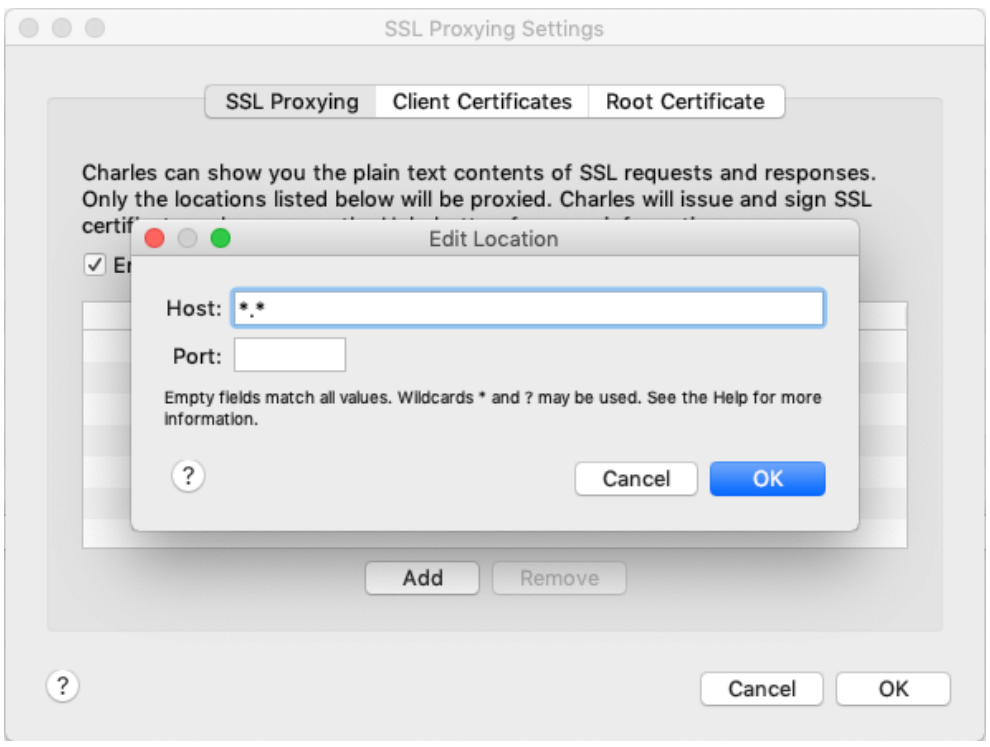
Integration Verification Guide

The purpose of this guide is to provide you with the necessary steps to verify the Amazon Publisher Services (APS) SDK integration.

Charles setup

To verify integration, we will use Charles, a web debugging proxy application to monitor app traffic. Download Charles here.

1. Once Charles is installed, go to Proxy > Proxy Settings, and click “Enable Transparent HTTP Proxying.”
2. Go to Proxy > SSL Proxying Settings. Click “Enable SSL Proxying Settings.” Then click Add, and enter “*.*” as the Host like the example below:



3. Go to Proxy, and uncheck “macOS Proxy” or “Windows / Internet Explorer Proxy” if applicable.

iOS

1. Go to Settings > Wi-fi, and connect to the same network as the computer running Charles.
2. Tap on the Wi-fi network name, and scroll down to “HTTP PROXY.” Tap “Configure Proxy,” and select Manual.
3. In Charles, select Help > Local IP Addresses. Enter the IP address in the Server section of the iOS proxy configuration page. Enter “8888” as the Port (this should match the port found in Proxy > Proxy Settings in Charles). Click Save.
4. In the iOS device, visit https://chls.pro/ssl in the browser. Click Allow to install the Charles certificate.
5. If the device is on iOS version 10.3+, go to Settings > General > About > Certificate Trust Settings, and tap the switch to trust the downloaded Charles certificate.

Android

Before using Charles to monitor your Android app traffic, you must add configuration to the project to trust Charles on Android devices on version N and later (v7.0+). Otherwise, Charles will be blocked from monitoring any requests on the Android device. Add the file res/xml/network_security_config.xml to your Android project:

```
<network-security-config>
  <debug-overrides>
    <trust-anchors>
      <!-- Trust user added CAs while debuggable only -->
      <certificates src="user" />
    </trust-anchors>
  </debug-overrides>
```

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:networkSecurityConfig="@xml/network_security_config" ... >
        ...
    </application>
</manifest>
```

Steps to connect Android device to Charles:

1. Go to Settings > Wi-fi and connect to the same network as the computer running Charles.
2. Tap and hold the Wi-Fi name. Select “Modify Network” from the pop-up window. Tap on “Show advanced options,” and select Manual from the “Proxy settings” dropdown.
3. In Charles, select Help > Local IP Addresses. Enter the IP address in the “Proxy hostname” section of the Android proxy configuration window. Enter “8888” as the “Proxy port” (this should match the port found in Proxy > Proxy Settings in Charles). Click Save.
4. In the Android device, visit https://chls.pro/ssl in the browser. Click Download to install the Charles certificate.

Verification steps

Please make sure that both test mode and debug logging are turned on for the build’s integration that is being verified.

iOS:

```
[[DTBAds sharedInstance] setTestMode:YES];
[[DTBAds sharedInstance] setLogLevel:DTBLogLevelAll];
```

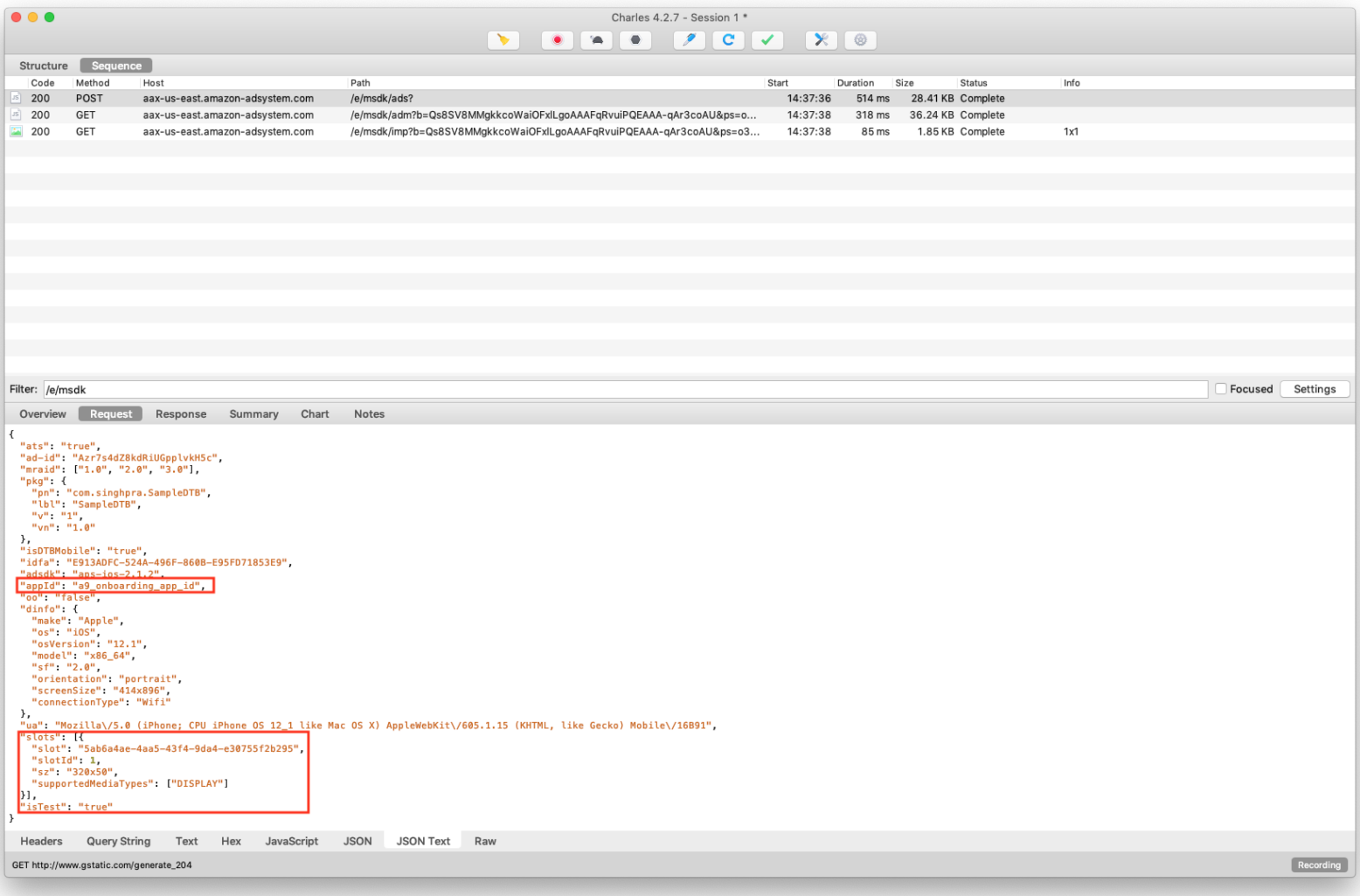
Android:

```
AdRegistration.enableTesting(true);
AdRegistration.enableLogging(true);
```

Once your device is connected to Charles, go to the page in your app where the Amazon Publisher Services (APS) SDK is implemented to request and serve ads. Amazon ad requests containing “/e/msdk” should start populating the Charles log. You can filter for “/e/msdk” to show only Amazon requests.

