unmanaged string of Wrappers and multiple ads in an Ad Pod.

Media Player Implementation Note	If multiple `<Ad>` elements are provided with `sequence` attributes and the player supports Ad Pods, all ads in the Pod must be played to the best of the player's ability. If not supported or the Pod cannot be played, the media player should use the error-tracking URI, if provided, to notify the server. A special exemption exists when using VMAP. Please visit http://iab.com/vmap for information on VMAP.

3.3.2 The Ad Element

TOC Schema

Properties for the `<Ad>` element are listed in the following table.

Player Support	Required (Ad Pod support is optional)
Required in Response	Yes (unless there is no ad to return; in which case `<Error>` should be used to provide an error URI)
Parent	VAST
Bounded	0+ (In a no ad response, `<Ad>` is not allowed, but in all other cases at least one `<Ad>` is required.)
Sub-elements	InLine* Wrapper*
Attributes	Description
id	An ad server-defined identifier string for the ad
sequence	A integer greater than zero (0) that identifies the sequence in which an ad should play; all `<Ad>` elements with sequence values are part of a pod and are intended to be played in sequence
conditionalAd	[Deprecated in VAST 4.1, along with apiFramework] A Boolean that identifies a conditional ad. In the case of programmatic ad serving, a VPAID ad unit or other mechanism might be used to decide whether there is an ad that matches the placement. When there is no match, an ad may not be served. Use of the `conditionalAd` attribute enables publishers to avoid accepting these ads in placements where an ad must be served. A value of `true` indicates that the ad is conditional and should be used in all cases where the InLine executable unit (such as VPAID) is not an ad but is instead a framework for finding an ad; a value of `false` is the default value and indicates that an ad is available.
adType	An optional string that identifies the type of ad. This allows VAST to support audio ad scenarios. Possible values – video, audio, hybrid. Default value – video (assumed to be video if attribute is not present) More details on the use case in section 1.5

*The Ad element requires exactly one child, which can either be an <InLine> or <Wrapper> element.

3.4 InLine

TOC Schema

Within the nested elements of an `<InLine>` ad are all the files and URIs necessary to play and track the ad. In a chain of `<Wrapper>` VAST responses, an `<InLine>` response ends the chain.

Player Support	Required
Required in Response	One of `<InLine>` or `<Wrapper>` is required, but both are not allowed
Parent	Ad
Bounded	0-1
Sub-elements	AdSystem* AdTitle* Impression* AdServingId* Category Description Advertiser Pricing Survey Error Extensions ViewableImpression AdVerifications Creatives* Expires

*required

3.4.1 AdSystem

TOC Schema

The ad serving party must provide a descriptive name for the system that serves the ad. Optionally, a version number for the ad system may also be provided using the `version` attribute.

Player Support	Required
Required in Response	Yes
Parent	InLine or Wrapper
Bounded	1
Content	A string that provides the name of the ad server that returned the ad
Attributes	Description
version	A string that provides the version number of the ad system that returned the ad

3.4.2 AdTitle

TOC Schema

The ad serving party must provide a title for the ad using the `<AdTitle>` element. If a longer description is needed, the `<Description>` element can be used.

Player Support	Required

Required in Response	Yes
Parent	InLine
Bounded	1
Content	A string that provides a common name for the ad

3.4.3 AdServingId

Any ad server that returns a VAST containing an <InLine> ad must generate a pseudo-unique identifier that is appropriate for all involved parties to track the lifecycle of that ad. This should be inserted into the <AdServingId> element, and also be included on all outgoing tracking pixels. The purpose of this id is to greatly reduce the amount of work required to compare impression-level data across multiple systems, which is otherwise done by passing proprietary IDs across different systems and matching them.

This value should be different for each <InLine> in a VAST (i.e. each <Ad> in an Ad Pod or buffet should have distinct <AdServingId> values). The player is responsible for parsing this value and using it to fill the associated macro (see section 6) on tracking pixels.

Using a GUID for the AdServingId value is recommended. Other formats are acceptable, provided they are generally sufficiently unique to allow different systems to match tracking data. Ad servers may also choose to prepend their AdSystem or a shortened version of their server name to ID value, so that the originating server can easily be identified from the ID alone.

Example: ServerName-47ed3bac-1768-4b9a-9d0e-0b92422ab066

Note that only <InLine> elements may provide an <AdServingId>. Servers providing <Wrapper> VASTs may learn the ad serving ID by including the [ADSERVINGID] macro in their tracking pixels.

Player Support	Required
Required in Response	Yes (for InLine)
Parent	InLine
Bounded	1
Content	A unique or pseudo-unique (long enough to be unique when combined with timestamp data) GUID .

3.4.4 Impression

TOC Schema

The ad server provides an impression-tracking URI for either the InLine ad or the Wrapper using the `<Impression>` element. All `<Impression>` URIs in the InLine response and any Wrapper responses preceding it should be triggered at the same time when the impression for the ad occurs, or as close in time as possible to when the impression occurs, to prevent impression-counting discrepancies.

Player Support	Required
Required in Response	Yes
Parent	InLine or Wrapper
Bounded	1+
Content	A URI that directs the media player to a tracking resource file that the media player must use to notify the ad server when the impression occurs. If there is no reason to include an Impression element, the placeholder "about:blank" should be used instead of a tracking URL. The player should disregard dispatching the tracking URI if it is set to "about:blank"..
Attributes	Description
id	An ad server id for the impression. Impression URIs of the same id for an ad should be requested at the same time or as close in time as possible to help prevent discrepancies.

3.4.5 Category

TOC Schema

Used in creative separation and for compliance in certain programs, a category field is needed to categorize the ad's content. Several category lists exist, some for describing site content and some for describing ad content. Some lists are used interchangeably for both site content and ad content. For example, the category list used to comply with the IAB Quality Assurance Guidelines (QAG) describes site content, but is sometimes used to describe ad content.

The VAST category field should only use AD CONTENT description categories.

The `authority` attribute is used to identify the organizational authority that developed the list being used. In some cases, the publisher may require that an ad category be identified. If required by the publisher and not provided, the publisher may skip the ad, notify the ad server using the `<Error>` URI, if provided (error code 204), and move on to the next option.

If category is used, the `authority=` attribute must be provided.

Player Support	Optional
Required in Response	No*
Parent	InLine
Bounded	0+
Content	A string that provides a category code or label that identifies the ad content category.
Attributes	Description
authority *	A URL for the organizational authority that produced the list being used to identify ad content category.

*Optional unless the publisher requires ad categories. The `authority` attribute is required if categories are provided.

Example:
<Category authority="iabtechlab.com">232</Category>
(where the IAB Ad Product Taxonomy is being used, and 232 is the category and maps to a particular category - say Automobiles or Designer Clothing)

3.4.6 Description

TOC Schema

When a longer description of the ad is needed, the `<Description>` element can be used.

Player Support	Required
Required in Response	No
Parent	InLine
Bounded	0-1
Content	A string that provides a long ad description

3.4.7 Advertiser

TOC Schema

Providing an advertiser name can help publishers prevent display of the ad with its competitors. Ad serving parties and publishers should identify how to interpret values provided within this element.

Player Support	Required
Required in Response	No
Parent	InLine
Bounded	0-1
Content	A string that provides the name of the advertiser as defined by the ad serving party. Recommend using the domain of the advertiser.
Attributes	Description
id	An (optional) identifier for the advertiser, provided by the ad server. Can be used for internal analytics.

3.4.8 Pricing

TOC Schema

Used to provide a value that represents a price that can be used by real-time bidding (RTB) systems. VAST is not designed to handle RTB since other methods exist, but this element is offered for custom solutions if needed. If the value provided is to be obfuscated or encoded, publishers and advertisers must negotiate the appropriate mechanism to do so.

When included as part of a VAST Wrapper in a chain of Wrappers, only the value offered in the first Wrapper need be considered.

Player Support	Recommended
Required in Response	No
Parent	InLine or Wrapper
Bounded	0-1
Content	A number that represents a price that can be used in real-time bidding systems
Attributes	Description
model*	Identifies the pricing model as one of: CPM, CPC, CPE, or CPV.
currency*	The three-letter ISO-4217 currency symbol that identifies the currency of the value provided (e.g. USD, GBP, etc.).

*required

3.4.9 Survey

TOC Schema

The survey node is deprecated in VAST 4.1, since usage was very limited and survey implementations can be supported by other VAST elements such as 3rd party trackers.

Ad tech vendors may want to use the ad to collect data for resource purposes. The `<Survey>` element can be used to provide a URI to any resource file having to do with collecting survey data. Publishers and any parties using the `<Survey>` element should determine how surveys are implemented and executed. Multiple survey elements may be provided.

A `type` attribute is available to specify the MIME type being served. For example, the attribute might be set to `type="text/javascript"`. Surveys can be dynamically inserted into the VAST response as long as cross-domain issues are avoided.

Player Support	Recommended
Required in Response	No
Parent	InLine
Bounded	0+
Content	A URI to any resource relating to an integrated survey.
Attributes	Description
type	The MIME type of the resource being served.

3.4.10 Expires

TOC Schema

The number of seconds in which the ad is valid for execution. In cases where the ad is requested ahead of time, this timing indicates how many seconds after the request that the ad expires and cannot be played. This element is useful for preventing an ad from playing after a timeout has occurred.

If no value is provided, the response can be played back at any time indefinitely after being received by the player.

Player Support	Recommended
Required in Response	No
Parent	InLine
Bounded	0-1
Content	An integer value that defines the expiry period (in seconds).

3.4.11 Error (InLine and Wrapper)

TOC Schema

The `<Error>` element contains a URI that the player uses to notify the ad server when errors occur with ad playback. If the URI contains an `[ERRORCODE]` macro, the media player must populate the macro with an error code as defined in section 2.3.6.

If no specific error can be found, error 900 may be used to indicate an undefined error; however, every attempt should be made to provide an error code that maps to the error that occurred. The `<Error>` element is available for both the InLine or Wrapper elements.

Player Support	Required
Required in Response	No
Parent	InLine or Wrapper
Bounded	0+
Content	A URI to a tracking resource to be used when an error in ad playback occurs.

3.5 ViewableImpression

TOC Schema

The ad server may provide URIs for tracking publisher-determined viewability, for both the InLine ad and any Wrappers, using the `<ViewableImpression>` element. Tracking URIs may be provided in three containers: `<Viewable>`, `<NotViewable>`, and `<ViewUndetermined>`.

The point at which these tracking resource files are pinged depends on the viewability standard the player has implemented, in agreement with or with the understanding of the buyer.

Player support for the `<ViewableImpression>` element is optional. When used, URIs for the Inline ad as well as any wrappers used to serve the ad should all be triggered at the same time, or as close in time as possible to when the criteria for the individual event is met.

Note – ViewableImpression is not applicable for Audio use cases.

Player Support	Optional
Required in Response	No
Parent	Inline or Wrapper
Bounded	0-1
Sub-elements	Viewable NotViewable ViewUndetermined
Attributes	Description
id	An ad server id for the impression. Viewable impression resources of the same id should be requested at the same time, or as close in time as possible, to help prevent discrepancies.

