Prebid Auction, Win, and Impression Events

Updated Apr 17, 2025

Overview

This document details the various scenarios where Demand Manager logs analytics events.

There are several different scenarios summarized in this table:

Environment	Format	Bid Source	Auctions Events	Bid Won Event	Impression Event
Web	Banner	Client-side	PBJS analytics adapter	PBJS analytics adapter	n/a
Web	Banner	Server-side	PBJS analytics adapter	PBJS analytics adapter	n/a
Web	Banner	Server-side	PBS analytics	PBS returns a win url containing a partial event, which is fired	n/a

				by Prebid.js	
Web	Instream Video	Server-side	PBJS analytics adapter	n/a	PBS can be set up to inject imp 'partial events' into the VAST which are fired by the video player. An analytics pipeline can join these events to the original bid request to get all fields.
Web	Instream Video	Server-side	PBS analytics	n/a	PBS can be set up to inject 'partial events' into the VAST which are fired by the player.
Web	Instream Video	Client-side	PBJS analytics adapter	n/a	PBJS event handlers can be used to insert impression tracking into VAST. Or modules can use the registerVastTrackers library.
Web	Outstream video	Client-side	PBJS analytics adapter	PBJS analytics adapter	PBJS event handlers can be used to insert impression tracking into VAST. Or modules can use the registerVastTrackers library.
App	Video	Server-side	PBS analytics	n/a	PBS can be set up to inject imp 'partial events' into the VAST which are fired by the video player.
App	Banner	Server-side	PBS analytics	PBS puts a 'wurl' partial event into PBC, which is fired by the PUC.	n/a
AMP	Banner	Server-side	PBS analytics	PBS puts a 'wurl' partial event into PBC, which is fired by the PUC.	n/a

Glossary

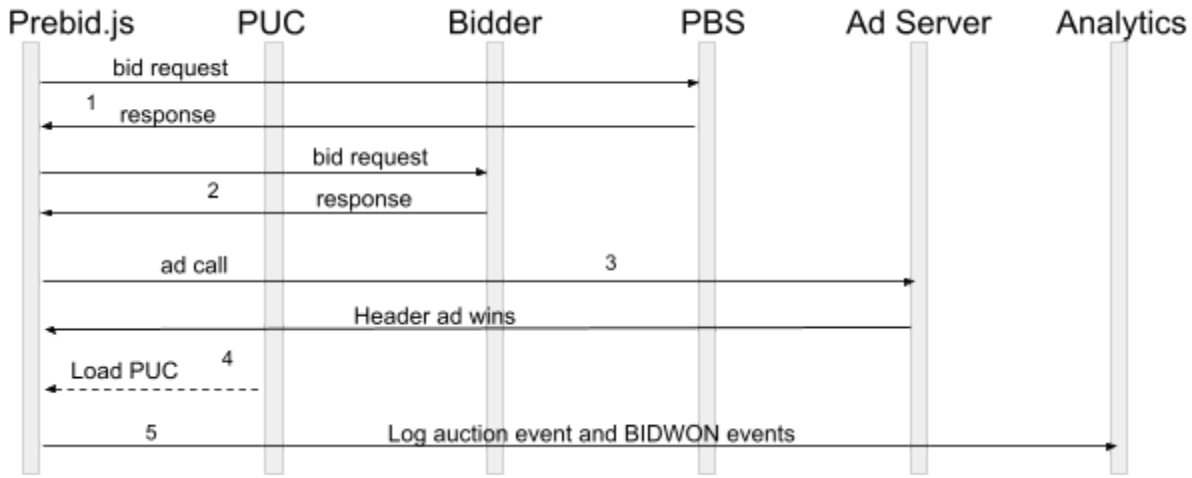
These are the terms that appear in the following handshake diagrams

- Prebid.js - the PBJS wrapper should include an analytics adapter
- PUC - the Prebid Universal Creative is scheduled in the ad server and coordinates the rendering of the winning creative.
- Bidder - a SSP or other demand source
- Player - a video player coordinates video content and ads
- PBS - Prebid Server calls out to bidders within the cloud
- PBC - Prebid Cache stores creatives that cannot be stored on the client device
- Ad Server - the system that manages the publishers ad campaigns
- App - a mobile application integrated with Prebid SDK that calls out to Prebid Server
- AMP - the Google Accelerated Mobile Pages system that caches a fast version of web pages for mobile devices
- Auction Event - analytics adapters are assumed to log an event for the auction as a whole and/or the bidResponse(s) from each bidder.
- Win Event - generated when the ad server has chosen an ad to be displayed
- Impression Event - generated only for video ads when the player is just about to start the ad video.

