Server Side Unwrapped VAST

**Authors**: [mpungaru@google.com](http://who/mpungaru)

**Last updated: 2020-06-25**

# Objective

Propose a change to the VAST (4.3+) standard to support server side unwrapping use cases in a native way by introducing a **<VASTAdData>** element in the <Wrapper> element. The new element would be present alongside <VASTAdTagURI> which would represent the URL that was called.

# Background

The Video Ad Serving Template ([VAST](https://iabtechlab.com/standards/vast/)) standard offers an XML dialect which can describe the video ads a player should render for the user. The VAST document contains instructions about the video ad creative that should be rendered, the URLs which should be pinged when various events happen with the ad creative (an error occurs, a view occurs, the user pauses the video, etc.) as well as other bits of information facilitating the communication between the ad server and the player.

The basic building block of a VAST document is the <Ad> element. An <Ad> element represents in the system that generated the response a single advertisement. Inside the <Ad> element, the advertisement can be described in one of two ways:

* **<InLine>**. In this case the actual assets that need to be played are known to the ad server generating the response and their location is copied in this element in the VAST document.
* **<Wrapper>**. In this case the assets that need to be played are not known to the ad server. Instead, the ad server returns a redirect to another ad server (inside the element <VASTAdTagURI>) alongside with URLs which would need to be pinged for various accounting purposes (views, errors, etc.). The video player would follow the redirect to get to the video ad assets and play them. The video player could go through several redirects (hops) before getting to the video assets. When the video assets are played the view URLs are pinged for all the ad servers through which the ad request has been processed.

The <Wrapper> element is very popular currently, a large portion of video ad requests going through at least one hop like this. Unfortunately, by going through these extra hops we run a higher likelihood of error (the error rates of all the ad servers involved in ad serving are compounded) and we increase the end user latency (the bandwidth for each user varies wildly).

In order to reduce the number of requests ending up in error and to maximize the opportunity, some ad servers have started calling the <Wrapper> stored redirect tags server-side and returning the final response as an <InLine> ad. The ad server will combine its own view events, quartile events, etc. with those belonging to the called ad servers into one unified response.

Though this model works reasonably well, if the player and one of the ad servers in the chain of ad calls have a special integration the data they use to communicate between them (most likely an extension) can’t be properly handled and runs the risk of being dropped. This lossy mechanism hinders innovation by eliminating the communication channel between ad servers in the chain and players.

# Overview

We will introduce a new element inside the <Wrapper> element -- **<VASTAdData>**. The <VASTAdData> element would contain the full VAST document that has been retrieved from another ad server. The <VASTAdData> element would live alongside the <VASTAdTagURI> element and serve as a cached response for the player.

# Detailed design

For the cases when the ad server has a redirect and it does server side unwinding of the ad tag, the <VASTAdData> element could be used to represent the VAST fetched from the other ad server. The <VASTAdData> element would contain a fully formed VAST document.

When the <VASTAdData> element is present in the <Wrapper> element, the <VASTAdTagURI> will contain the exact URL that was used to call the partner ad server.

The VAST document inside the <VASTAdData> element must be a valid VAST document. To ensure this happens, the ad servers might have to rewrite the VAST document. Here are some cases in which the ad server would have to rewrite the received VAST document:

* The retrieved VAST document still contains redirects and the ad server wants to follow those to get to the final assets. This would lead to having an <VASTAdData> inside an <VASTAdData>. There is no requirement for an <VASTAdData> to fully resolve all <Wrapper> elements, those could still be present if the ad server decides to let the player follow some of the redirects.
* One of the redirects in the received VAST document fails retrieval and the ad server replaces it with an ad from the ad buffet.
* We have a VAST redirect which ends up in error inside the pod. The ad server would need to rewrite the sequence indices.