3.5.1 Viewable

TOC Schema

The `<Viewable>` element is used to place a URI that the player triggers if and when the ad meets criteria for a viewable video ad impression.

Player Support	Optional
Required in Response	No
Parent	ViewableImpression
Bounded	0+
Content	A URI that directs the media player to a tracking resource file that the media player should request at the time that criteria is met for a viewable impression.

3.5.2 NotViewable

TOC Schema

The `<NotViewable>` element is a container for placing a URI that the player triggers if the ad is executed but never meets criteria for a viewable video ad impression.

Player Support	Optional
Required in Response	No
Parent	ViewableImpression
Bounded	0+
Content	A URI that directs the media player to a tracking resource file that the media player should request if the ad is executed but never meets criteria for a viewable impression.

3.5.3 ViewUndetermined

TOC Schema

The `<ViewUndetermined>` element is a container for placing a URI that the player triggers if it cannot determine whether the ad has met criteria for a viewable video ad impression.

Player Support	Optional
Required in Response	No
Parent	ViewableImpression
Bounded	0+
Content	A URI that directs the media player to a tracking resource file that the media player should request if the player cannot determine whether criteria is met for a viewable impression.

3.6 Creatives

TOC Schema

The `<Creatives>` (plural) element is a container for one or more `<Creative>` (singular) element used to provide creative files for the ad. For an InLine ad, the `<Creatives>` element nests all the files necessary for executing and tracking the ad.

In a Wrapper, elements nested under `<Creatives>` are used mostly for tracking. Companion and Icon creative may be included in a Wrapper, but files for Linear and NonLinear ads can only be provided in the InLine version of the ad.

Player Support	Required
Required in Response	Yes
Parent	InLine or Wrapper
Bounded	1 for InLine 0-1 for Wrapper
Sub-elements	Creative

* required

3.7 Creative

TOC Schema

Each `<Creative>` element contains nested elements that describe the type of ad being served using nested sub-elements. Multiple creatives may be used to define different components of the ad. At least one `<Linear>` element is required under the Creative element.

Player Support	Required
Required in Response	Yes
Parent	Creatives for both InLine and Wrapper formats
Bounded	1+
Sub-elements	UniversalAdId* CreativeExtensions Linear CompanionAds
Attributes	Description
id	A string used to identify the ad server that provides the creative.
adId	Used to provide the ad server's unique identifier for the creative. In VAST 4, the UniversalAdId element was introduced to provide a unique identifier for the creative that is maintained across systems. Please see section 3.7.1 for details on the UniversalAdId.
sequence	A number representing the numerical order in which each sequenced creative within an ad should play.
apiFramework	A string that identifies an API that is needed to execute the creative.

*required

General Implementation Note	The Creative `sequence` attribute should not be confused with the Ad `sequence` attribute. Creative `sequence` identifies the sequence of multiple creative within a single ad and does NOT define a Pod. Conversely, the Ad `sequence` identifies the sequence of multiple ads and defines an Ad Pod. See section 3.3.1 for details about Ad Pods.

3.7.1 UniversalAdId

TOC Schema

A required element for the purpose of tracking ad creative, the `<UniversalAdId>` is used to provide a unique creative identifier that is maintained across systems. This creative ID may be generated with an authoritative program, such as the AD-ID® program in the United

States, or Clearcast in the UK. Some countries may have specific requirements for ad-tracking programs.

The UniversalAdId element is required in 4, but the attribute value for `idRegistry` and the idValue in the node are to be used to support a company's need for tracking ad creative. If no common registry is used, a value of "unknown" may be used. Ad servers and publishers should discuss what is required for this field to support a successful ad campaign.

Note: A creative id can also be included in the `adId` attribute used in the `<Creative>` element, but that creative id should be used to specify the ad server's unique identifier. The UniversalAdId is used for maintaining a creative id for the ad across multiple systems.

Note – VAST 4.0 had an attribute "idValue" that was a duplicate of the node value and so was removed as of 4.1. Media players should not fail if this attribute is present, but should always use the Content as the source of truth for the creative ID value.

Player Support	Required
Required in Response	Yes
Parent	Creative only in the InLine format
Bounded	1+
Content	A string identifying the unique creative identifier. Default value is "unknown"
Attributes	Description
idRegistry*	A string used to identify the URL for the registry website where the unique creative ID is cataloged. Default value is "unknown."

*required

Examples -

<UniversalAdId idRegistry="ad-id.org">CNPA0484000H<UniversalAdId>

<UniversalAdId idRegistry="clearcast.co.uk"> AAA/BBBB123/030<UniversalAdId>

Note: With VAST 4.2 we now allow multiple UniversalAdID elements to be passed in a VAST response

3.7.2 CreativeExtensions

TOC Schema

When an executable file is needed as part of the creative delivery or execution, a `<CreativeExtensions>` element can be added under the `<Creative>`. This extension can be used to load an executable creative with or without using the `<MediaFile>`.

A `<CreativeExtension>` (singular) element is nested under the `<CreativeExtensions>` (plural) element so that any XML extensions are separated from VAST XML. Additionally, any XML used in this extension should identify an XML name space (xmlns) to avoid confusing any of the extension element names with those of VAST.

The nested `<CreativeExtension>` includes an attribute for `type`, which specifies the MIME type needed to execute the extension.

Player Support	Recommended
Required in Response	No
Parent	Creative only in the InLine format
Bounded	0-1
Sub-elements	CreativeExtension

3.7.3 CreativeExtension

TOC Schema

Used as a container under the CreativeExtensions element, this node is used to delineate any custom XML object that might be needed for ad execution.

Player Support	Recommended
Required in Response	No
Parent	CreativeExtensions only in the InLine format
Bounded	0+
Content	Custom XML object
Attributes	Description
type	The MIME type of any code that might be included in the extension.

3.8 Linear

TOC Schema

Linear Ads are the video or audio formatted ads that play linearly within the streaming content, meaning before the streaming content, during a break, or after the streaming content. A Linear creative contains a `<Duration>` element to communicate the intended runtime and a `<MediaFiles>` element used to provide the needed video or audio files for ad execution.

Player Support	Required
Required in Response	Yes
Parent	Creative for both InLine and Wrapper formats
Bounded	0-1
Sub-elements	Duration* MediaFiles* AdParameters TrackingEvents VideoClicks Icons
Attributes	Description
skipoffset	Time value that identifies when skip controls are made available to the end user; publisher may define a minimum `skipoffset` value in its policies and disregard Skippable creative when `skipoffset` values are lower than publisher's minimum.

*required

3.8.1 Duration

TOC Schema

The ad server uses the `<Duration>` element to denote the intended playback duration for the video or audio component of the ad. Time value may be in the format HH:MM:SS.mmm where .mmm indicates milliseconds. Providing milliseconds is optional.

Player Support	Required
Required in Response	Yes
Parent	Linear only in the InLine format
Bounded	1
Content	A time value for the duration of the Linear ad in the format HH:MM:SS.mmm (.mmm is optional and indicates milliseconds).

3.8.2 AdParameters

TOC Schema

Some ad serving systems may want to send data to the media file when first initialized. For example, the media file may use ad server data to identify the context used to display the creative, what server to talk to, or even which creative to display. The optional `<AdParameters>` element for the Linear creative enables this data exchange.

The optional attribute `xmlEncoded` is available for the `<AdParameters>` element to identify whether the ad parameters are xml-encoded. If true, the media player can only decode the data using XML. Media players operating on earlier versions of VAST may not be able to

XML-decode data, so data should only be xml-encoded when being served to media players capable of XML-decoding the data.

When a VAST response is used to serve a VPAID ad unit, the `<AdParameters>` element is currently the only way to pass information from the VAST response into the VPAID object; no other mechanism is provided.

Player Support	Required
Required in Response	No
Parent	Linear only in the InLine format Companion for both InLine and Wrapper formats
Bounded	0-1
Content	Metadata for the ad.
Attributes	Description
xmlEncoded	Identifies whether the ad parameters are xml-encoded.

3.9 MediaFiles

TOC Schema

Since the first version of VAST, the MediaFiles element was designated for linear video files. Over the years as digital video technology advanced, the media files placed in a VAST tag have come to include complex files that require API integration. Players not equipped with the technology to execute such files may be unable to play the ad or execute interactive components. In ads that require API integration, VAST 4 separates media and interactive files. While <MediaFiles> node focus shifts to the exclusive delivery of media (video and audio), the dedicated <InteractiveCreativeFile> element (see section 3.9.3) opens opportunities for rendering modern secure interactive components in parallel with video and audio assets. The dedicated <Verification> element allows for measurement capabilities. Disjoining media and executable files enables a wider range of players to consume enhanced ads as well as performance improvements.

It is worth noting that when multiple *MediaFile* nodes are present, the publisher should decide which file to play based on attributes of the *MediaFile* nodes and not the structure of the document (e.g. defaulting to the first *MediaFile* included in the document).

Linear media files should be submitted as follows:

> **Video/Audio file only:** Include three `<MediaFile>` elements (section 3.9.1), each with a URI to a ready-to-serve video or audio file at quality levels for high, medium, and low. Please review the IAB Digital Video Ad Format Guidelines for guidance on ready-to-serve file quality specifications.
>
> **Video/Audio file for use in ad-stitching:** In addition to the three ready-to-serve files, use the `<Mezzanine>` element (section 3.9.2) to include a URI to the raw video or audio file. Please review the IAB Digital Video Ad Format Guidelines for guidance on mezzanine file specifications.

Interactive linear video file: In addition to at least one ready-to-serve video/audio file included in the `<MediaFile>` element, use the `<InteractiveCreativeFile>` element (section 3.9.3) to include a URI to the interactive media file, specifying the API framework required to execute the file.

The components of the `<MediaFiles>` elements:

Player Support	Required if `<Linear>` is supported
Required in Response	Yes (Linear ads)
Parent	Linear only for InLine format
Bounded	1 (When `<Linear>` is used)
Sub-elements	MediaFile* Mezzanine** InteractiveCreativeFile ClosedCaptionFiles

*required **required in ad-stitched video executions

3.9.1 MediaFile

TOC Schema

In VAST 4.x `<MediaFile>` should only be used to contain the video or audio file for a Linear ad. In particular, three ready-to-serve files should be included, each of a quality level for high, medium, or low. A ready-to-serve video/audio file is a media that is transcoded to a level of quality that can be transferred over an internet connection within a reasonable time for viewing. Each ready-to-serve file must be of the same MIME type and, if different MIME types files are made available for the ad, three ready-to-serve files should represent each MIME type separately.

When an interactive API is needed to deliver and execute the Linear Ad, the URI to the interactive file should be included in the `<InteractiveCreativeFile>`. In addition, at least one ready-to-serve video ad should be available in `<MediaFile>` so that the video ad can be played by the video player.