Desktop and Mobile Web Banner

Web Banner with Client-Side Analytics

In the most common scenario for events, Prebid.js is available to log both the auction event and the bids won event:



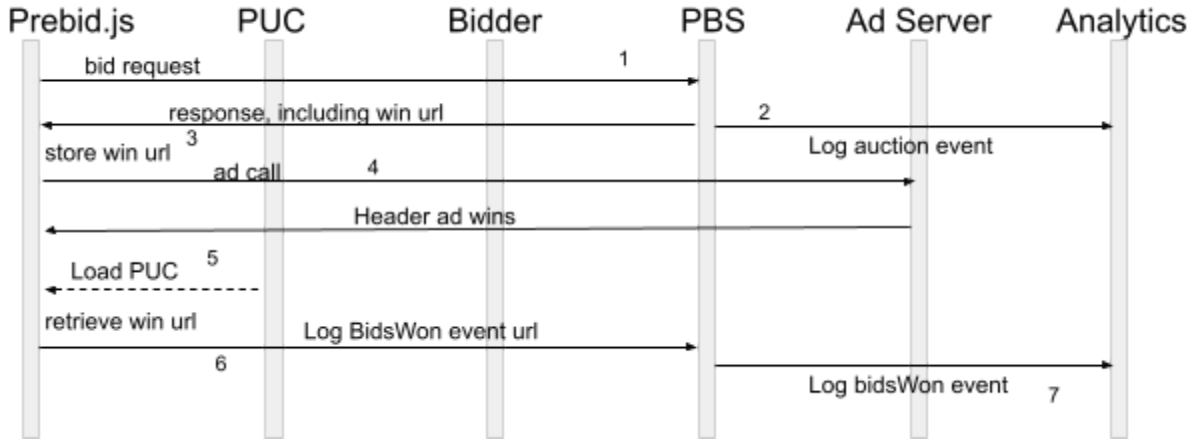
The process:

1. Bid requests go to Prebid Server which responds as normal. Prebid Server doesn't log any analytics, trusting that client-side analytics will handle everything. The PrebidServerBidAdapter creates a bidResponse for each AdUnit/bidder combination including a bidId supplied by Prebid Server.
2. Bid requests go directly to bidder endpoints, and Prebid.js collects the responses.
3. The ad call is made with one or more bids attached.
4. When header bidding wins, an ad creative or Publisher code initiates the rendering function.
    a. All of these things invoke rendering:
        i. Prebid Universal Creative
        ii. Prebid Dynamic Creative
        iii. Page calls renderAd()
    b. The rendering function initiates the handling of the bid-won and imp events:
        i. Fires the bid-won event for analytics adapters
        ii. Handle the Prebid EventTrackers array:
            1. Drop pixels for bid-won trackers
            2. Check the adunit's 'deferBilling' flag
                a. If billing is *not* deferred for the adunit, then count the impression: drop pixels for the imp Prebid EventTrackers and call bidder-specific onBidBillable() functions
        iii. If it's a native ad and there are native tracker events, drop pixels for Native imp tracker events.
    c. If billing is 'deferred' for the adunit, then the publisher (or the bidViewability module) will call the "triggerBilling" function. This function drops pixels for the imp Prebid Event Trackers and calls any bidder-specific onBidBillable method.
    d. Native clicks are handled by both the PUC and the dynamic creative but with a different function.
5. The analytics Adapter should try to minimize the number of JSON packages sent to the analytics endpoint in order to lighten the footprint on the user's device.

a. Any BidWon event that happens after the Auction events are logged in their own message.