APS bid request

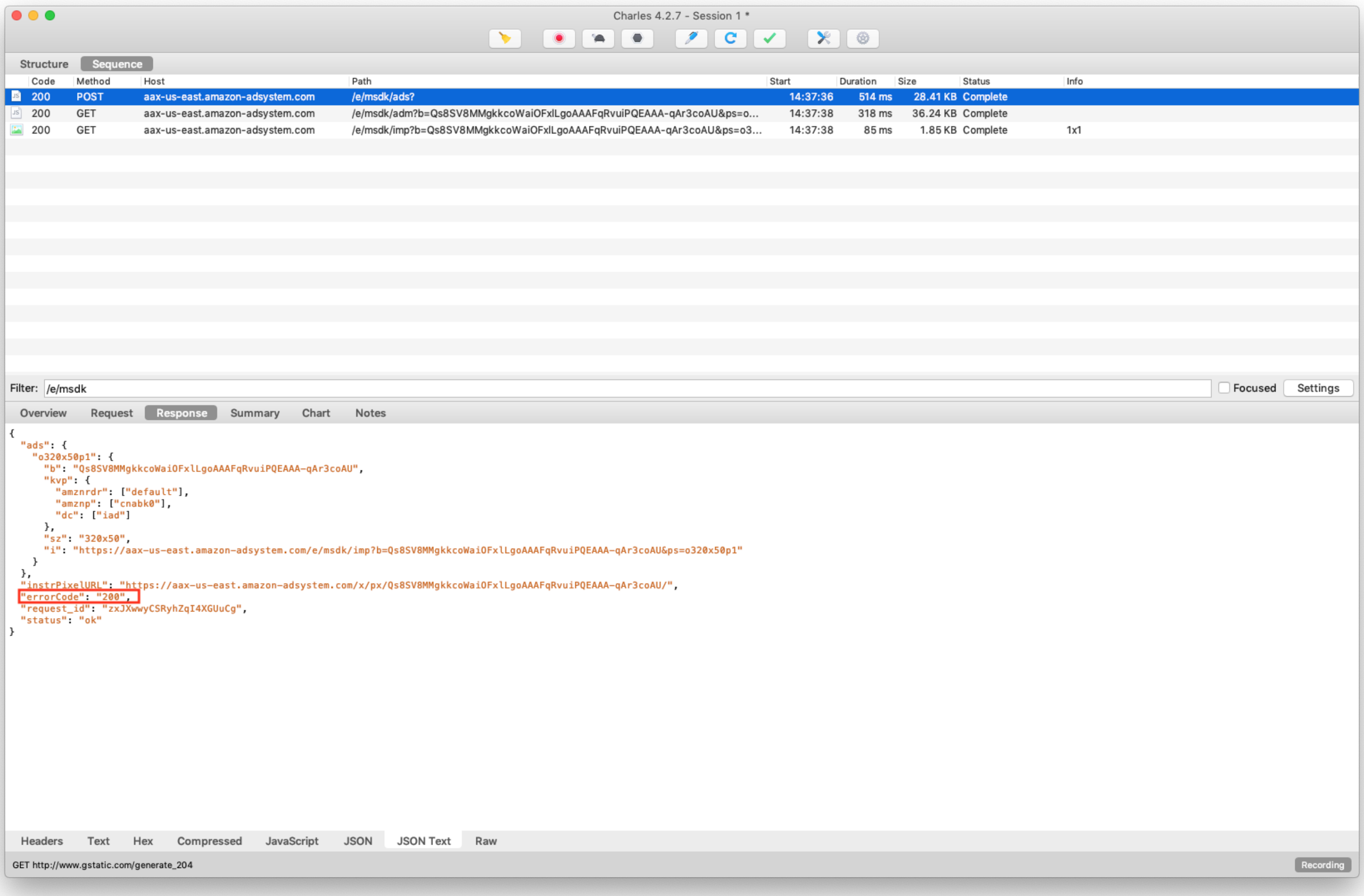
The bid request (/e/msdk/ads) should contain the isTest parameter set to “true.” Please make sure the appId, slot, and ad dimensions (sz) are all valid and corresponding to each other. Contact us for further assistance.