Guidelines for ad files that fulfill quality levels of high, medium, or low can be found in the IAB Digital Video Ad Format Guidelines. An adaptive bitrate streaming file featuring files at the three quality levels may also be provided.

Player Support	Required
Required in Response	Yes
Parent	MediaFiles only for InLine format
Bounded	1+
Content	A CDATA-wrapped URI to a media file.
Attributes	Description
delivery*	Either “progressive” for progressive download protocols (such as HTTP) or “streaming” for streaming protocols.
type*	MIME type for the file container. Popular MIME types include, but are not limited to “video/mp4” for MP4, “audio/mpeg” and "audio/aac" for audio ads.
width*	The native width of the video file, in pixels. (0 for audio ads)
height*	The native height of the video file, in pixels. (0 for audio ads)
codec	The codec used to encode the file which can take values as specified by RFC 4281: http://tools.ietf.org/html/rfc4281.
id	An identifier for the media file.

bitrate or minBitrate and maxBitrate	For progressive load video or audio, the `bitrate` value specifies the average bitrate for the media file; otherwise the `minBitrate` and `maxBitrate` can be used together to specify the minimum and maximum bitrates for streaming videos or audio files.
scalable	a Boolean value that indicates whether the media file is meant to scale to larger dimensions.
maintainAspectRatio	a Boolean value that indicates whether aspect ratio for media file dimensions should be maintained when scaled to new dimensions.
apiFramework**	[Deprecated in 4.1 in preparation for VPAID being phased out] identifies the API needed to execute an interactive media file, but current support is for backward compatibility. Please use the `<InteractiveCreativeFile>` element to include files that require an API for execution.
fileSize	Optional field that helps eliminate the need to calculate the size based on bitrate and duration. Units - Bytes
mediaType	Type of media file (2D / 3D / 360 / etc). Optional. Default value = 2D

*required
**if an API framework is needed to execute the ad, please use `<InteractiveCreativeFile>` to provide API files.

3.9.2 Mezzanine

TOC Schema

The media player may use a raw mezzanine file to transcode video or audio files at quality levels specific to the needs of certain environments. An XSD will validate this element as optional, but a mezzanine file is required in ad-stitched executions and whenever a publisher requires it. If no mezzanine file is available, this element may be excluded; however, publishers that require it may ignore the VAST response when not provided. If an ad is rejected for this reason, error code 406 is available to communicate the error when an `<Error>` URI and macro are provided.

Publishers consume mezzanine files to transcode the media into a form publisher's system and user devices support. The mezzanine file should never be used for the direct ad playback.

.

The mezzanine file specifications are defined in the Digital Video Ad Format Guidelines.

Player Support	Optional
Required in Response	No*
Parent	MediaFiles only in InLine format.
Bounded	0+
Content	A CDATA-wrapped URI to a raw, high-quality media file.
Attributes	Description
delivery*	Either `progressive` for progressive download protocols (such as HTTP) or streaming for streaming protocols.

type*	MIME type for the file container. Popular MIME types include, but are not limited to “video/mp4” for MP4, “audio/mpeg” and "audio/aac" for audio ads.
width*	The native width of the video file, in pixels.
height*	The native height of the video file, in pixels.
codec	The codec used to encode the file which can take values as specified by RFC 4281: http://tools.ietf.org/html/rfc4281.
id	An identifier for the media file.
fileSize	Optional field that helps eliminate the need to calculate the size based on bitrate and duration.
mediaType	Type of media file (3D / 360 / etc). Optional. Default value = 2D

* VAST tags served to ad-stitching servers require a mezzanine file; server may reject the VAST response if no mezzanine file is provided.

3.9.3 InteractiveCreativeFile

TOC Schema

For any media file that uses interactive APIs for advanced creative functionality, the `<InteractiveCreativeFile>` element is used to identify the file and the framework needed for execution.

Providing the interactive portion for a media file in a section of VAST separate from the video/audio file enables players to more easily play the video/audio file when no support is available to execute the API, especially for players that work with an ad-stitching service or make ad calls from a server on behalf of the player.

The player should attempt to execute the interactive file before attempting to load any `<MediaFile>`, but if the file cannot be executed, the player should trigger any included error URIs and use error code 409 when macros are provided.

The “file” can be a direct URL or can be an inline data URI. While providing a URL was customary for VAST before 4.3, saving connection time by providing a small, inline, data URI enhances the consumer ad experience by saving connection times. This saving is only possible if the inline data URI is small enough. Since the data URI payload will likely be a HTML payload, the recommendation is to make this as small as possible. Once the size of the data URI becomes prohibitively large, use a URL to save the response size of the VAST, itself.

Note re Audio Ads: While not in use today, this could be used for pure audio interactivity outside of a click on devices like Alexa.

Player Support	Required
Required in Response	No
Parent	MediaFiles only in InLine format
Bounded	0+
Content	A CDATA-wrapped URI or inline data URI to a file providing creative functions for the media file.
Attributes	Description
type	Identifies the MIME type of the file provided.
apiFramework	Identifies the API needed to execute the resource file if applicable.

variableDuration	`Boolean`. Useful for interactive use cases. Identifies whether the ad always drops when the duration is reached, or if it can potentially extend the duration by pausing the underlying video or delaying the adStopped call after adVideoComplete. If set to `true`, the extension of the duration should be user-initiated (typically, by engaging with an interactive element to view additional content).

3.9.4 ClosedCaptionFiles

TOC Schema

Optional node that enables closed caption sidecar files associated with the ad media (video or audio) to be provided to the player. Multiple files with different mime-types may be provided as children of this node to allow the player to select the one it is compatible with.

Note: It is expected that all the media files tied to parent MediaFiles node are associated with the same original creative and therefore of the same media length as well as accurately synchronized with closed captioned media segments times, so all the files under ClosedCaptionFiles should work for all the MediaFile nodes.

Player Support	Optional
Required in Response	No
Parent	MediaFiles only in InLine format
Bounded	1
Sub elements	ClosedCaptionFile

3.9.5 ClosedCaptionFile

TOC Schema

Individual closed caption files for various languages.

Player Support	Optional
Required in Response	No
Parent	ClosedCaptionFiles
Bounded	0+
Content	A CDATA-wrapped URI to a file providing Closed Caption info for the media file.
Attributes	Description
type	Identifies the MIME type of the file provided.
language	Language of the Closed Caption File using ISO 631-1 codes. An optional locale suffix can also be provided. Example:- “en”, “en-US”, “zh-TW”.

Examples

```
<MediaFiles>
…
        <ClosedCaptionFiles>
                <ClosedCaptionFile type="text/srt  language="en">
                        <![CDATA[https://mycdn.example.com/creatives/creative001.srt]]>
                </ClosedCaptionFile>
                <ClosedCaptionFile type="text/srt" language="fr">
                        <![CDATA[https://mycdn.example.com/creatives/creative001-1.srt]]>
                </ClosedCaptionFile>
                <ClosedCaptionFile type="text/vtt  language="zh-TW">
                        <![CDATA[https://mycdn.example.com/creatives/creative001.vtt]]>
                </ClosedCaptionFile>
                <ClosedCaptionFile type="application/ttml+xml"  language= zh-CH">
```

```
                    <![CDATA[https://mycdn.example.com/creatives/creative001.ttml]]>
                </ClosedCaptionFile>
            </ClosedCaptionFiles>
...
</MediaFiles>
```

3.10 VideoClicks

TOC Schema

The `<VideoClicks>` element provides URIs for `clickthroughs`, `clicktracking`, and `custom` clicks and is available for Linear Ads in both the InLine and Wrapper formats. Both InLine and Wrapper formats offer the ClickThrough, ClickTracking and CustomClick elements. These elements are defined in the following sections.

Player Support	Required
Required in Response	No
Parent	Linear in both the InLine and Wrapper format
Bounded	0-1
Sub-elements	ClickThrough ClickTracking CustomClick

3.10.1 ClickThrough

TOC Schema

The `<ClickThrough>` is a URI to the advertiser's site that the media player opens when a viewer clicks the ad. The clickthrough is available in the InLine and Wrapper formats and is used when the Linear ad unit cannot handle a clickthrough.

Player Support	Required
Required in Response	One ClickThrough element is required if `<VideoClicks>` in the InLine format is used.
Parent	Linear in both the InLine and Wrapper format
Bounded	0-1 (if `<VideoClicks>` is used)
Content	a URI to the advertiser's site that the media player opens when a viewer clicks the ad.
Attributes	Description
id	A unique ID for the clickthrough.

3.10.2 ClickTracking

TOC Schema

Multiple `<ClickTracking>` elements can be used in the case where multiple parties would like to track the Linear ad clickthrough.

Player Support	Required
Required in Response	No

Parent	Linear in both InLine and Wrapper formats.
Bounded	0+
Content	A URI for tracking when the ClickThrough is triggered.
Attributes	Description
id	A unique ID for the click to be tracked.

3.10.3 CustomClick

TOC Schema

The `<CustomClick>` is used to track any interactions with the linear ad that do not include the clickthrough click and do not take the viewer away from the media player. For example, if an ad vendor wants to track that a viewer clicked a button to change the ad's background color, the `<CustomClick>` element holds the URI to notify the ad vendor that this click happened. An API may be needed to inform the player that a click occurred and that the corresponding URI should be activated.

Player Support	Required
Required in Response	No
Parent	Linear in both Wrapper and InLine formats.
Bounded	0+
Content	A URI for tracking custom interactions.
Attributes	Description
id	A unique ID for the custom click to be tracked.

3.11 Icons

TOC Schema

The industry icon feature was defined in VAST 3.0 to support initiatives such as privacy programs. An example of such a program is the AdChoices program for interest-based advertising (IBA). Though the VAST icon feature was initially created to support privacy programs, it was designed to support other programs that require posting an icon with the linear ad.

This feature is only offered for Linear Ads because icons can be easily inserted in NonLinear ads and companion creative using existing features. Icon source files may also be included in a wrapper if necessary.

The structure for Linear icons uses the `<Icons>` element (plural) as a container for one or more `<Icon>` elements (singular). Each `<Icon>` element provides containers for the creative resource file in section 3.15. Icon tracking is described in sections 3.11.2 to 3.11.5.

See section 2.3.7 for details about industry icon support in VAST.

Player Support	Required
Required in Response	No
Parent	`Linear` in both InLine and Wrapper formats
Bounded	0-1

Sub-elements	Icon

3.11.1 Icon

TOC Schema

Nested under the `<Icons>` element, the Icon is used to provide one or more creative files for the icon that represents the program being implemented along with any icon tracking elements. Multiple `<Icon>` elements may be used to represent multiple programs.