Web Banner via Prebid Server with Server-Side Analytics

Another scenario is when there's no client-side analytics, so Prebid Server has to be responsible for logging the auction events and needs to know about the bidsWon events. It is only able to log events within its own scope -- i.e. if there are client-side bidders, it will not see their auctions or bidsWon.



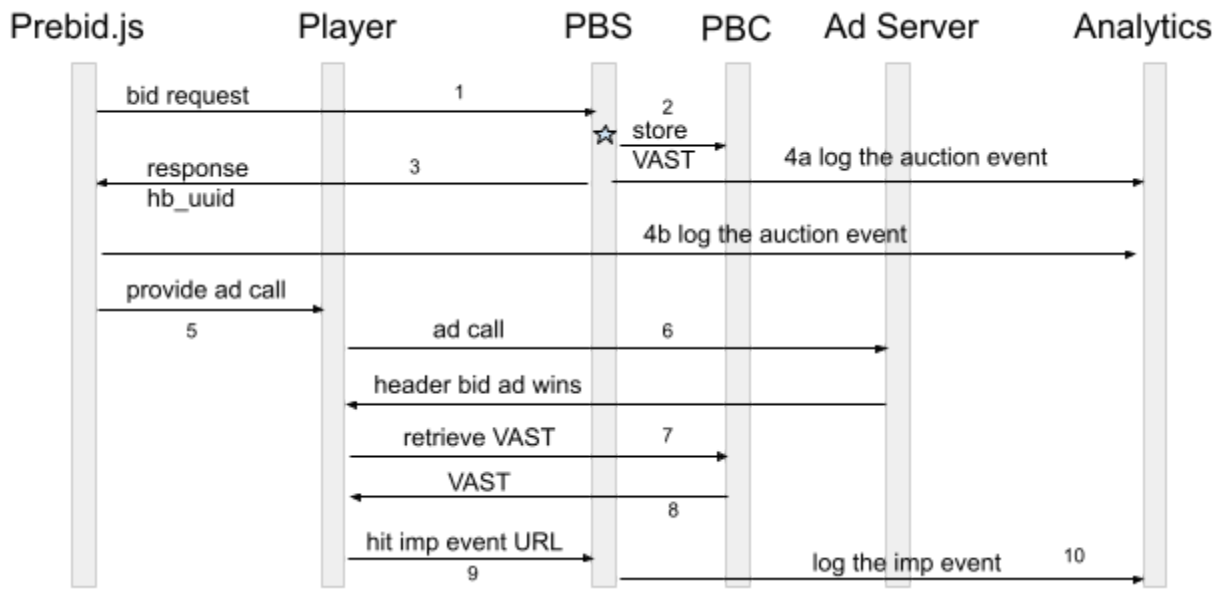
The process:

1. The bid request goes to Prebid Server which responds as normal, except that a win event URL for each bid may be provided.
2. Prebid Server logs the auction event to the analytics system right away.
3. Upon receiving the response, The Prebid.js PrebidServerBidAdapter stores any provided win url with the other bidresponse eventtrackers.
4. The ad call is made with one or more bids.
5. When header bidding wins, something calls the PBJS renderAd() function. This may be the Prebid Universal Creative, the 'Dynamic Creative", or something else.
6. The rendering function fires all the win eventtrackers, including the PBS win URLs added in step 3.
7. Prebid Server receives the event URL and makes it available to the server-side analytics adapter, which calls the analytics system.

Video

Web Server-Side Instream Video

In order to get video to register events into an analytics system, Prebid Server modifies the VAST XML to inject an additional impression event. This happens at the star in the diagram below.



1. Prebid.js calls Prebid Server for a video ad requesting server-side caching.
2. When a bidder responds with VAST XML, Prebid Server may modify the VAST XML, injecting an additional impression tag before storing in Prebid Cache Server
3. Prebid Server responds with the bid information, including the 'hb_uuid' targeting value which contains the cache ID.
4. The auction event will be logged by the server analytics or the client analytics.
5. Prebid.js provides the video player with the ad call.
6. The player makes the call to the ad server.
7. When header bidding wins, the player retrieves the VAST XML from Prebid Cache.
8. This is the VAST that was modified to have the additional impression tracking tag.
9. The video player hits the additional tracking URL which goes to Prebid Server.
10. The analytics adapter on Prebid Server calls the analytics endpoint.