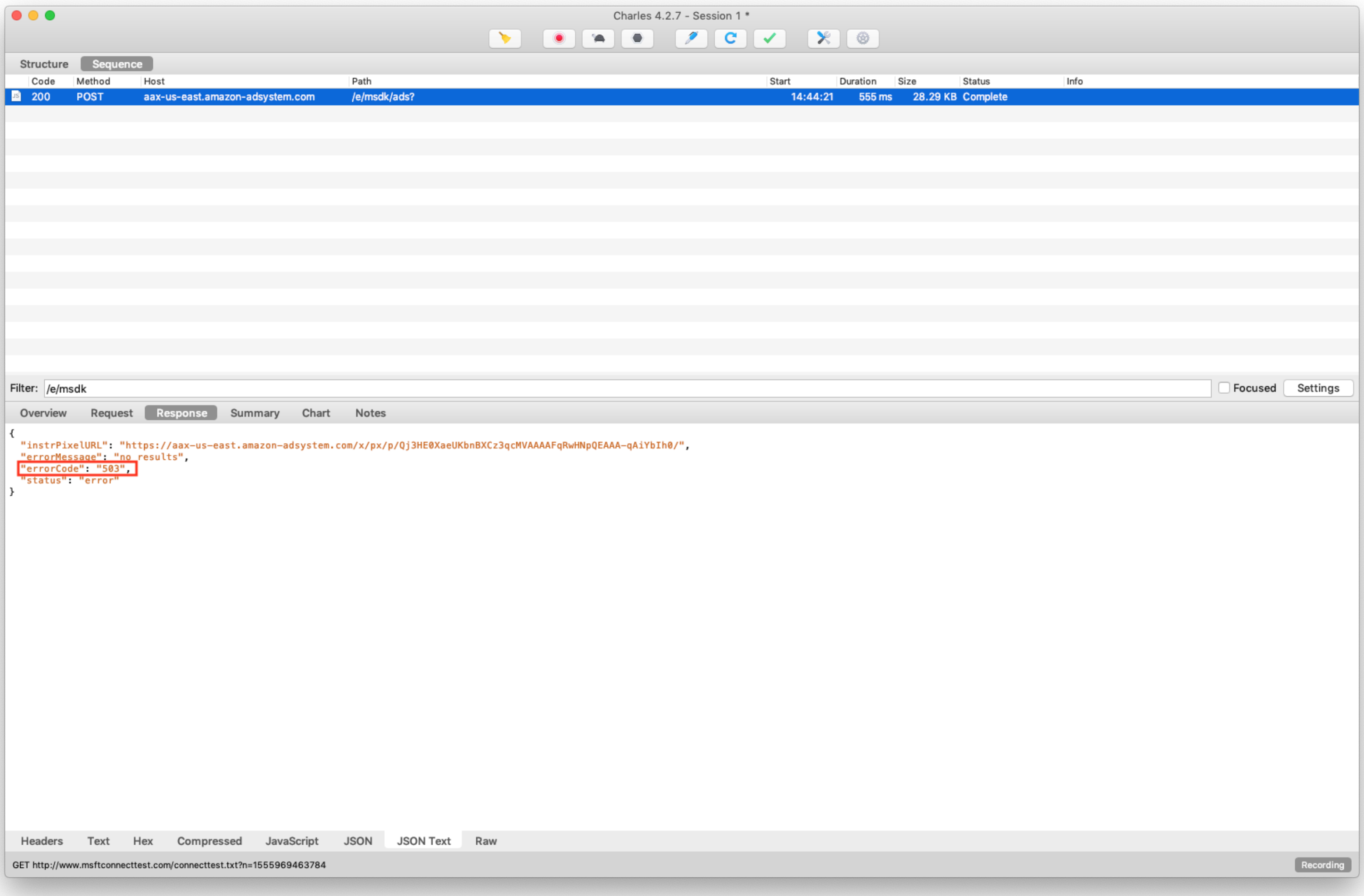
APS bid response

Three bid response codes are possible:

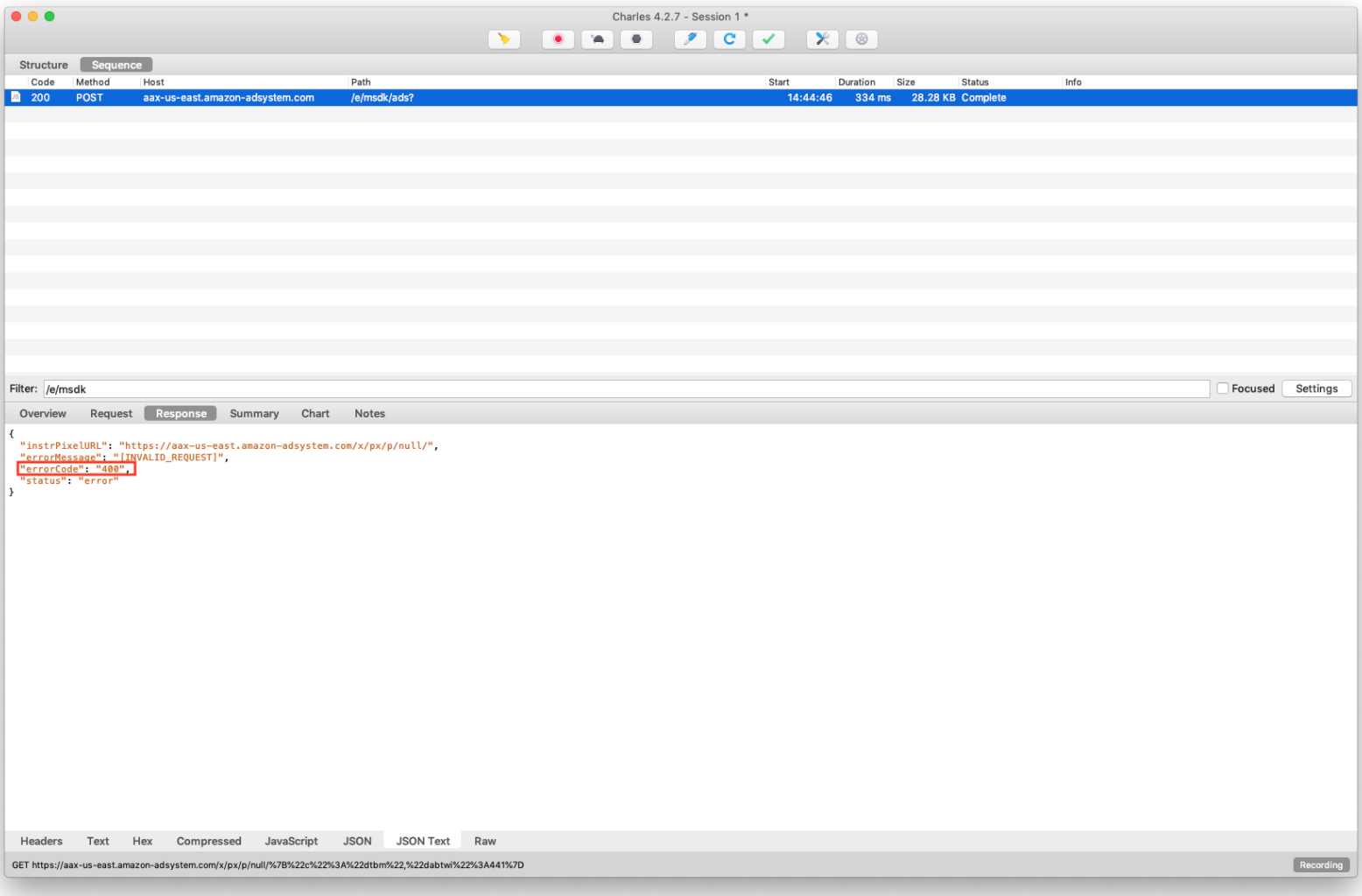
- **200 – A successful bid response has been returned.**



- 503 – There is no bid available. (This is not an error.)
  - Test mode is turned off: If you have test mode off, this is expected behavior since Amazon does not have 100% bid rate in production. Bidding in production mode is heavily influenced by the user and device, so the device ID in the request, for example, can lead to occasional no-bid responses.
  - Test mode is turned on: If you have test mode on, "no bid available" can be the result of making bid requests from the EU. Please check whether you have GDPR implemented correctly according to the IAB's GDPR guidelines.



- 400 – There is an error in the bid request.
  - This is due to a mismatch between the appId, slotId, and ad dimensions. Please make sure these values are all valid and corresponding with each other.