Player Support	Required
Required in Response	Yes, if at least one `<Icons>` element is provided
Parent	Icons for both InLine and Wrapper formats
Bounded	1+ (if `<Icons>` is used)
Sub-Elements	StaticResource IFrameResource HTMLResource IconClicks IconViewTracking
Attributes	Description
program	The program represented in the icon (e.g. "AdChoices").
width	Pixel width of the icon asset.
height	Pixel height of the icon asset.
xPosition	The x-coordinate of the top, left corner of the icon asset relative to the ad display area. Values of "left" or "right" also accepted and indicate the leftmost or rightmost available position for the icon asset.
yPosition	The y-coordinate of the top left corner of the icon asset relative to the ad display area; values of "top" or "bottom" also accepted and indicate the topmost or bottommost available position for the icon asset.
duration	The duration the icon should be displayed unless clicked or ad is finished playing; provided in the format HH:MM:SS.mmm or `HH:MM:SS` where `.mmm` is milliseconds and optional.
offset	The time of delay from when the associated linear creative begins playing to when the icon should be displayed; provided in the format `HH:MM:SS.mmm` or `HH:MM:SS`.
apiFramework	Identifies the API needed to execute the icon resource file if applicable.
pxratio	The pixel ratio for which the icon creative is intended. The pixel ratio is the ratio of physical pixels on the device to the device-independent pixels. An ad intended for display on a device with a pixel ratio that is twice that of a standard 1:1 pixel ratio would use the value "2 " Default value is "1 ".
altText	Alternative text for the image. In an html5 image tag this should be the text for the alt attribute. This should enable screen readers to properly read back a description of the icon for visually impaired users.
hoverText	Hover text for the image. In an html5 image tag this should be the text for the title attribute.

3.11.2 IconViewTracking

TOC Schema

The view tracking for icons is used to track when the icon creative is displayed. The player uses the included URI to notify the icon server when the icon has been displayed.

Player Support	Required
Required in Response	No
Parent	Icon for both InLine and Wrapper formats
Bounded	0+
Content	A URI for the tracking resource file to be called when the icon creative is displayed.

3.11.3 IconClicks

TOC Schema

The `<IconClicks>` element is a container for `<IconClickThrough>` and `<ClickTracking>`.

Player Support	Required
Required in Response	No
Parent	Icon for both InLine and Wrapper formats
Bounded	0-1
Sub-elements	IconClickThrough IconClickTracking IconClickFallbackImages

3.11.4 IconClickThrough

TOC Schema

The `<IconClickThrough>` is used to provide a URI to the industry program page that the media player opens when the icon is clicked.

Player Support	Required
Required in Response	No
Parent	IconClicks for both InLine and Wrapper formats
Bounded	0-1
Content	A URI to the industry program page opened when a viewer clicks the icon.

3.11.5 IconClickTracking

TOC Schema

`<IconClickTracking>` is used to track click activity within the icon.

Player Support	Required
Required in Response	No
Parent	IconClicks for both InLine and Wrapper formats
Bounded	0+
Content	A URI to the tracking resource file to be called when a click corresponding to the `id` attribute (if provided) occurs.
Attributes	Description
id	An id for the click to be measured.

3.11.6 IconClickFallbackImages

TOC Schema

The <IconClickFallbackImages> element is used to provide information disclosure for platforms which do not support HTML rendering, by baking the information into an image. This is a fallback for when the buyer cannot rely on <IconClickThrough> for disclosure. When an icon click occurs, the ad must pause and the image must be rendered above the

video. The player must provide a means for the user to close the dialogue, for example by pressing the back button. The image must not be obstructed and should not be downloaded unless a click-action occurs.

Player Support	Recommended
Required in Response	No
Parent	IconClicks for both InLine and Wrapper formats
Bounded	0-1
Sub-elements	IconClickFallbackImage

3.11.6.1 IconClickFallbackImage

TOC Schema

The <IconClickFallbackImage> element is used to display information when an icon click occurs.

Player Support	Recommended
Required in Response	No
Parent	IconClickFallbackImages
Bounded	1+
Sub-elements	AltText StaticResource
Attributes	Description
width*	Pixel width of the image asset
height*	Pixel height of the image asset

3.12 NonLinearAds

TOC Schema

NonLinear ads are the overlay ads that display as an image or rich media on top of video content during playback. Within an InLine ad, at least one of `<Linear>` or `<NonLinearAds>` needs to be provided within the `<Creative>` element.

NonLinearAds are not applicable to Audio use cases.

The `<NonLinearAds>` element is a container for the `<NonLinear>` creative files and tracking resources. If used in a wrapper, only the tracking elements are available. NonLinear creative cannot be provided in a wrapper ad. NonLinear ad creative use non-video creative files that are described in section 3.15. Tracking event elements are described in section 3.14. Ad parameters are used to provide contextual information to the ad and are described in section 3.8.2.

Player Support	Optional (either `<Linear>` or `<NonLinearAds>` must be supported)
Required in Response	At least one of Linear> or NonLinearAds is required.
Parent	Creative for both InLine and Wrapper formats
Bounded	0-1
Sub-elements	NonLinear TrackingEvents

3.12.1 NonLinear

TOC Schema

Each `<NonLinear>` element may provide different versions of the same creative using the `<StaticResource>`, `<IFrameResource>`, and `<HTMLResource>` elements in the InLine VAST response. In a Wrapper response, only tracking elements may be provided.

Player Support	Required if `<NonLinearAds>` is supported
Required in Response	Yes if `<NonLinearAds>` is provided
Parent	NonLinearAds for both InLine and Wrapper formats
Bounded	1+ (if `<NonLinearAds>` is used)
Sub-elements	StaticResource (InLine only) IFrameResource (InLine only) HTMLResource (InLine only) AdParameters (InLine only) NonLinearClickThrough (InLine only) NonLinearclickTracking (InLine and Wrapper)
Attributes	Description
id	An optional identifier for the creative.
width*	The pixel width of the placement slot for which the creative is intended.
height*	The pixel height of the placement slot for which the creative is intended
expandedWidth	The maximum pixel width of the creative in its expanded state.
expandedHeight	The maximum pixel height of the creative in its expanded state.
scalable	Identifies whether the creative can scale to new dimensions relative to the video player when the video player is resized
maintainAspectRatio	Identifies whether the aspect ratio of the creative should be maintained when it is scaled to new dimensions as the video player is resized
apiFramework	The API necessary to communicate with the creative if available.
minSuggestedDuration	The minimum suggested duration that the creative should be displayed; duration is in the format HH:MM:SS.mmm (where .mmm is in milliseconds and is optional)

3.12.2 NonLinearClickThrough

TOC Schema

Most NonLinear creative can provide a clickthrough of their own, but in the case where the creative cannot provide a clickthrough, such as with a simple static image, the `<NonLinearClickThrough>` element can be used to provide the clickthrough.

A clickthrough may need to be provided for an InLine ad in the following situations:

- Static image file
- Any static resource file where the media player handles the click, such as when “playerHandles=true” in a VPAID AdClickThru event.

`NonLinearClickThrough` is only available for InLine ads.

Player Support	Required if `<NonLinearAds>` is supported
Required in Response	No
Parent	NonLinear only in InLine format
Bounded	0-1
Content	A URI to the advertiser’s page that the media player opens when the viewer clicks the NonLinear ad.

3.12.3 NonLinearClickTracking

TOC Schema

When the NonLinear ad creative handles the clickthrough in an InLine ad, the `<NonLinearClickTracking>` element is used to track the click, provided the ad has a way to notify the player that that ad was clicked, such as when using a VPAID ad unit. The NonLinearClickTracking element is also used to track clicks in Wrappers.

NonLinearClickTracking might be used for an InLine ad when:

- Any static resource file where the media player handles the click, such as when "playerHandles=true" in a VPAID AdClickThru event.

NonLinearClickTracking is used in a Wrapper Ad in the following situations:

- Static image file
- Flash file with no API framework (deprecated)
- Flash file in which apiFramework=clickTAG (deprecated)
- Any static resource file where the media player handles the click, such as when "playerHandles=true" in a VPAID AdClickThru event

Player Support	Required if `<NonLinearAds>` is supported
Required in Response	No
Parent	NonLinear for both InLine and Wrapper formats
Bounded	0+
Content	A URI to a tracking resource file used to track a NonLinear clickthrough
Attributes	Description
id	An id provided by the ad server to track the click in reports.

3.13 CompanionAds

TOC Schema

Companion Ads are secondary ads included in the VAST tag that accompany the video/audio ad. The `<CompanionAds>` element is a container for one or more `<Companion>` elements, where each Companion element provides the creative files and tracking details. Companion Ads, including any creative, may be included in both InLine and Wrapper formatted VAST ads.

The `required` attribute for the `<CompanionAds>` element provides information about which Companion creative to display when multiple Companions are supplied and whether the ad can be displayed without its Companion creative. The value for `required` can be one of three values: all, any, or none.

The expected behavior for displaying Companion ads depends on the following values:

- **all:** the media player must attempt to display the contents for all `<Companion>` elements provided. If all companion creative cannot be displayed, the ad should be disregarded and the ad server should be notified using the `<Error>` element.
- **any:** the media player must attempt to display content from at least one of the `<Companion>` elements provided (i.e. display the one with dimensions that best fit the page). If none of the companion creative can be displayed, the ad should be disregarded and the ad server should be notified using the `<Error>` element.
- **none:** the media player may choose to not display any of the companion creative, but is not restricted from doing so. The ad server may use this option when the advertiser prefers that the master Linear or NonLinear ad be displayed even if the companion cannot be displayed.

If not provided, the media player can choose to display content from any or none of the `<Companion>` elements.

VAST 4.1 includes a new attribute for companions: *renderingMode*.

Previous versions of VAST did not allow for the ad server to specify how and when companions would be shown. An asset and width/height information were included, and the assumption was that the companion would be shown alongside the video as a banner. This usage of companions has dropped out of favor over time, while a new usage of the companion as an asset to be displayed full-screen after the video has gained favor in mobile in-app inventory. The renderingMode attribute accommodates the newer end-card use case as part of VAST while laying the groundwork for additional uses of the companion.

The renderingMode attribute accepts a few values. The publisher player/SDK has control of which of these renderingMode values are supported and this should be communicated as part of the publisher ad format specs.

Companion as End-Card

A value of "end-card" signals to the player that this companion ad is meant to be shown after the video stops playing. The end-card should match the dimensions of the preceding video. If the companion width and height are not zero, the player may use these values to infer the aspect ratio of the companion ad.

Companion duration is a new consideration for the end-card and assumed to be controlled by the publisher player/SDK and communicated as part of the publisher ad format specs. Known variations in market include an “infinite” duration, which requires the viewer to close the end-card after it is shown, and a timed duration. For any companion that suspends content playback, such as an in-stream ad, and does not include a time-out, the player/SDK must implement a close control to prevent users from being trapped in the ad. For out-stream ads that do not interfere with content, the close control is not mandatory.

It is also up to the publisher whether a skippable video should show an associated end-card when the video is skipped. Most implementations by major mobile SDKs currently do so.

Click-throughs triggered from the companion should make sure to open in a new browser window rather than replacing the existing end card or another window needed by the app. This ensures that the consumer can exit the webpage that’s loaded upon clicking through the ad and to make sure that the app experience isn’t disrupted.