Web Client-Side Instream Video with Client-Side Caching

Unlike server-side video, the VAST XML coming into the browser doesn't go through Prebid Server, so will not have the tracking strings added. Prebid.js supports configuration that allows the publisher to initiate "client-side caching". In order to modify the VAST XML, the page can configure an endpoint on Prebid Server that can perform the same impression tracking modifications as if it had come through the server in the first place.

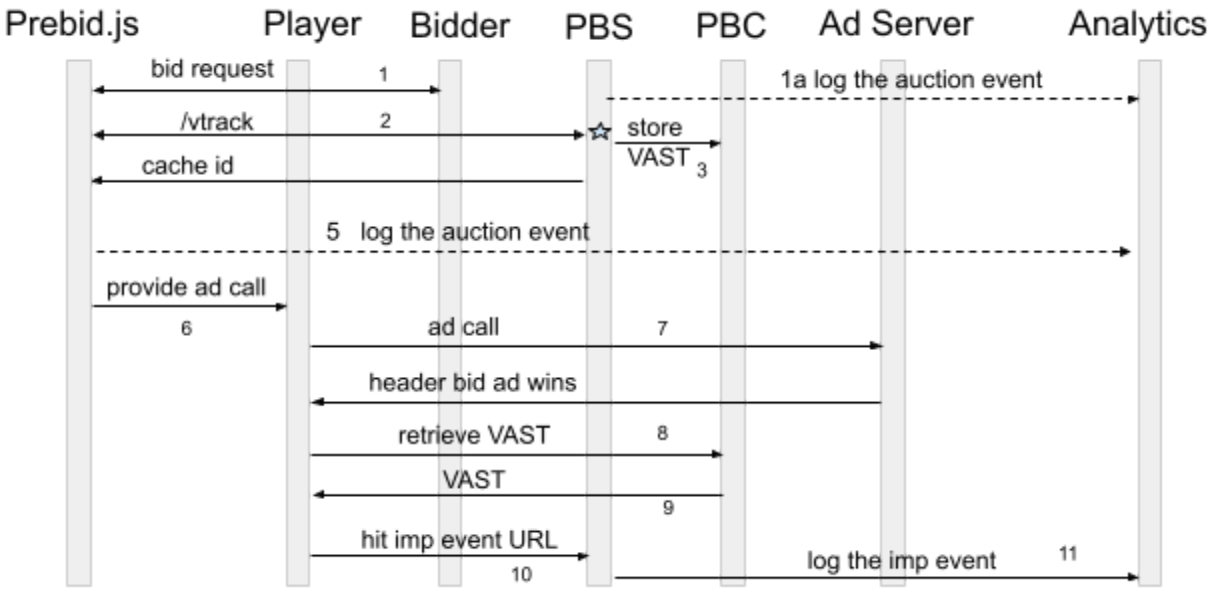
```
pbjs.setConfig({
    cache: {
        vasttrack: true,
        url: "https://PBS_HOST/vtrack?a=1001"
    }
});
```

Prebid.js POSTs the XML to the specified endpoint with this JSON:

```
{"puts":[{
    "bidid": BIDID,
    "bidder": "BIDDER",
    "timestamp": TIMESTAMP
    "type":"xml",
    "value":"<VAST…/VAST>",
    "ttlseconds":3600
}]}
```

The /vtrack PBS endpoint:

- parses the posted JSON and pulls out the bidid, bidders, and timestamp parameters
- use the parameters to create the additional <impression> object in the VAST XML
- POSTs the modified JSON to Prebid Cache, waits for the results from PBC, and forwards them to the client with the cache ID.