Mediation platform request

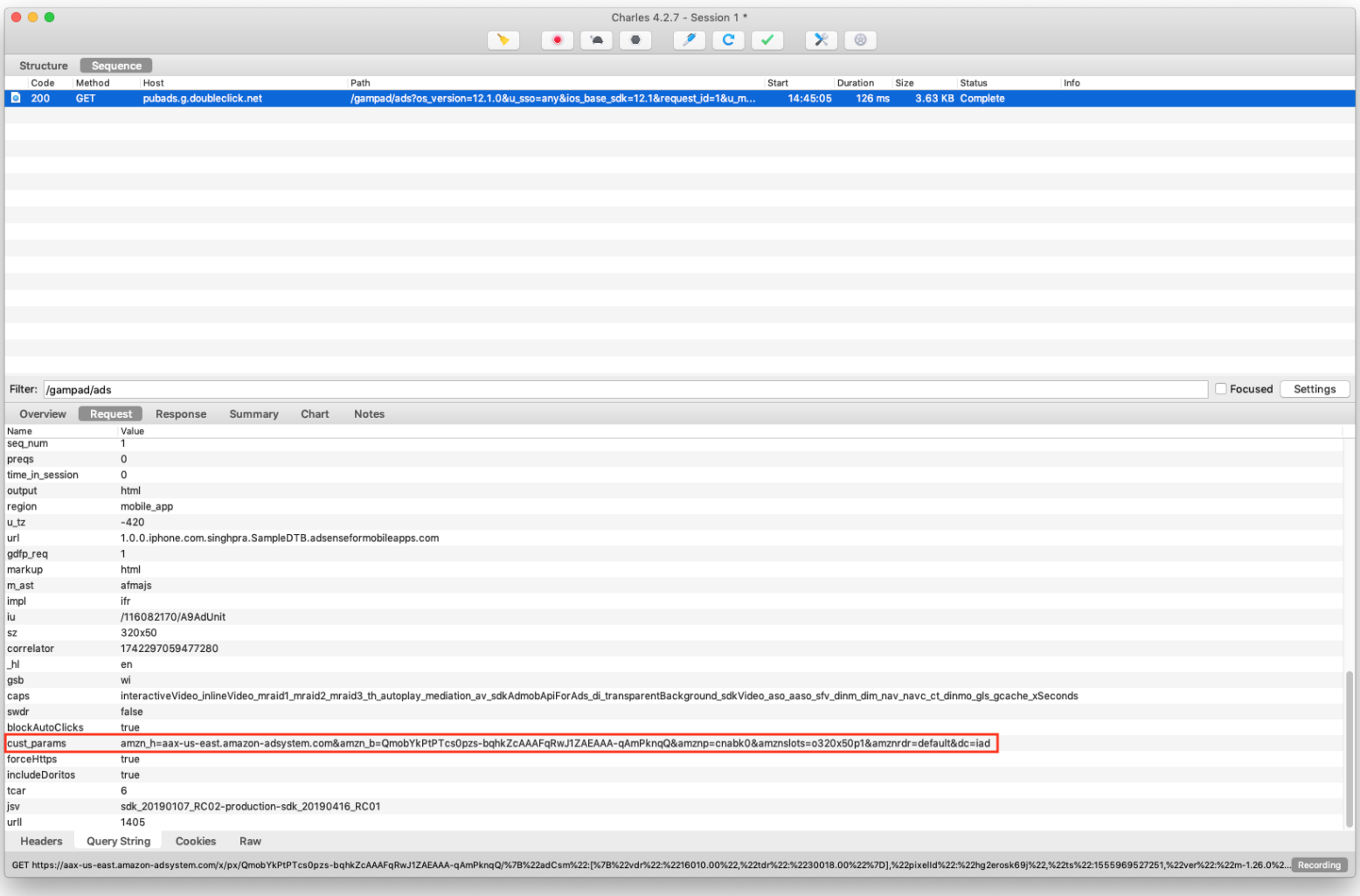
When APS responds successfully with a bid (200 status code), the APS bid parameters should be forwarded in the following ad server request.

Check whether the following bid parameters are included in the ad server request:

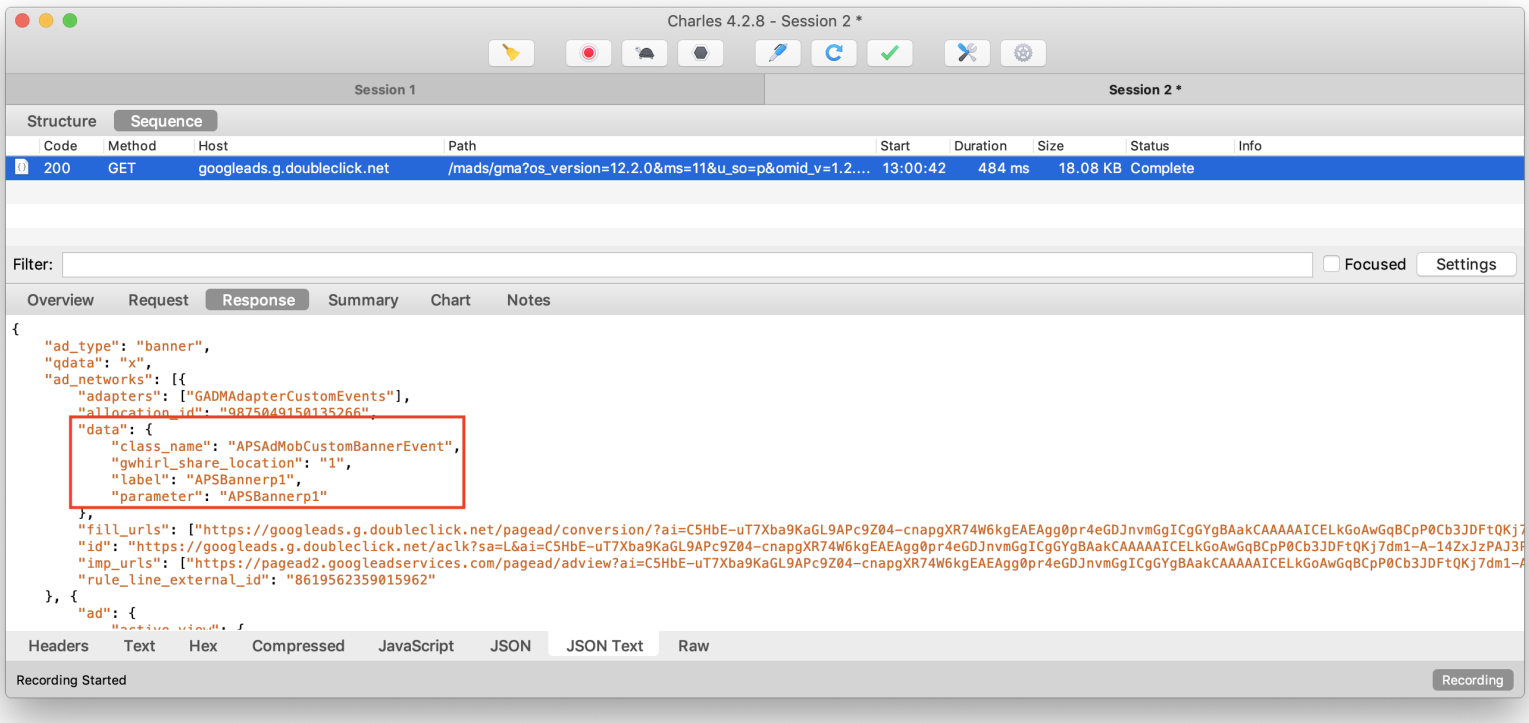
- amzn_b (or amzn_vid for video) – bid ID
- amzn_h – host name
- amznp – hashed bidder ID
- amznslots – pricepoint of bid

Google Ad Manager example:

The Amazon keywords are set on the “cust_params” key in GAM’s GET request.