The VAST event of closeLinear must be fired upon the companion closing. This allows for ads that use companions to know when the companion was dismissed.

Companies providing the end card creative should adhere to IAB Tech Lab LEAN guidelines.

Companion as Concurrent Display Ad

A value of "concurrent" signals to the player that this companion ad is meant to be shown alongside the video for the duration of the video playback. This reflects the original use of the companion in desktop inventory.

Additional Creative Uses of the Companion Ad

The companion ad may be used for new implementations, and as such new values of the renderingMode attribute may be used if supported by the publisher and the ad server. The renderingMode may use other values other than the ones listed to support these additional use cases.

For example, proprietary formats that show content alongside a video could be supported by the standard with a “split-screen” renderingMode, displaying a 1:1 aspect ratio video, alongside an equal sized companion in both portrait and landscape mode.

The goal is that renderingMode will provide some initial standards support for format innovation in environments that cannot, or will not, support VPAID with future spec changes to follow market developments.

Default or Empty renderingMode

The renderingMode attribute may be omitted. In this case, the player will assume that the renderingMode value is set as “default” and will handle the companion in whatever way it does by default.

Non-Creative Use of the Companion Ad

The companion is intended to be used as an additional creative element. Inclusion of a companion to support non-creative functionality (e.g. additional tracking) is considered to be contrary to the intention of the spec.

Player Support	Optional
Required in Response	No
Parent	Creative for both InLine and Wrapper formats
Bounded	0-1
Sub-elements	Companion
Attributes	Description
required	Accepts one of the following values: “all” “any” or “none.” See descriptions listed in this section.

3.13.1 Companion

TOC Schema

Both InLine and Wrapper VAST responses may contain multiple companion items where each one may contain one or more creative resource files using the elements: `StaticResource`, `IFrameResource`, and `HTMLResource`. Each `<Companion>` element may provide different versions of the same creative.

The resource elements for providing creative resources are defined in section 3.15. Tracking elements are also available for each companion element. Ad parameters are used to provide contextual information to the ad and are described in section 3.8.2.

Player Support	Required if `<CompanionAds>` is supported
Required in Response	At least one Companion is required if `CompanionAds` is provided
Parent	CompanionAds for both InLine and Wrapper formats
Bounded	1+ if `<CompanionAds>` is used
Sub-elements	StaticResource IFrameResource HTMLResource AdParameters AltText CcompanionClickThrough CompanionClickTracking TrackingEvents
Attributes	Description
width*	The pixel width of the placement slot for which the creative is intended.
height*	The pixel height of the placement slot for which the creative is intended.
id	An optional identifier for the creative.
assetWidth	The pixel width of the creative.
assetHeight	The pixel height of the creative.
expandedWidth	The maximum pixel width of the creative in its expanded state.
expandedHeight	The maximum pixel height of the creative in its expanded state.
apiFramework	The API necessary to communicate with the creative if available.
adSlotId	Used to identify desired placement on a publisher's page. Values to be used should be discussed between publishers and advertisers.
pxratio	The pixel ratio for which the companion creative is intended. The pixel ratio is the ratio of physical pixels on the device to the device-independent pixels. An ad intended for display on a device with a pixel ratio that is twice that of a standard 1:1 pixel ratio would use the value "2." Default value is "1."
renderingMode	Used to indicate when and where to use this companion ad. Values can be "default" or "end-card" or "concurrent". If this field is empty or not given, "default" will be used.

*required

3.13.2 AltText

TOC Schema

The AltText element is used to provide a description of the companion creative when an ad viewer mouses over the ad.

Player Support	Required if `<CompanionAds>` is supported
Required in Response	No
Parent	Companion for both InLine and Wrapper formats
Bounded	0-1
Content	A `string` to describe the creative when an ad viewer mouses over the ad.

3.13.3 CompanionClickThrough

TOC Schema

Most companion creative can provide a clickthrough of their own, but in the case where the creative cannot provide a clickthrough, such as with a simple static image, the CompanionClickThrough element can be used to provide the clickthrough.

A clickthrough may need to be provided for an InLine ad in the following situations:

- Static image file
- Any static resource file where the media player handles the click, such as when "playerHandles=true" in a VPAID AdClickThru event.

Player Support	Required if `<CompanionAds>` is supported
Required in Response	No
Parent	Companion for both InLine and Wrapper formats
Bounded	0-1
Content	A URI to the advertiser's page that the media player opens when the viewer clicks the companion ad.

3.13.4 CompanionClickTracking

TOC Schema

When the companion ad creative handles the clickthrough in an InLine ad, the CompanionClickTracking element is used to track the click, provided the ad has a way to notify the player that that ad was clicked, such as when using a VPAID ad unit. The CompanionClickTracking element is also used in Wrappers to track clicks that occur for the Companion creative in the InLine ad that is returned after one or more wrappers.

CompanionClickTracking might be used for an InLine ad when:

- Any static resource file where the media player handles the click, such as when "playerHandles=true" in a VPAID AdClickThru event

CompanionClickTracking is used in a Wrapper in the following situations:

- Static image file. Any static resource file where the media player handles the click, such as when "playerHandlesClick=true" in VPAID
- Any static resource file where the media player handles the click, such as when "playerHandlesClick=true" in VPAID

Player Support	Required if `<CompanionAds>` is supported
Required in Response	No
Parent	Companion for both InLine and Wrapper formats
Bounded	0+
Content	A URI to a tracking resource file used to track a companion clickthrough
Attributes	Description

id	An id provided by the ad server to track the click in reports.

3.14 Tracking Event Elements

TOC Schema

The `<TrackingEvents>` element is a container for `<Tracking>` elements used to define specific tracking events described in section 3.14.1. Multiple tracking events can be used to help all the relevant parties track the ad's performance. Each tracking event URI should be included one `<Tracking>` element, using the `event` attribute to identify which event is to be tracked.

The following example shows the section of a VAST response that represents 3 tracking events: two start events, each for a different server, and a complete event.

```
<TrackingEvents>
      <Tracking event="start">
            <![CDATA[http://server1.com/start.jpg]]>
      </Tracking>
      <Tracking event="start">
            <![CDATA[http://server2.com/start2.jpg]]>
      </Tracking>
      <Tracking event="progress" offset="3">
            <![CDATA[http://server1.com/progress.jpg]]>
      </Tracking>

      <Tracking event="complete">
            <![CDATA[http://server1.com/complete.jpg]]>
      </Tracking>
</TrackingEvents>
```

3.14.1 Tracking Event Descriptions

TOC Schema

VAST is used to track a number of ad events using the `<TrackingEvents>` and `<Tracking>` elements. Tracking for impressions is covered in section 3.4.3 and clickthroughs are covered in their relevant sections. Review the schema in section 5 to find more details about tracking the different ad types in VAST. Each `<Tracking>` element contains a URI for the tracking resource of one event. The media player uses these URIs to notify the ad server when the identified event occurs.

In some cases the media player cannot detect that an event has occurred unless a third party, such as the ad creative or a verification script, communicates the event through a framework such as OMID or VPAID. For example, the `adExpand` event for NonLinear ads requires the ad to notify the media player that it has expanded. In such cases, the player must support these tracking events to the extent that they support the individual frameworks.

The following list of metrics is derived from the IAB Digital Video In-Stream Ad Metric Definitions where more detailed metric definitions can be found.

The values accepted for tracking events are described in the following list:

Player Operation Metrics (for use in Linear and NonLinear Ads)

- **mute:** the user activated the mute control and muted the creative.
- **unmute:** the user activated the mute control and unmuted the creative.
- **pause:** the user clicked the pause control and stopped the creative.
- **resume:** the user activated the resume control after the creative had been stopped or paused.
- **rewind:** the user activated the rewind control to access a previous point in the creative timeline.
- **skip:** the user activated a skip control to skip the creative.
- **playerExpand:** the user activated a control to extend the player to a larger size. This event replaces the fullscreen event per the 2014 Digital Video In-Stream Ad Metric Definitions.
- **playerCollapse:** the user activated a control to reduce player to a smaller size. This event replaces the exitFullscreen event per the 2014 Digital Video In-Stream Ad Metric Definitions.
- **notUsed**: This ad was not and will not be played (e.g. it was prefetched for a particular ad break but was not chosen for playback). This allows ad servers to reuse an ad earlier than otherwise would be possible due to budget/frequency capping. This is a terminal event; no other tracking events should be sent when this is used. Player support is **optional** and if implemented is provided on a best effort basis as it is not technically possible to fire this event for every unused ad (e.g. when the player itself is terminated before playback).

Linear Ad Metrics

- **loaded**: This event should be used to indicate when the player considers that it has loaded and buffered the creative's media and assets either fully or to the extent that it is ready to play the media
- **start:** This event is used to indicate that an individual creative within the ad was loaded and playback began. As with creativeView, this event is another way of tracking creative playback. Macros defined to describe auto-play and muted states.
- **firstQuartile:** The creative played continuously for at least 25% of the total duration at normal speed.
- **midpoint:** The creative played continuously for at least 50% of the total duration at normal speed.
- **thirdQuartile:** The creative played continuously for at least 75% of the duration at normal speed.
- **complete:** The creative was played to the end at normal speed so that 100% of the creative was played.
- **otherAdInteraction:** An optional metric that can capture all other user interactions under one metric such a s hover-overs, or custom clicks. It should NOT replace clickthrough events or other existing events like mute, unmute, pause, etc.

- **progress:** The creative played for a duration at normal speed that is equal to or greater than the value provided in an additional `offset` attribute for the `<Tracking>` element under Linear ads. Values can be time in the format `HH:MM:SS` or `HH:MM:SS.mmm` or a percentage value in the format `n%`. Multiple progress events with different values can be used to track multiple progress points in the linear creative timeline. This event can be used in addition to, or instead of, the "quartile" events (`firstQuartile`, `midpoint`, `thirdQuartile`, `complete`). The additional `<Tracking>` `offset` value can be used to help track a view when an agreed upon duration or percentage of the ad has played.
- **closeLinear**: The viewer has chosen to close the linear ad unit. This is currently in-use by some of the largest mobile SDKs to mark the dismissal of the end card companion that follows the video, as well as a close of the video itself, if applicable.

NonLinear Ad Metrics

- **creativeView:** Not to be confused with an impression, this event indicates that an individual creative portion of the ad was viewed. An impression indicates that at least a portion of the ad was displayed; however an ad may be composed of multiple creative, or creative that only play on some platforms and not others. This event enables ad servers to track which ad creative are viewed, and therefore, which platforms are more common.
- **acceptInvitation:** The user clicked or otherwise activated a control used to pause streaming content, which either expands the ad within the player's viewable area or "takes-over" the streaming content area by launching an additional portion of the ad. An ad in video format ad is usually played upon acceptance, but other forms of media such as games, animation, tutorials, social media, or other engaging media are also used.
- **adExpand:** The user activated a control to expand the creative.
- **adCollapse:** The user activated a control to reduce the creative to its original dimensions.
- **minimize:** The user clicked or otherwise activated a control used to minimize the ad to a size smaller than a collapsed ad but without fully dispatching the ad from the player environment. Unlike a collapsed ad that is big enough to display it's message, the minimized ad is only big enough to offer a control that enables the user to redisplay the ad if desired.
- **close:** The user clicked or otherwise activated a control for removing the ad, which fully dispatches the ad from the player environment in a manner that does not allow the user to re-display the ad.
- **overlayViewDuration:** The time that the initial ad is displayed. This time is based on the time between the impression and either the completed length of display based on the agreement between transactional parties or a close, minimize, or accept invitation event.
- **otherAdInteraction:** An optional metric that can capture all other user interactions under one metric such a s hover-overs, or custom clicks. It should NOT replace clickthrough events or other existing events like mute, unmute, pause, etc.