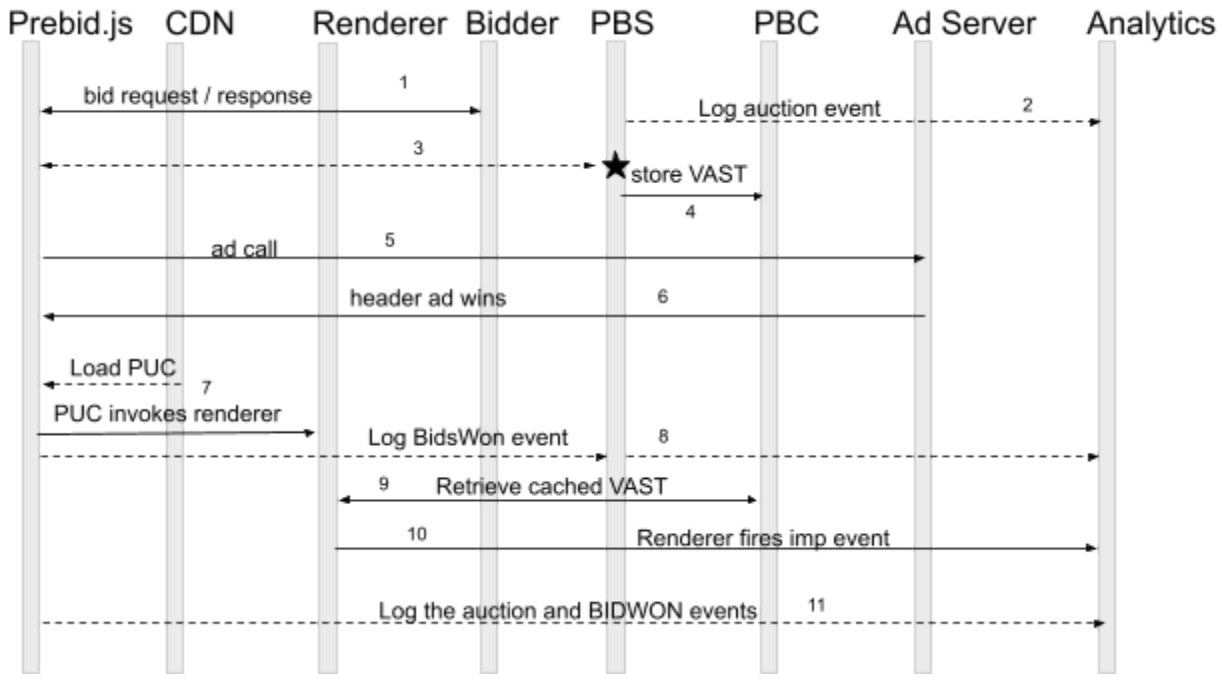


1. Prebid.js calls the bidder for a video ad resulting in VAST XML being returned to the device.
    a. If the call was to a Prebid Server with auction logging on, it will initiate the analytics call.
2. The page is configured to call Prebid Server's /vtrack endpoint to store that VAST server-side.
3. Prebid Server modifies the VAST XML, injecting an additional impression tag before storing in Prebid Cache Server
4. Prebid Server responds with the bid information, including the cache ID.
5. If the auction event wasn't logged by the server analytics then client analytics should do it.
6. Prebid.js provides the video player with the ad call.
7. The player makes the call to the ad server.
8. When header bidding wins, the player retrieves the VAST XML from Prebid Cache.
9. This is the VAST that was modified to have the additional impression tracking tag.
10. The video player hits the additional tracking URL which goes to Prebid Server.
11. The analytics adapter on Prebid Server calls the analytics endpoint.

Web Outstream Video

Outstream video functions as a hybrid of web banner and web instream video. Like banner, outstream uses the Prebid Universal Creative (PUC), which will trigger Prebid bidwon events when invoked. Like instream, outstream video creatives use VAST and are cached in Prebid Server, where an impression tracking URL is inserted into the VAST document.

In the scenario depicted below, both bidwon and impression events fire for the outstream ad. This scenario assumes that the publisher has configured Prebid to cache video creatives in Prebid Server and that special renderer(s) that are VAST-compliant. Most (but not all) analytics customers are expected to meet these requirements. For those who don't, we should expect to see a bidwon event but not an impression event.