Google AdMob example:



Make sure debug log is enabled. Go to your console log and filter by “DTB – Trace.” Once AdMob responds back, you should see the log “Amazon Custom Event was hit.” This indicates that our Custom Event was invoked by AdMob. Following this log, you should see either “Amazon Custom Event was accepted” or “Amazon Custom Event was ignored” printed out. It is normal to see both on your log statements but the last log should be “Amazon Custom Even was accepted,” because this indicates that our ad will render.

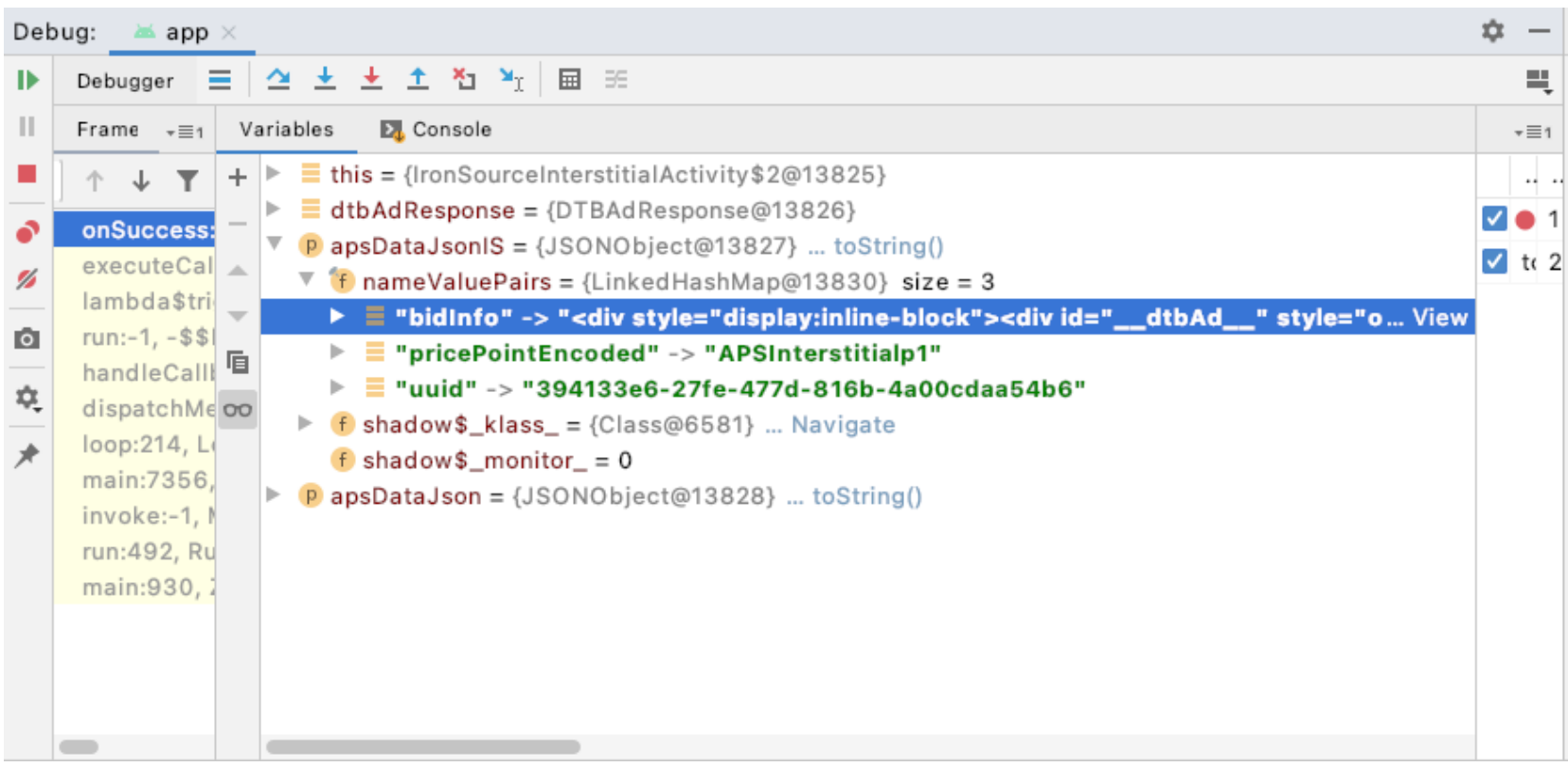
Sample logs:
“Amazon Custom Event was hit”
“Amazon Custom Event was ignored”
“Amazon Custom Event was hit”
“Amazon Custom Event was ignored”
“Amazon Custom Event was hit”
“Amazon Custom Event was accepted”

If our “Amazon Custom Event was accepted” statement does not show up, either something is not set up correctly on the AdMob dashboard or another partner within your waterfall has won the auction.

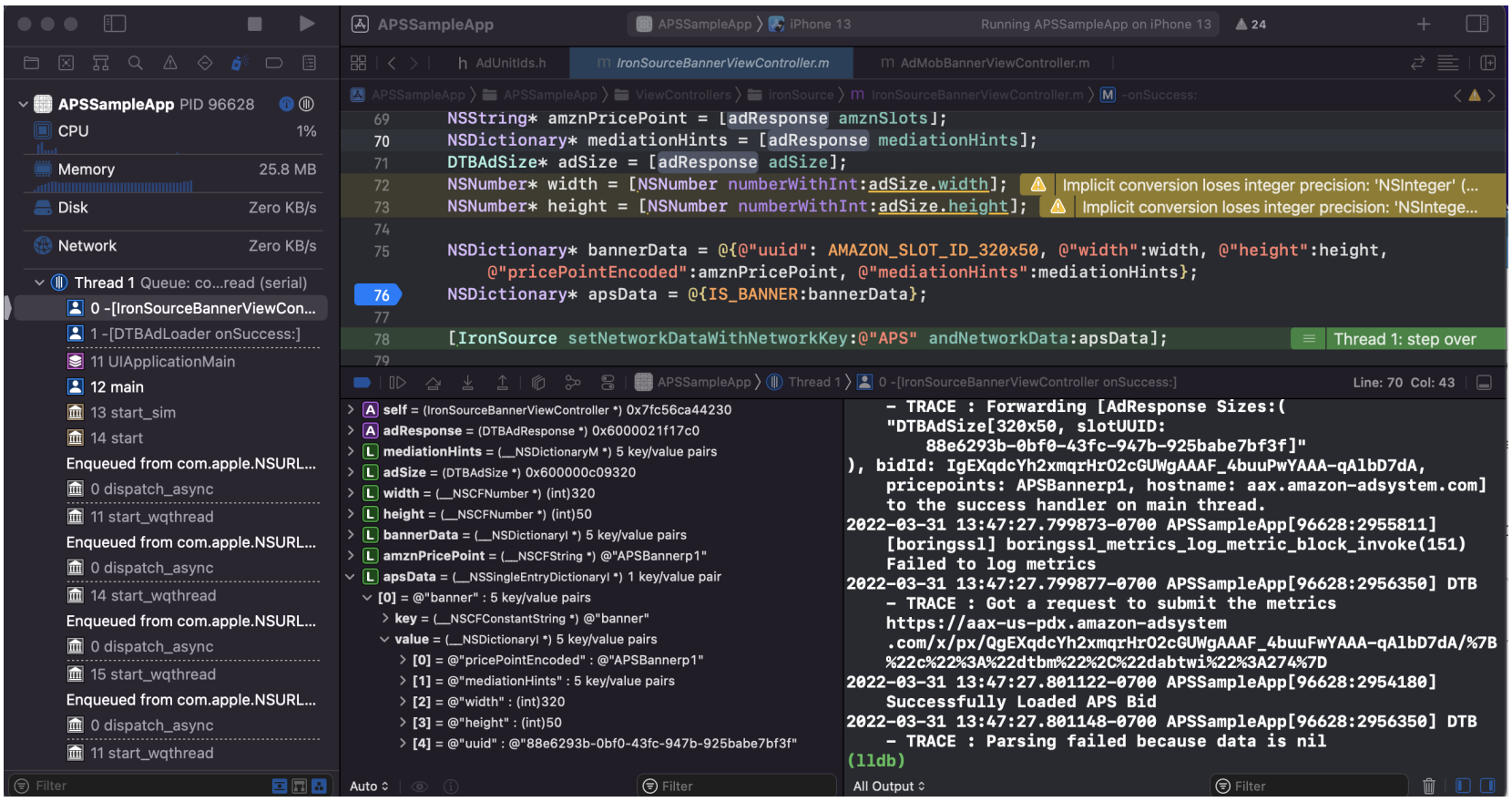
Unity LevelPlay Example:

To check if the Amazon keys are passed to Unity LevelPlay, place a break point inside the onSuccess and make sure that you have all the values set.