Companion Ad Metric

- **creativeView:** Since Companion Ads use browser technology for display, tracking metrics can be built into the creative. The only VAST event available for tracking companion creative is the creativeView event. This event enables ad servers to track when companion creative are viewed.

Interactive Ad Metric

- interactiveStart: With VAST 4, video playback and interactive creative playback now happens in parallel. Video playback and interactive creative start may not happen at the same time. A separate way of tracking the interactive creative start is needed. The interactive creative specification (SIMID, etc.) will define when this event should be fired.

3.14.2 TrackingEvents

TOC Schema

The `<TrackingEvents>` element is available for `Linear`, `NonLinear`, and `Companion`, elements in both InLine and Wrapper formats. When the media player detects that a specified event occurs, the media player is required to trigger the tracking resource URI provided in the nested `<Tracking>` element. When the server receives this request, it records the event and the time it occurred.

Player Support	Required under supported ad types
Required in Response	No
Parent	Linear NonLinear Companion
Bounded	0-1
Sub-elements	Tracking

3.14.3 Tracking

TOC Schema

Each `<Tracking>` element is used to define a single event to be tracked. Multiple tracking elements may be used to define multiple events to be tracked, but may also be used to track events of the same type for multiple parties.

When using the progress event, an `offset` attribute for linear ads can be used to notify the ad server when the ad's progress has reached the identified percentage or time value indicated. When percentages are used, the progress event can offer tracking that represent the quartile events (`firstQuartile`, `midpoint`, `thirdQuartile`, and `complete`).

When skippable ads are supported, the progress event is used to identify when the ad counts as a view even if the ad is skipped. For example, if the tracking `offset` is set to 00:00:15 (15 seconds) but the ad is skipped after 20 seconds, then a `creativeView` event may be recorded for the Linear creative.

If adType is “audio” or “hybrid”, progress events should be fired even if the media playback is in the background.

The `offset` attribute is only available for the `<Tracking>` element under `<Linear>`.

Player Support	Required under supported ad types
Required in Response	No
Parent	TrackingEvents for both InLine and Wrapper formats
Bounded	0+
Content	A URI to the tracking resource for the event specified using the `event` attribute.
Attributes	Description
event	A string that defines the event being tracked. Accepted values are listed in section 3.14.1 and differ for `<Linear>`, `<NonLinear>`, and `<Companion>`.
offset	Only available when `<Linear>` is the parent. Accepts values of time in the format `HH:MM:SS` or as a percentage in the format `n%`. When the progress of the Linear creative has matched the value specified, the included URI is triggered. If the duration is not known when the offset is set to a percentage value, the progress event may be ignored.

3.15 Creative Resource Files for Non-Video and Non-Audio Creative

TOC Schema

NonLinear ads, Companions, and Industry Icons are non-video and non-audio creative, so creative files are nested using elements that define the type of creative resource file provided: `StaticResource`, `IFrameResource`, and `HTMLResource`.

These resource nodes are available under the elements: `<NonLinear>`, `<Companion>`, and `<Icon>` in the InLine format; however, in Wrapper format, resource files may only be provided under the `<Companion>` and `<Icon>` elements. NonLinear elements in Wrapper format are only used for tracking, and resource files are not allowed.

Multiple creative files may be included using these components, but each element should contain one or more files to represent different versions of the creative for use in different environments. The media player can choose which file to use when more than one resource file is provided within a single container.

For example, if an ad server wants to submit both a static image and an HTML creative for a NonLinear ad, then the NonLinear portion of the VAST response would be formatted as follows:

```
<NonLinearAds>
      <NonLinear>
            <StaticResource>
                  <![CDATA[http://adserver.com/staticresourcefile.jpg]]>
            </StaticResource>
            <HTMLResource>
                  <![CDATA[<html><body>I'm a html snippet</body></html>]]>
            </HTMLResource>


      </NonLinear>
</NonLinearAds>
```

The three resource file elements are described in the following sections.

3.15.1 StaticResource

TOC Schema

The URI to a static creative file to be used for the ad component identified in the parent element, which is either: <NonLinear>, <Companion>, or <Icon>.

Player Support	Required for <Icon> and for <NonLinear> or <Companion> when supported
Required in Response	One of <StaticResource>, <IFrameResource>, or <HTMLResource> is required if <NonLinear>, <Companion>, or <Icon> is used
Parent	NonLinear, Companion, or Icon in the InLine format Companion or Icon in the Wrapper format (Resource files are not provided for NonLinear ads in a Wrapper)
Bounded	0+
Content	A URI to the static creative file to be used for the ad component identified in the parent element.
Attributes	Description
creativeType*	Identifies the MIME type of the creative provided.

*required

3.15.2 IFrameResource

TOC Schema

The URI to an HTML resource file to be loaded into an iframe by the publisher. Associated with the ad component identified in the parent element, which is either: <NonLinear>, <Companion>, or <Icon>.

Player Support	Required for <Icon> and for <NonLinear> or <Companion> when supported
Required in Response	One of <StaticResource>, <IFrameResource>, or <HTMLResource> is required if <NonLinear>, <Companion>, or <Icon> is used
Parent	NonLinear, Companion, or Icon in the InLine format Companion or Icon in the Wrapper format (Resource files are not provided for NonLinear ads in a Wrapper)
Bounded	0+
Content	A URI to the iframe creative file to be used for the ad component identified in the parent element.

3.15.3 HTMLResource

TOC Schema

A “snippet” of HTML code to be inserted directly within the publisher’s

HTML page code.

Player Support	Required for <Icon> and for <NonLinear> or <Companion> when supported
Required in Response	One of <StaticResource>, <IFrameResource>, or <HTMLResource> is required if <NonLinear>, <Companion>, or <Icon> is used in the Inline format
Parent	NonLinear, Companion, or Icon in the InLine format Companion or Icon in the Wrapper format (Resource files are not provided for NonLinear ads in a wrapper)
Bounded	0+

Content	A HTML code snippet (within a CDATA element)

3.16 AdVerifications

TOC Schema

The `<AdVerifications>` element contains one or more `<Verification>` elements, which list the resources and metadata required to execute third-party measurement code in order to verify creative playback.The `<AdVerifications>` element is used to contain one or more `<Verification>` elements, which are used to initiate a controlled container where code can be executed for collecting data to verify ad playback details.

Player Support	Required
Required in Response	No
Parent	InLine
Bounded	0-1
Sub-elements	Verification

3.17 Verification

TOC Schema

The `<Verification>` element contains the executable and bootstrapping data required to run the measurement code for a single verification vendor. Multiple `<Verification>` elements may be listed, in order to support multiple vendors, or if multiple API frameworks are supported. At least one `<JavaScriptResource>` or `<ExecutableResource>` should be provided. At most one of these resources should selected for execution, as best matches the technology available in the current environment.

If the player is willing and able to run one of these resources, it should execute them BEFORE creative playback begins. Otherwise, if no resource can be executed, any appropriate tracking events listed under the `<Verification>` element must be fired.

Player Support	Required
Required in Response	No
Parent	AdVerifications
Bounded	0+
Sub-elements	JavaScriptResource ExecutableResource TrackingEvents

	VerificationParameters
Attributes	Description
vendor*	An identifier for the verification vendor. The recommended format is [domain]-[useCase], to avoid name collisions. For example, "company.com-omid".

*required

3.17.1 JavaScriptResource

TOC Schema

A container for the URI to the JavaScript file used to collect verification data.

Some verification vendors may provide JavaScript executables which work in non-browser environments, for example, in an iOS app enabled by JavaScriptCore. These resources only require methods of the API framework, without relying on any browser built-in functionality.

Players that execute verification code in a browser or webview context should prefer `browserOptional="false"` resources if both are available, but may also execute `browserOptional="true"` resources. Players that execute verification code in a non-browser environment (e.g. JavaScriptCore) may only execute resources marked `browserOptional="true"`. If only `browserOptional="false"` resources are provided, the player must trigger any provided `verificationNotExecuted` tracking events with reason code 2, to indicate the provided code is not supported (see Section 3.17.4).

Player Support	Optional**
Required in Response	No
Parent	Verification
Bounded	0+
Content	A CDATA-wrapped URI to the JavaScript used to collect data
Attributes	Description
apiFramework*	The name of the API framework used to execute the AdVerification code
browserOptional*	Boolean value. If `true`, this resource is optimized and able to execute in an environment without DOM and other browser built-ins (e.g. iOS' JavaScriptCore).

*required

**while optional, if neither `JavascriptResource` or `ExecutableResource` are executed, the player must trigger the verificationNotExecuted tracking events with reason code 2

```
<JavaScriptResource apiFramework="omid" browserOptional="true">
    <![CDATA[https://verificationvendor.com/omid.js]]>
</JavaScriptResource>
```

3.17.2 ExecutableResource

TOC Schema

A reference to a non-JavaScript or custom-integration resource intended for collecting verification data via the listed apiFramework.

Player Support	Optional**

Required in Response	No
Parent	Verification
Bounded	0+
Content	A CDATA-wrapped reference to the resource. This may be a URI, but depending on the execution environment can be any value which enables the player to load the required verification code.
Attributes	Description
apiFramework*	The name of the API framework used to execute the AdVerification code
type*	The type of executable resource provided. The exact value used should be agreed upon by verification integrators and vendors who are implementing verification in a custom environment.

*required

**while optional, if neither `JavascriptResource` or `ExecutableResource` are executed, the player must trigger the verificationNotExecuted tracking events with reason code 2

3.17.3 TrackingEvents

TOC Schema

The verification vendor may provide URIs for tracking events relating to the execution of their code during the ad session. The player must trigger the request of these URIs in the scenarios listed in section 3.17.4.

Player Support	Required
Required in Response	No
Parent	Verification
Bounded	0-1
Sub-elements	Tracking

3.17.4 Tracking

TOC Schema

Each `<Tracking>` element is used to define a single event to be tracked by the verification vendor. Multiple tracking elements may be used to define multiple events to be tracked, but may also be used to track events of the same type for multiple parties.

One event type is currently supported:

- **verificationNotExecuted**: The player did not or was not able to execute the provided verification code

The following macros should be supported specifically in URIs for this event type (in addition to all macros from the global macro set in section 6).

- **[REASON]** - The reason code corresponding to the cause of the failure.