1. Prebid.js calls the bidder for a video ad resulting in VAST XML being returned to the device.
2. If the call was to a Prebid Server with auction logging on, it will initiate the analytics call.
3. If the page is configured that way, call Prebid Server's /vtrack endpoint to store that VAST server-side. Prebid Server responds with the bid information, including the cache ID.
4. Prebid Server modifies the VAST XML, injecting an additional impression tag before storing in Prebid Cache Server.
5. Prebid.js makes the ad call call
6. When header bidding wins, the player retrieves the VAST XML from Prebid Cache.
7. The PBJS render function is invoked (either by the Prebid Universal Creative or the Dynamic Creative) and the outstream renderer is invoked.
8. If server-side analytics is in use, the BidsWon event is sent to Prebid Server which forwards it to the analytics endpoint.
9. The renderer loads the VAST that was modified to have the additional impression tracking tag.
10. The video player hits the additional tracking URL which goes to Prebid Server.
11. If client-side analytics is turned on it calls the analytics endpoint.

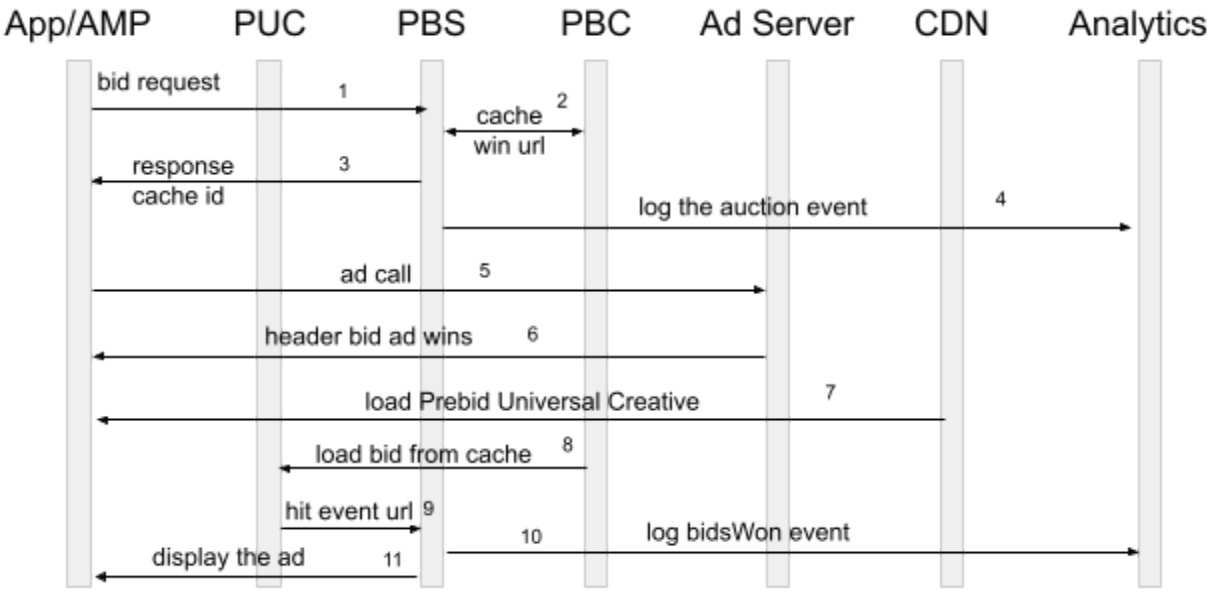
Mobile App Video

Similar to Web Server-Side Instream Video except with the SDK as the source instead of Prebid.js.