Android

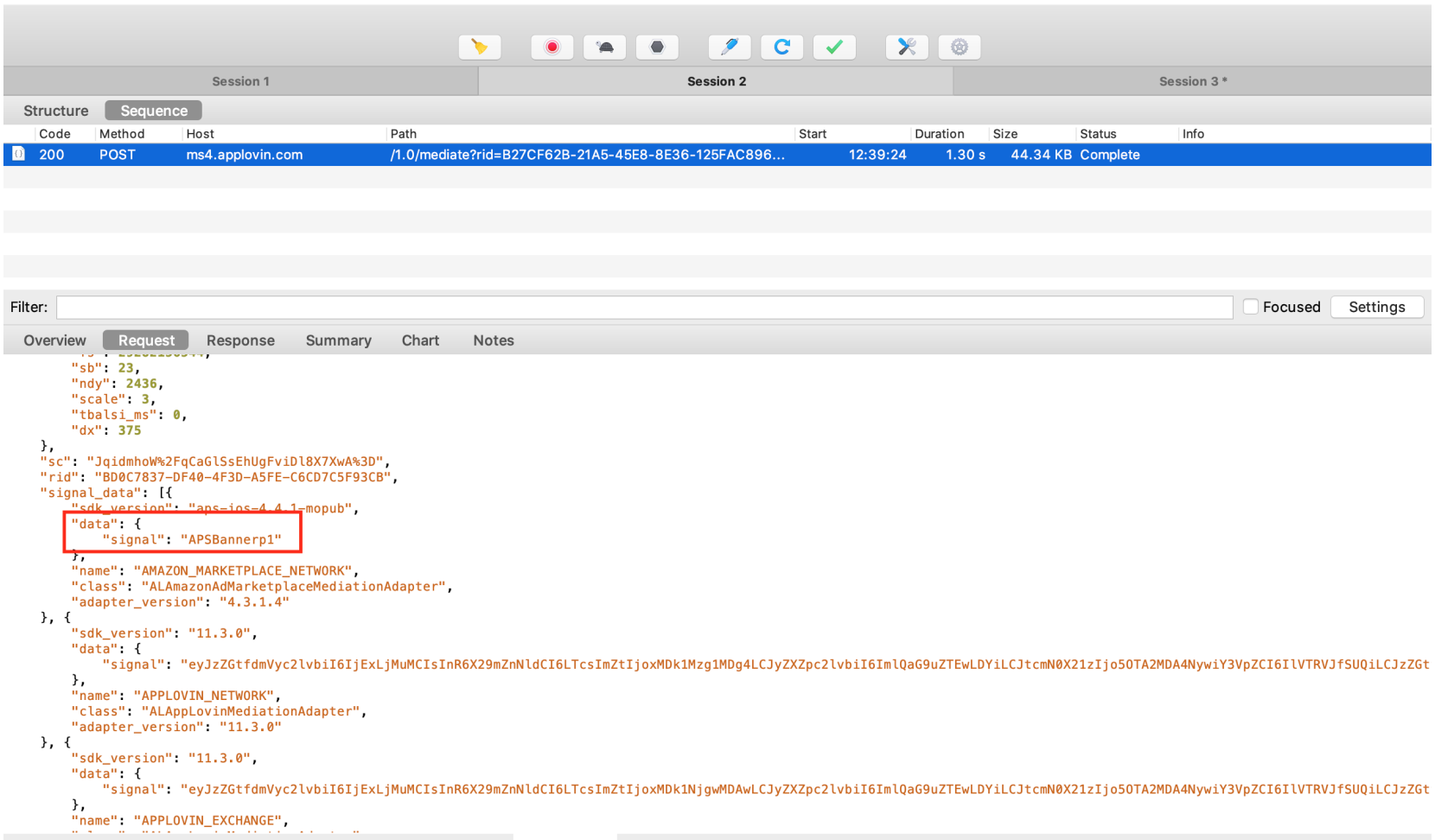


iOS



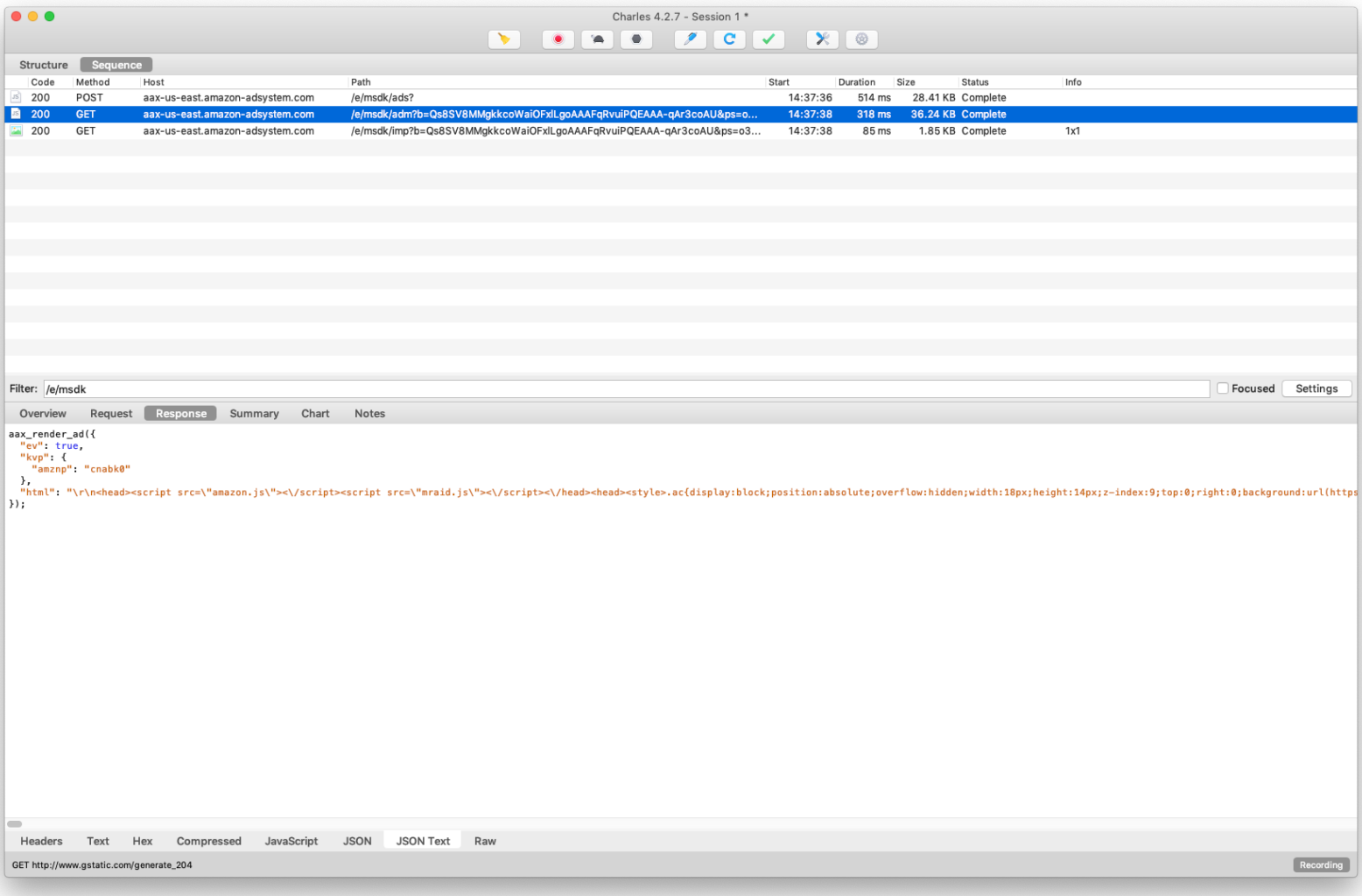
MAX example:

In the MAX bid request, under the signal_data object, you can find the Amazon price point key. Make sure that this matches with the price point key that APS returns in the bid response.