# Example

Let’s assume we have the following ad calls and responses that a video player would do.

ad-server1.com → returns 2 redirects (to ad-server2.com and ad-server3.com)

|  |
| --- |
| <VAST>  <Ad id="1" sequence="1">  <Wrapper>  <AdSystem>Ad Server 1</AdSystem>  <VASTAdTagURI><![CDATA[ https://ad-server2.com ]]></VASTAdTagURI>  <Impression><![CDATA[ https://ad-server1.com/impression-ad-1 ]]></Impression>  </Wrapper>  </Ad>  <Ad id="2" sequence="2">  <Wrapper>  <AdSystem>Ad Server 1</AdSystem>  <VASTAdTagURI><![CDATA[ https://ad-server3.com ]]></VASTAdTagURI>  <Impression><![CDATA[ https://ad-server1.com/impression-ad-2 ]]></Impression>  </Wrapper>  </Ad>  </VAST> |

ad-server2.com → returns an inline ad

|  |
| --- |
| <VAST>  <Ad id="1" sequence="1">  <InLine>  <AdSystem>Ad Server 2</AdSystem>  <Impression><![CDATA[ https://ad-server2.com/impression-ad-1 ]]></Impression>  <Creatives>...</Creatives>  </InLine>  </Ad>  </VAST> |

ad-server3.com → returns one inline ad, a redirect (which will error out) and another inline ad.

|  |
| --- |
| <VAST>  <Ad id="1" sequence="1">  <InLine>  <AdSystem>Ad Server 3</AdSystem>  <Impression><![CDATA[ https://ad-server3.com/impression-ad-1 ]]></Impression>  <Creatives>...</Creatives>  </InLine>  </Ad>  <Ad id="2" sequence="2">  <Wrapper>  <AdSystem>Ad Server 3</AdSystem>  <VASTAdTagURI><![CDATA[ https://ad-server-with-error.com ]]></VASTAdTagURI>  <Impression><![CDATA[ https://ad-server3.com/impression-ad-2 ]]></Impression>  </Wrapper>  </Ad>  <Ad id="3" sequence="3">  <InLine>  <AdSystem>Ad Server 3</AdSystem>  <Impression><![CDATA[ https://ad-server3.com/impression-ad-3 ]]></Impression>  <Creatives>...</Creatives>  </InLine>  </Ad>  </VAST> |

Here is the expected outcome if Ad Server 1 does full server side unwinding.

|  |
| --- |
| <VAST>  <Ad id="1" sequence="1">  <Wrapper>  <AdSystem>Ad Server 1</AdSystem>  <VASTAdTagURI><![CDATA[ https://ad-server2.com ]]></VASTAdTagURI>  **<VASTAdData>**  <VAST>  <Ad id="1" sequence="1">  <InLine>  <AdSystem>Ad Server 2</AdSystem>  <Impression><![CDATA[ https://ad-server2.com/impression-ad-1 ]]></Impression>  <Creatives>...</Creatives>  </InLine>  </Ad>  </VAST>  **</VASTAdData>**  <Impression><![CDATA[ https://ad-server1.com/impression-ad-1 ]]></Impression>  </Wrapper>  </Ad>  <Ad id="2" sequence="2">  <Wrapper>  <AdSystem>Ad Server 1</AdSystem>  <VASTAdTagURI><![CDATA[ https://ad-server3.com ]]></VASTAdTagURI>  **<VASTAdData>**  <VAST>  <Ad id="1" sequence="1">  <InLine>  <AdSystem>Ad Server 3</AdSystem>  <Impression><![CDATA[ https://ad-server3.com/impression-ad-1 ]]></Impression>  <Creatives>...</Creatives>  </InLine>  </Ad>  <Ad id="3" sequence="2">  <InLine>  <AdSystem>Ad Server 3</AdSystem>  <Impression><![CDATA[ https://ad-server3.com/impression-ad-3 ]]></Impression>  <Creatives>...</Creatives>  </InLine>  </Ad>  </VAST>  **</VASTAdData>**  <Impression><![CDATA[ https://ad-server1.com/impression-ad-2 ]]></Impression>  </Wrapper>  </Ad>  </VAST> |

You can see in the response above that the VAST document from ad-server2.com has been copied verbatim while from the response from ad-server3.com the second ad has been removed and the sequence indices have been updated.