Reason Code	Description
1	Verification resource rejected. The publisher does not recognize or allow code from the vendor in the parent `<Verification>`.
2	Verification not supported. The API framework or language type of verification resources provided are not implemented or supported by the player/SDK.
3	Error during resource load. The player/SDK was not able to fetch the verification resource, or some error occurred that the player/SDK was able to detect. *Examples of detectable errors*: malformed resource URLs, 404 or other failed response codes, request time out. *Examples of potentially undetectable errors*: parsing or runtime errors in the JS resource.

Player Support	Required
Required in Response	No
Parent	TrackingEvents under Verification elements
Bounded	0+
Content	A URI to the tracking resource for the event specified using the event attribute.
Attributes	Description
event*	A string that defines the event being tracked. Accepted values are listed in section 3.17.3

*required

3.17.5 VerificationParameters

TOC Schema

`<VerificationParameters>` contains a CDATA-wrapped string intended for bootstrapping the verification code and providing metadata about the current impression. The format of the string is up to the individual vendor and should be passed along verbatim.

Player Support	Required
Required in Response	No
Parent	Verification
Bounded	0-1
Content	CDATA-wrapped metadata string for the verification executable.

3.18 Extensions

TOC Schema

Ad servers can use this XML node for custom extensions of VAST. When used, custom XML should fall under the nested `<Extension>` (singular) element so that custom XML can be separated from VAST elements. An XML namespace (xmlns) should also be used for the custom extension to separate it from VAST components.

The following example includes a custom XML element within the `<Extensions>` element.

```
<Extensions>
    <Extension>
    <CustomXML>…</CustomXML>
    <Extension>
</Extensions>
```

The publisher must be aware of and be capable of executing any VAST extensions in order to process the content.

Player Support	Optional
Required in Response	No
Parent	InLine or Wrapper
Bounded	0-1
Sub-elements	Extension

3.18.1 Extension

TOC Schema

One instance of `<Extension>` should be used for each custom extension. The `type` attribute is a custom value which identifies the extension.

Player Support	Optional
Required in Response	No
Parent	Extensions
Bounded	0+
Content	Custom XML object
Attributes	Description
type	A string that identifies the type of extension.

3.19 Wrapper

TOC Schema

VAST Wrappers are used to redirect the media player to another server for either an additional `<Wrapper>` or the VAST `<InLine>` ad. In addition to the URI that points to another file, the Wrapper may contain tracking elements that provide tracking for the InLine ad that is served following one or more wrappers. A Wrapper may also contain `<Companion>`

creative and <Icon> creative. And while <Linear> and <NonLinear> elements are available in the Wrapper, they are only used for tracking. No media files are provided for Linear elements, nor are resource files provided for NonLinear elements. Other elements offered for InLine ads may not be offered for Wrappers.

To find out if an element is offered for Wrappers, check the human-readable schema in section 5.

Player Support	Required
Required in Response	One of either InLine or Wrapper required but both are not allowed
Parent	Ad
Bounded	0-1
Sub-elements	Impression* VASTAdTagURI* AdSystem Pricing Error ViewableImpression AdVerifications Extensions Creatives BlockedAdCategories
Attributes	Description
followAdditionalWrappers	a Boolean value that identifies whether subsequent Wrappers after a requested VAST response is allowed. If false, any Wrappers received (i.e. not an Inline VAST response) should be ignored. Otherwise, VAST Wrappers received should be accepted (default value is "true.")
allowMultipleAds	a Boolean value that identifies whether multiple ads are allowed in the requested VAST response. If true, both Pods and stand-alone ads are allowed. If false, only the first stand-alone Ad (with no sequence values) in the requested VAST response is allowed. Default value is "false."
Attributes	Description
fallbackOnNoAd	a Boolean value that provides instruction for using an available Ad when the requested VAST response returns no ads. If true, the media player should select from any stand-alone ads available. If false and the Wrapper represents an Ad in a Pod, the media player should move on to the next Ad in a Pod; otherwise, the media player can follow through at its own discretion where no-ad responses are concerned.

* required

3.19.1 VASTAdTagURI

TOC Schema

While VAST Wrappers don't provide all the same elements offered for an InLine ad, the <VASTAdTagURI> is the only element that is unique to Wrappers. The VASTAdTagURI is used to provide a URI to a secondary VAST response. This secondary response may be another Wrapper, but eventually a VAST wrapper must return an <InLine> ad. In VAST 4 the player is only required to accept five wrappers ads. If no InLine ads are returned after 5 Wrappers, the player may move on to the next option.

Player Support	Required
Required in Response	Yes (if <Wrapper> is used)
Parent	Wrapper

Bounded	1 (if <Wrapper> is used)
Content	A URI to a VAST response that may be another VAST Wrapper or a VAST InLine ad. The number of VAST wrappers should not exceed 5 before an InLine ad is served. After 5 VAST wrapper responses, acceptance of additional VAST responses is at the publisher's discretion.

3.19.2 BlockedAdCategories

TOC Schema

Ad categories are used in creative separation and for compliance in certain programs. In a wrapper, this field defines ad categories that cannot be returned by a downstream ad server. This value is used to populate the [BLOCKEDADCATEGORIES] request macro in VASTAdTagURI strings, and can also be used by the player to reject InLine ads with Category fields that violate the BlockedAdCategories fields of upstream wrappers (see section 3.4.5). If an InLine ad is skipped due to a category violation, the client must notify the ad server using the `<Error>` URI, if provided (error code 205), and move on to the next option.

Player Support	Optional
Required in Response	No*
Parent	Wrapper
Bounded	0+
Content	A string that provides a comma separated list of category codes or labels per authority that identify the ad content.
Attributes	Description
authority *	A URL for the organizational authority that produced the list being used to identify ad content.

*Optional unless the publisher requires ad categories. The `authority` attribute is required if categories are provided.

4 Migration to VAST 4.x

VAST 4 offers features to support long-form video, server-side tracking, industry-wide creative tracking, and viewability and verification tracking. While the advance in features is alluring, media players will need time to upgrade their systems. During the transition period from VAST 3.0 to 4 (or 2.0 to 4), prepare to manage varying feature support in the market. VAST 4 was designed to be backward compatible with version 3.0 and VAST 3.0 was designed to be backwards compatible with version 2.0.

However, features introduced in the newer versions will typically not be back ported to older-versioned players. Also, features explicitly called out as deprecated or removed will break backward compatibility.

The following sections outline a few notes to consider as VAST 4 is introduced into the market.

4.1 Advertisers and Ad Technology Vendors

Design ads that can be successfully delivered to lower versioned VAST players while still optimizing the response with new 4 capabilities. For example:

- VAST 4 ads discourage the use of VPAID or other interactive ad units that require an API to execute in the <MediaFile>. The new <InteractiveCreativeFile> was provided to accommodate such ads. However, in older versions, an interactive unit may be provided in addition to the video <MediaFile> in order to ensure interactive files are executed where possible in older VAST version players.
- In a 4 response, use both the Creative adId attribute as well as the new <UniversalAdId> element to provide a creative ad ID.

4.2 Ad Servers and Networks

Be prepared to manage the variability with VAST versions. For example, if a player specifically requests a VAST 3.0 response, then the ad server should limit responses to VAST 3.0.

If one or more verification vendors are involved, use VAST 4 to provide verification code in the new <AdVerification> node, but expect that older versioned players will not recognize the verification node.

An important change discussed during 4.1 is the concept of standardized ad requests using AdCOM and POST requests. This is something that likely will require a phased approach on ad servers and so should be planned accordingly. The group recommends that servers start supporting POST requests (in addition to GET requests) in the near future and look into related scaling issues, because the AdCOM based ad request support will be developed next.

4.3 Media Players

VAST 4 players should continue to accept ads on older versions of VAST because it will take time for the entire industry to upgrade.

5 Human Readable VAST XML Schema

The following schema models the structure for VAST along with available attributes. Click the section number for more detail.

Element	Attributes	Required	Section
VAST	version	Yes	3.2
/Error		No	3.2.1
VAST/Ad	id, sequence, conditionalAd, adType	Yes	3.3
VAST/Ad/InLine		Yes*	3.4
/AdSystem	version	Yes	3.4.1
/AdTitle		Yes	3.4.2
/Impression	id	Yes	3.4.4
/AdServingId		Yes	3.4.3
/Category	authority	No	3.4.5
/Description		No	3.4.6
/Advertiser	id	No	3.4.7
/Pricing	model, currency	No	3.4.8
/Survey	type	No	3.49 8
/Error		No	3.4.11
/Expires			3.4.10
/ViewableImpression	id	No	3.5
/Viewable		No	3.5.1
/NotViewable		No	3.5.2
/ViewUndetermined		No	3.5.3
/AdVerifications		No	3.16
/Verification	vendor	No	3.17
/JavaScriptResource	apiFramework, browserOptional	No	3.17.1
/ExecutableResource	apiFramework, language	No	3.17.2
/TrackingEvents		No	3.17.3
/Tracking	event		
/VerificationParameters			
/Extensions		No	3.18
/Extension	type	Yes	3.18.1
/Creatives		Yes	3.6
/Creative	id, sequence, adId, apiFramework	Yes	3.7
/UniversalAdId	idRegistry	Yes	3.7.1
/CreativeExtensions		No	3.7.2
/CreativeExtension	type		3.7.3
/Linear	skipoffset	Yes (linear)	3.8
/Duration		Yes	3.8.1
/AdParameters	xmlEncoded	No	3.8.2
/MediaFiles		Yes	3.9
/Mezzanine	delivery, type, width, height, codec, id, fileSize, mediaType	Yes (ad-stitching)	3.9.2

Element	Attributes	Required	Section
/MediaFile	id, delivery, type, bitrate, minBitrate, maxBitrate, width, height, scalable, mantainAspectRatio, codec, apiFramework, fileSize, mediaType	Yes	3.9.1
/InteractiveCreativeFile	type, apiFramework, variableDuration	No	3.9.3
/ClosedCaptionFiles			
/ClosedCaptionFile	type, language		
/VideoClicks		No	3.10
/ClickThrough	id	No	3.10.1
/ClickTracking	id	No	3.10.2
/CustomClick	id	No	3.10.3
/TrackingEvents		No	3.14
/Tracking	event, offset	No	3.14.3
/Icons		No	3.11
/Icon	program, width, height, xPosition, yPosition duration, offset, apiFramework, pxratio	Yes	3.11.1
/StaticResource /IFrameResource /HTMLResource	creativeType (StaticResource only)	Yes	3.15.1
/IconClicks		No	3.11.3
/IconClickThrough		No	3.11.4
/IconClickTracking	id	No	3.11.5
/IconClickFallbackImages		No	
/IconClickFallbackImage	AltText,StaticResource		
/IconViewTracking		No	3.11.2
/NonLinearAds		Yes (NonLinear ads)	3.12
/NonLinear	id, width, height, expandedWidth, No expandedHeight, scalable, maintainAspectRatio, minSuggestedDuration, apiFramework		3.12.1
/NonLinearClickThrough			3.12.2
/NonLinearClickTracking			3.12.3
/TrackingEvents		No	3.14
/Tracking	event	No	3.14.3
/CompanionAds	required	No	3.13
/Companion	id, width, height, assetWidth, assetHeight,	No	3.13.1

	expandedWidth, expandedHeight, apiFramework, adSlotId, pxratio, renderingMode		
/StaticResource /IFrameResource /HTMLResource	creativeType (StaticResource only)	Yes	3.15.1
/AdParameters	xmlEncoded	No	3.8.2
/AltText		No	3.13.2
/CompanionClickThrough		No	3.13.3
/CompanionClickTracking	id	No	3.13.4
/TrackingEvents		No	3.14
/Tracking	event	No	3.14.3