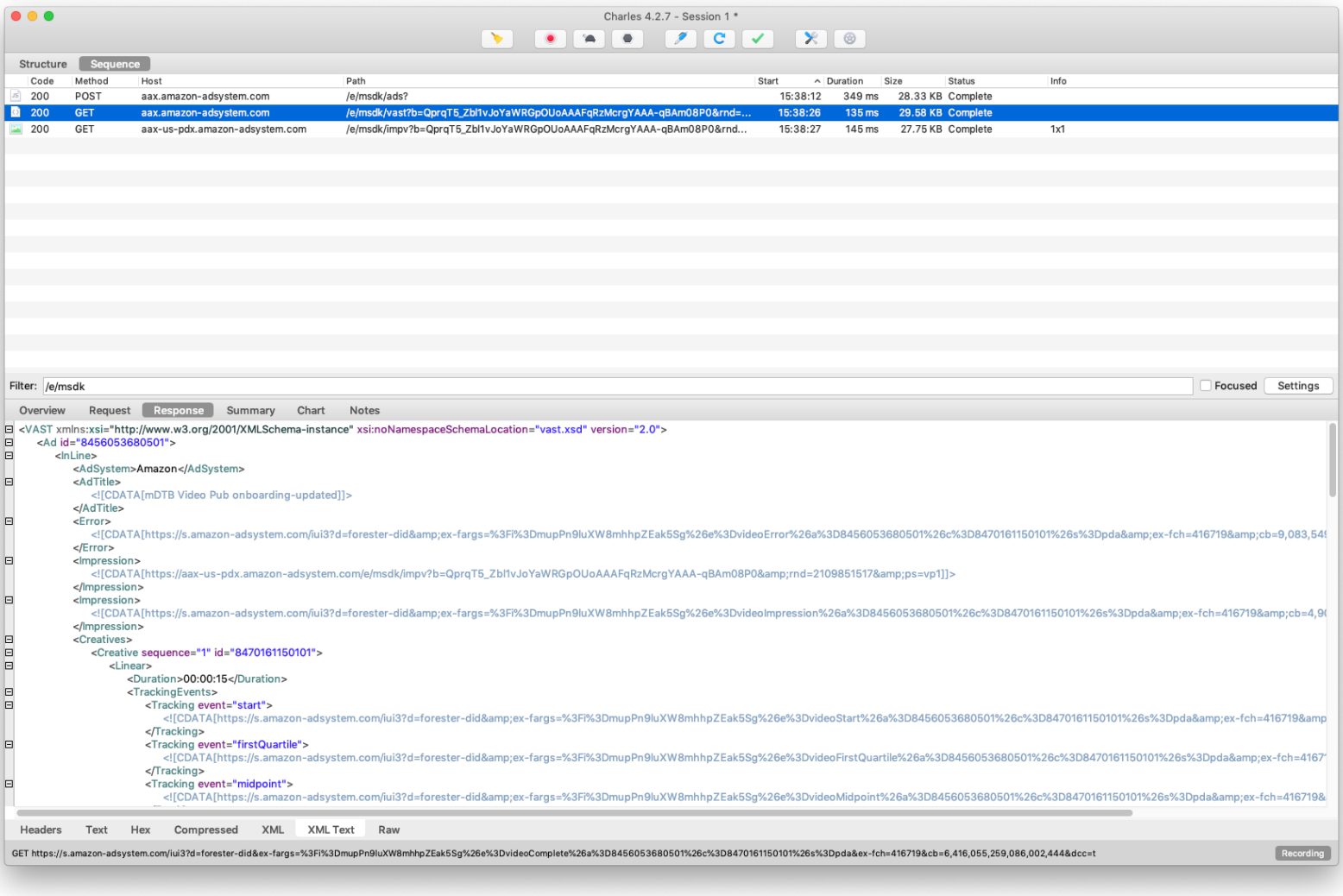


Ad rendering

For display bids that win the ad server auction, the rendering url (/e/msdk/adm) will execute the ad-rendering function in the response. The function will contain the HTML ad-markup to be rendered.