Mobile App and AMP Banner

Mobile apps and Accelerated Mobile Pages (AMP) can be thought of as the same scenario because they both:

- rely on Prebid Server for header bidding
- do not have access to a Prebid.js analytics adapter
- utilize the Prebid Universal Creative (PUC)



1. The mobile app or the AMP page calls Prebid Server, which conducts an auction.
2. Prebid Server caches the bid in Prebid Cache.
3. The bids are returned to the client.
4. Prebid Server initiates the analytics call.
5. App/AMP makes the ad call call
6. When header bidding wins, the ad server returns a block of javascript.
7. The Prebid Universal Creative is loaded.
8. Which loads the winning creative from Prebid Cache.
9. The BidsWon event URL is hit.
10. Which causes Prebid Server to call the analytics endpoint.

11. The ad is displayed in an iframe on the device.

The event URL has a format like this:

```
https://PBS_HOST/event?t=win&b=BIDID&f=i&a=ACCOUNT&ts=TIMESTAMP&bidder=BIDDER&int=INTEGRATION
```

Where:

- BIDID is replaced with the imp.bid.id
- BIDDER is the bidder code
- ACCOUNT is replaced with the publisher's account ID
- TIMESTAMP comes from ext.prebid.auctiontimestamp or is generated by PBS. Most likely the latter for app and AMP.

When Prebid Server receives this URL (when PxlUC performs 'hit event url' in the diagram above), the PBS analytics adapter will create the appropriate JSON request to send to the analytics endpoint ('log win event' in the diagram) and return to the browser with 1x1 pixel for display ('respond to client').

Prebid Server Details

This document was originally a requirements document for establishing Prebid Server tracking. The following sections have been converted to a reference.

Prebid Server VAST Tracking

Prebid Server supports injection of tracking into VAST documents in two ways:

- VAST bids that come through a server-side bid have tracking added if the account has enabled events unless the bid adapter has asked it not to in the YAML.
- Client-side VAST bids can be stored in Prebid Cache with tracking added through the /vtrack endpoint. The same account and bidder conditions are considered before adding tracking.

There are exactly 3 kinds of events that Prebid Server analytics emits and recognizes. Currently no other type of event is supported.

1. auction events: these are initiated by analytics adapters, not Prebid Server core.
2. win events: emitted for non-video bidresponses to track wins in the ad server
3. imp events:
    a. Injected into VAST and invoked by the video player when appropriate
    b. Offered to the client in non-video scenarios in case someone wants to add render tracking to their creatives.

Note: other VAST events may someday be supported.

The minimal VAST impression tracking URL is:

```
https://PBS_HOST/event?t=imp&b=BIDID&f=b&a=ACCOUNT&ts=TIMESTAMP&bidder=BIDDER&int=INTEGRATION
```

Where:

- t=imp defines this as an event of type impression
- f=b defines that the client expects a blank return
- BIDID is replaced with the imp.bid.id
- BIDDER is the bidder code
- ACCOUNT is replaced with the publisher's account ID
- TIMESTAMP comes from ext.prebid.auctiontimestamp or is generated by PBS

Other fields could be added as noted in #4 above if desired, though caution is suggested about supplying CPM values on the win event since they're easier to spoof. If CPM values are specified, it is recommended that some kind of encoding scheme is utilized.

Other tracking and caching notes:

1. If a bid adapter returns just a url (bidResponse.nurl) and not full VAST XML, Prebid generates a wrapper and inject impression tracking into the wrapper when appropriate per the other requirements.
2. Prebid Server caches VAST in Prebid Cache when request.ext.prebid.cache.vastxml is specified.
3. Prebid Server returns the modified VAST or VAST wrapper as cached **unless** ext.prebid.cache.vastxml.returnCreative exists and is false. The result goes in response.seatbid[].bid[].imp[].adm
4. Prebid Server should not set seatbid[].bid[].imp.ext.prebid.events for video requests -- the impression URL is in the VAST and this could cause double-counting.
5. Prebid Server sets seatbid[].bid[].imp.ext.prebid.events.{win,imp} for non-video events. This allows the client flexibility in determining how it will invoke tracking.

Prebid Server Event Algorithm Summary

Account-level configuration defines whether an account needs server-side analytics (and therefore events) by channel. e.g account 111 gets AMP/App events, but not web events. Account 222 gets events for all 3 channels: Web/AMP/App. If an account doesn't specify the channel config, the default is that AMP and App are "true" if events are enabled.