Element	Attributes	Required	Section
VAST/Ad/Wrapper	followAdditionalWrappers, allowMultipleAds, fallbackOnNoAd	No*	3.19
/Impression	id	Yes	3.4.3
/VASTAdTagURI		Yes	3.19.1
/AdSystem	version	Yes	3.4.1
/Pricing	model, currency	No	3.4.7
/Error		No	3.5
/ViewableImpression	id	No	3.5
/Viewable		No	3.5.1
/NotViewable		No	3.5.2
/ViewUndetermined		No	3.5.3
/AdVerifications		No	3.16
/Verification	vendor	No	3.17
/JavaScriptResource	apiFramework, browserOptional	No	3.17.1
/ExecutableResource	apiFramework,language	No	3.17.2
/TrackingEvents		No	3.17.3
/Tracking	event		
/VerificationParameters			
/BlockedAdCategories	authority	No	3.19.2
/Extensions		No	3.18
/Extension	type	No	3.18.1
/Creatives		No	3.6
/Creative	id, sequence, adId	No	3.7
/Linear		Yes	3.8
/TrackingEvents		No	3.14
/Tracking	event, offset	No	3.14.3
/VideoClicks		No	3.10
/ClickTracking	id	No	3.10.2
/CustomClick	id	No	3.10.3
/ClickThrough	id	No	
/Icons		No	3.11

/Icon	program, width, height, xPosition, yPosition duration, offset, apiFramework, pxratio	Yes	3.11.1
/StaticResource /IFrameResource /HTMLResource	creativeType (StaticResource only)	No	3.15
/IconClicks		No	3.11.3
/IconClickThrough		No	3.11.4
/IconClickTracking	id	No	3.11.5
/IconClickFallbackImages			
/IconClickFallbackImage	AltText,StaticResource		
/IconViewTracking		No	3.11.2
/InteractiveCreativeFile	type, apiFramework, variableDuration		3.9.3
/NonLinearAds		Yes	3.12
/NonLinear			3.12.1
/NonLinearClickTracking			3.12.2
/TrackingEvents		No	3.14
/Tracking	event	No	3.14.3

Element	Attributes	Required	Section
/CompanionAds	required	No	3.13
/Companion	id, width, height, assetWidth, assetHeight, expandedWidth, expandedHeight, apiFramework, adSlotId, pxratio, renderingMode	No	3.13.1
/StaticResource /IFrameResource /HTMLResource	creativeType (StaticResource only)	No	3.15
/AdParameters	xmlEncoded	No	3.8.2
/AltText		No	3.13.2
/CompanionClickThrough		No	3.13.3
/CompanionClickTracking	id	No	3.13.4
/TrackingEvents		No	3.14
/Tracking	event	No	3.14.3

*Either the InLine element or the Wrapper element is required and only one is allowed.

6 Macros

6.1 Introduction

Ad servers and other entities need access to additional data from the publisher to meet client needs for a clearer view into the details of how and where their video is being shown.

The following macros enable the media player to provide these additional data points. Some may need to be relayed from the publisher ad server to the player in turn before the player can pass them on.

The following overview outlines the various macros, in which contexts they are applicable and their meaning.

Macro Formatting and Replacement

All macro names are surrounded by square brackets, for example: `[EXAMPLE]`. When replacing the macro with a value, the whole name - including brackets - needs to be replaced with the value.

For example, if you'd want to replace the `[EXAMPLE]` macro in the URL `https://mydomain.com/something?test=[EXAMPLE]` with the value `somevalue`, you would get `https://mydomain.com/something?test=somevalue`

Macro Replacement Responsibility

The responsibility to properly replace macros with their proper values lies with the party that will perform the HTTP request.

In the most common scenario, both for VAST URLs and tracking pixel URLs, this would be the video player that's executing the ad.

In some cases a server might perform the macro replacement on behalf of the video player, for example in the case of server-side ad insertion where the server is performing tracking pixel requests on behalf of the client.

Marking Macro Values as Unknown or Unavailable

For any macros that are marked as optional or deprecated and where the actual macro is not provided, the following special values must be inserted into the macro to indicate the reason for not providing the information:

If the macro value is...	Then replace macro with...
Value is unknown, but would be shared if it was known	`-1`
Value is known, but information can't be shared because of policy (unwilling to share)	`-2`

Implementation Note: do not replace all unknown macros with `-1`, only do this for macros specifically mentioned in this section that you decide not to implement.

Macro Value URI Encoding

Some macros must be populated as a series of values rather than a single value. These macros use the `Array<T>` data type. This is a list of `T` values where `T` is another data type like `string` or `integer`. When replacing a macro with such a list, the value should be rendered as a set of values separated by a comma (","), and no spacing. For example: the values `stringA` and `stringB` would be encoded as `stringA,stringB`.

When replacing macros, make sure to apply `encodeURIComponent` to any value, to avoid creating invalid URLs. However, note that encoding should be applied to individual values only, not the entire macro replacement string (i.e. unencoded commas should separate distinct values).

For example, to encode the values `abc/def` and `y=z`, you'd replace the macro with `abc%2Fdef,y%3Dz`

Note that each individual value is properly encoded, but the comma between values is not.

In the examples given below, if the URI-encoded version differs from the unencoded original, both are given for the sake of clarity. However, **the encoded version must always be used as a macro substitution.**

Note: In order to ensure that new macros can be added without requiring a new VAST version at every change, we are maintaining the latest list on the IAB Tech Lab's VAST github repository. Please visit http://interactiveadvertisingbureau.github.io/vast/vast4macros/vast4-macros-latest.html for the latest list and for more details on the process of adding new macros.

6.2 List of Macros

Please reference the Macro list in GitHub for the most up to date macros.

https://interactiveadvertisingbureau.github.io/vast/vast4macros/vast4-macros-latest.html

7 VAST Terminology

As the video and audio advertising industry has evolved, certain terminology has gained widespread adoption. The following definitions represent some of that terminology as it relates to video and audio ad serving discussed in this document.

Ad Pod: An Ad Pod is sequence of Linear ads played back-to-back, like a commercial break with multiple ad spots on TV.

Companion Ad: Commonly a display banner or rich media ad that appears on the page outside of the media player. Companion Ads may remain on the page after the related in-stream ad ends. A Companion Ad can also be a skin that wraps the video or audio experience.

Clickthrough: A URL for page that opens when a user clicks the ad creative.

InLine Ad: A VAST ad response that contains all the information needed to play the video or audio ad. No additional calls to other ad servers are needed after a VAST InLine ad response is received.

In-Stream Ad: Any ad that appears inside a streaming media player, whether it's an image overlay or a Linear video or audio ad, such as an ad that plays in a 30 second ad spot.

Linear Ad: Linear Ads are like TV or Radio commercials and can appear before the content video/audio plays (pre-roll), during a break in the content video/audio (mid-roll), or after the content video/audio ends(post-roll). Linear ads may be video/audio, rich media or still image ads. Using an API or other technology, Linear ads can be interactive and ad duration can be extended when a user interacts.

Master Ad: For video or audio ad campaigns that include an in-stream ad plus one or more Companion ads, the in-stream portion of the ad unit is referred to as the master ad. In this master-companion relationship, the master ad must always be shown.

NonLinear Ad: An in-stream ad that appears concurrently with the video or audio content playback. NonLinear ads usually cover the bottom or top fifth of the media player and can be text, image or interactive ads. Using an API or other technology, the media player may allow user-initiated interaction in a NonLinear ad to stop content video/audio playback. NonLinear ads can only appear at some point between content video/audio start and end (mid-roll positions) and generally disappear after 10-20 seconds if there is no interaction. [Note: *NonLinear ads have failed to achieve scale in the market.*

Overlay Ad: A NonLinear ad format in which an image or text displays on top of video content. Overlay ads are commonly referred to as simply "NonLinear Ads;" however NonLinear Ads may also include non-overlay formats that are served within the media player but without covering any video content.

Primary Ad Server: The first ad server that the media player calls to for ad content. The primary ad server is usually the ad server used by the publisher.

Secondary Ad Server: The ad server that the media player calls after receiving a VAST redirect (Wrapper) from the primary ad server. Secondary ad servers may include agency or ad network ad servers. Also, secondary ad servers may redirect the media player to a third ad server and the third ad server may redirect to a fourth, and so on. Eventually, an ad server must provide a VAST response that includes all the creative elements needed to display the ad.

VAMG: Video Ad Measurement Guidelines is an IAB guideline that defines the set of events that should be tracked when a video ad is played.

VAST: The Video Ad Serving Template is an IAB guideline and XML schema that describes the XML structure for a video or audio ad response. VAST enables ad responses to come from any ad server.

VAST Redirect: A VAST ad response that points to another VAST response (sometimes referred to as the downstream VAST response).

VAST Tag: A URI that returns a VAST response when called.

Video Ad: Any ad displayed in the context of a video experience. A video experience may include in-banner video, in-text video, in-stream video and other formats. VAST applies only to in-stream video where a media player is used to manage the video experience independent of any other content. For example, video served within an ad banner is considered rich media and is NOT addressed in the VAST guideline.

Media Player: A media playback environment used to manage a video or audio experience. Media players are provided by an Online Video Platform (OVP) vendor or can be custom-built by the publisher.

VMAP: Video Multi Ads Playlist is an IAB guideline that describes the XML structure for a playlist of video ads sent from an ad server to a media player.

VPAID: Video Player Ad Interface Definition is an IAB guideline that defines the communication protocols between an interactive ad and the media player that is rendering it.

Wrapper: in the context of VAST, a Wrapper is a response that provides a URI that the media player uses to call a secondary VAST response. The secondary response may be either another Wrapper or a VAST InLine response.

8 Corrections & Clarifications

The following corrections were made to the VAST 4.2 doc. These did not change the actual technical spec or XSD, and so did not result in an update to the version.

1. Fixed HTMLResource description & example back to original intent (from VAST 2/3)
    a. Changes - Section 3.15 & 3.15.3
2. Fixed IconClickFallbackImage(s) references
    a. Changes - section 3.11.3, 3.11.6.1
3. Fixes to “human readable” list (section 7)
4. Minor Language/grammar fixes throughout the doc
5. Added reference to “latest” macros list in section 6