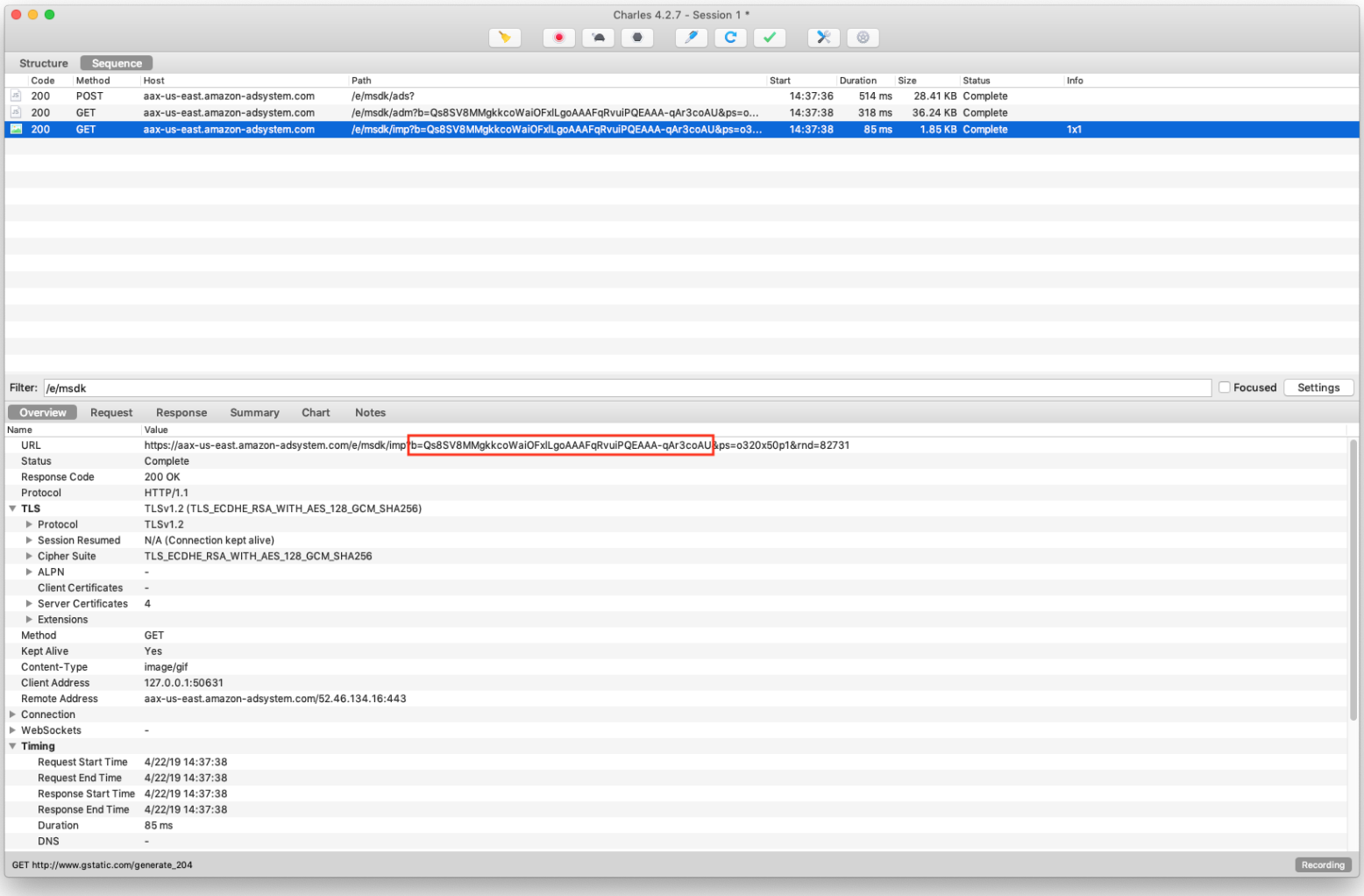


For video bids that win the ad server auction, the rendering url (/e/msdk/vast) will contain the VAST XML ad-markup in the response.



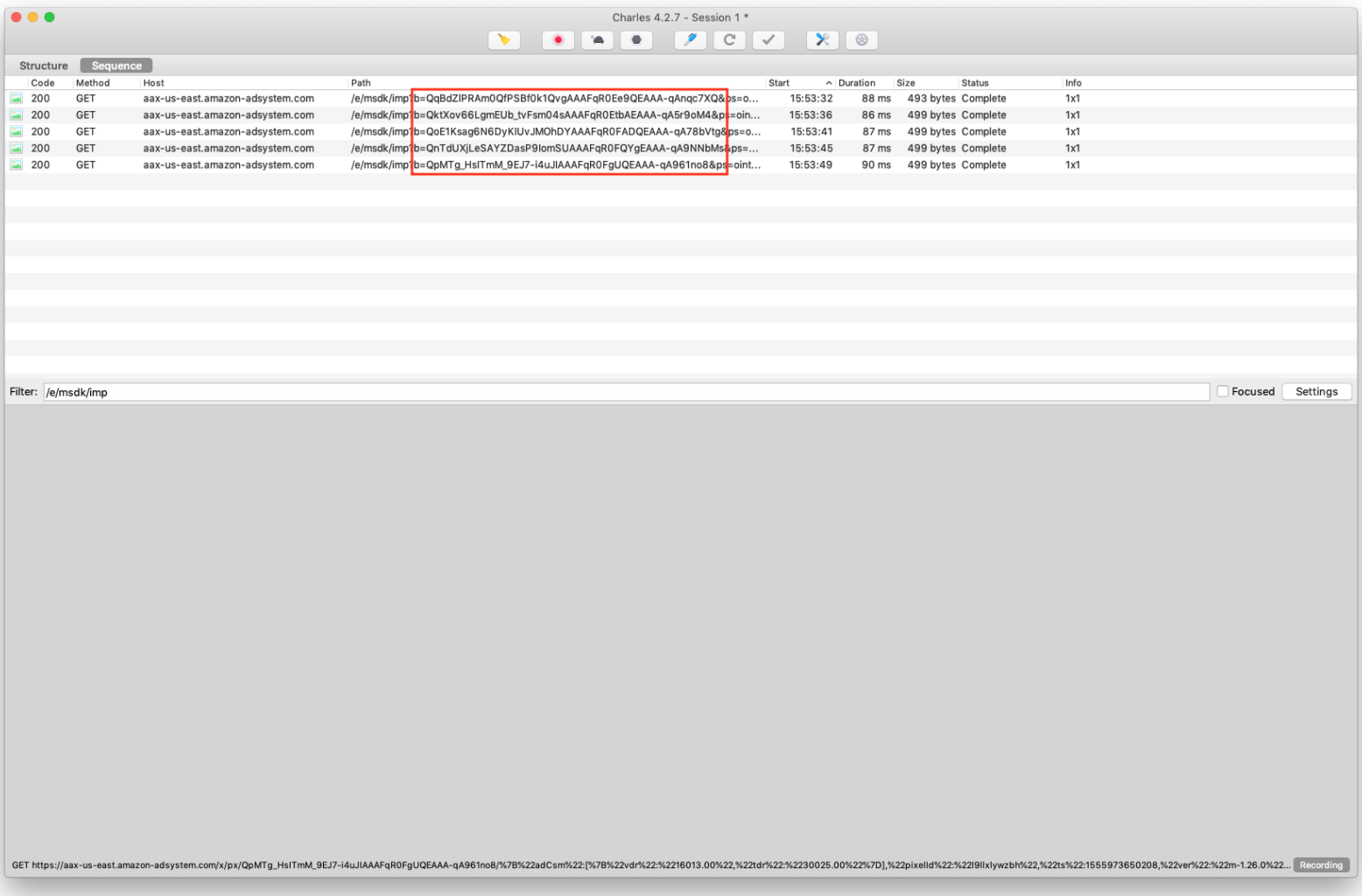
Ad impression

The impression pixel (/e/msdk/imp) should fire after the Amazon ad has rendered. The “b=” parameter contains the bid ID. This value is important for identifying impressions.



Auto-refresh

If you have implemented auto-refresh, please check if the impression pixels ("/e/msdk/imp" for display or "/e/msdk/impv" for video) contain unique bid IDs. This is found in the "b=" parameter of the url. It is important that bid IDs are not reused, since they are only valid for one impression. Re-used bid IDs will not be monetized and will cause large reporting discrepancies with your ad server. Below is how the Charles log should look when auto-refresh is working properly:



You can also filter by ad server requests in Charles to check if the same bid ID (amzn_b) is being forwarded for multiple requests.

Bids from our integration will last 10 minutes. Any use of a bid ID after the expiration time period will result in a blank ad for the user and an unpaid impression.

Please turn off test mode before going live in production. Failure to do so will result in unpaid test impressions from our integration.

? Help