There are several event URL outputs that Prebid Server has to coordinate:

1. Auction Events
    1. Server-side analytics adapters will need access to the account-level event config so they can tell whether to log auction events.
2. Video impression Events - generated if events are enabled for the account because there's no such thing as client-side counting of video impressions.
    1. If the bidder returned only a 'nurl', then PBS creates a VAST wrapper and treats it as if it was the response from the adapter.
    2. Inject VAST <impression> tag
3. Win Events - when account events are on:
    1. Need to place 'wurl' in the PBC cache along with cached bids
    2. Needs to emit response.seatbid[].bid[].ext.prebid.events.win

Here's the event algorithm:

1. If the account doesn't support analytic events on this channel and request.ext.prebid.events is not defined, we're done - no need for any type of event logic.
2. If the bid request was video and the response is VAST XML or URL:
    1. If the response is just a VAST URL (bidresponse.nurl) without bidresponse.adm, generate a VAST wrapper and place it in bidresponse.adm, then continue to determine whether impressions should be injected.
    2. (If imp events are enabled for this account/channel OR request.ext.prebid.events is defined) AND we're allowed to modify the bidder's VAST, then inject an <impression> tag with this URL -- https://PBS_HOST/event?t=imp&b=BIDID&f=b&a=ACCOUNT&ts=TIMESTAMP&bidder=BIDDER&int=INTEGRATION
        i. If ext.prebid.cache.vastxml is specified, then cache the VAST in PBC.
        ii. If ext.prebid.cache.vastxml.returnCreative:false, the client is going to rely on the cached VAST, so remove seatbid[].bid[].imp[].adm for video responses.
3. If bid caching is turned on (ext.prebid.cache.bids) and the bidResponse isn't mediatype video
    1. If the win events are enabled for this account/channel combination **or** if the ext.prebid.events object is defined in the original request, then add 'wurl' to bids cached in PBC. URL is https://PBS_HOST/event?t=win&b=BIDID&f=i&a=ACCOUNT&ts=TIMESTAMP&bidder=BIDDER&int=INTEGRATION
4. Finally, for mediatypes other than video, consider setting seatbid[].bid[].ext.prebid.events.win and seatbid[].bid[].ext.prebid.events.imp
    1. If the win events are enabled for this account/channel combination or if the ext.prebid.events object is defined in the original request, then add
        i. Add response.seatbid[].bid[].ext.prebid.events.**win** to the openrtb output -- https://PBS_HOST/event?t=**win**&b=BIDID&f=i&a=ACCOUNT&ts=TIMESTAMP&bidder=BIDDER&int=INTEGRATION and
        ii. If imp events are enabled for this account/channel combination, Add response.seatbid[].bid[].ext.prebid.events.**imp** to the openrtb output --

https://PBS_HOST/event?t=**imp**&b=BIDID&f=i&a=ACCOUNT&ts=TIMESTAMP&bidder=BIDDER&int=INTEGRATION

bidType	account analytics channel flag && account events flag	request ext prebid events specified	bidder allows vast modify	request ext. prebid.cache.bids	Modify the VAST	Add bid ext prebid events	add bid.wurl to PBC	Add x=0 to event URL	Corresponds to algorithm step
video	false	no	*	*	no	no	no	no	1
video	yes	no	yes	*	yes	no	no	no	2.2
video	no	yes	yes	*	yes	no	no	no	2.2
video	*	*	no	*	no	no	no	no	2.2
banner native	yes	no	*	yes	no	yes	yes	no	3.1, 4.1
banner native	yes	no	*	no	no	yes	no	no	3.1, 4.1

Global algorithm steps not represented in the truth table because they're static rules that apply in all cases:

1. If the video response is just a VAST URL (bidresponse.nurl), always create a VAST wrapper
2. If ext.prebid.cache.vastxml is specified, then cache the VAST in PBC.
3. If ext.prebid.cache.vastxml.returnCreative:false, the client is going to rely on the cached VAST, so remove seatbid[].bid[].imp[].adm for video responses.