Dual Prebid Implementation Guide

Document Scope **1**

Prerequisites **1**

Implementation **2**

- Step 1 - add custom prebid.js build to you page. 2
- Step 2 - define initial timeout and adUnit Configuration 2
- Step 3 - initialise GPT and define prebid functions on page used to que and push bids for ad-server targeting. 4
- Step 4 - update your existing ‘pbjs’ / hosted prebid build 7

Sample Test Page **8**

Document Scope

The purpose of this document is to explain the differences between a standard prebid integration vs. a custom integration of running 2 prebid instances on page.

Prerequisites

It is assumed that:

- You already have a prebid JS instance on page using the standard prebid JS name space - ‘pbjs’
- You are happy to host a custom prebid build for your second prebid JS instance on page that will use a non-standard namespace - eg: `pbjs.newspassid.com’. Ozone Project is happy to provide this build to you.
- You will integrate the code amends we recommend below AFTER your existing hosted/pbjs instance build is called.
- You will be required to make a small amendment to your existing ‘pbjs’ build. A separate document is available explaining what needs to be changed here. Without doing this - you risk having one prebid instance not being able to provide DFP with the targeting information that’s needed on page.

Implementation

This section of the document will reference the standard basic prebid JS implementation example on prebid.org and point out the modifications that are needed on page for the additional prebid build this document is advising on adding.

The standard prebid example code can be seen here - https://docs.prebid.org/dev-docs/examples/basic-example.html

Step 1 - add custom prebid.js build to you page.

A custom prebid.js file will be provided to you - add this to your page so it's called after your existing/original prebid JS file is loaded.

Step 2 - define initial timeout and adUnit Configuration

Per the standard prebid example you typically would define the following:

```
        <script>
            var div_1_sizes = [
                [300, 250],
                [300, 600]
            ];
            var div_2_sizes = [
                [728, 90],
                [970, 250]
            ];
            var PREBID_TIMEOUT = 1000;
            var FAILSAFE_TIMEOUT = 3000;

            var adUnits = [
                {
                    code: '/19968336/header-bid-tag-0',
                    mediaTypes: {
                        banner: {
                            sizes: div_1_sizes
                        }
                    },
                    bids: [{
                        bidder: 'appnexus',
                        params: {
                            placementId: 13144370
                        }
                    }]
                },
                {
                    code: '/19968336/header-bid-tag-1',
                    mediaTypes: {
                        banner: {
                            sizes: div_2_sizes
                        }
```

```
                    },
                    bids: [{
                        bidder: 'appnexus',
                        params: {
                            placementId: 13144370
                        }
                    }]
                }
            ];
```

The following amendments are advised here when implementing an additional build for prebid on page:

- Amend timeout variable namespace
- Amend adUnit variable namespace

The code snippet from the basic example on the previous page therefore will look something like this once amended (edits highlighted in yellow for clarification)

```
        <script>
            var div_1_sizes = [
                [300, 250],
                [300, 600]
            ];
            var div_2_sizes = [
                [728, 90],
                [970, 250]
            ];
            var NP_PREBID_TIMEOUT = 3000;
            var NP_FAILSAFE_TIMEOUT = 4000;

            var adUnitsNewspassid = [
                {
                    code: '/19968336/header-bid-tag-0',
                    mediaTypes: {
                        banner: {
                            sizes: div_1_sizes
                        }
                    },
                    bids: [{
                        bidder: 'appnexus',
                        params: {
                            placementId: 13144370
                        }
                    }]
                },
                {
                    code: '/19968336/header-bid-tag-1',
                    mediaTypes: {
```

```
                        banner: {
                            sizes: div_2_sizes
                        }
                    },
                    bids: [{
                        bidder: 'appnexus',
                        params: {
                            placementId: 13144370
                        }
                    }]
                }
            ];
```

Note the adUnits are examples here - you should define your actual adUnit setup here.

Step 3 - initialise GPT and define prebid functions on page used to que and push bids for ad-server targeting.

Per the basic example prebid recommends adding the following code to your page:

```
            var googletag = googletag || {};
             googletag.cmd = googletag.cmd || [];
             googletag.cmd.push(function() {
                 googletag.pubads().disableInitialLoad();
             });

             var pbjs = pbjs || {};
             pbjs.que = pbjs.que || [];

             pbjs.que.push(function() {
                 pbjs.addAdUnits(adUnits);
                 pbjs.requestBids({
                     bidsBackHandler: initAdserver,
                     timeout: PREBID_TIMEOUT
                 });
             });

             function initAdserver() {
                 if (pbjs.initAdserverSet) return;
                 pbjs.initAdserverSet = true;
                 googletag.cmd.push(function() {
                     pbjs.que.push(function() {
                         pbjs.setTargetingForGPTAsync();
                         googletag.pubads().refresh();
                     });
                 });
             }
             // in case PBJS doesn't load
```

```
            setTimeout(function() {
                initAdserver();
            }, FAILSAFE_TIMEOUT);
```

For the custom build a large part of this code will change and can be copied and pasted below.

Note: We omit the GPT code in the basic example as you already will have this defined in your existing integration code from your hosted prebid JS example. We also have added additional code snippets that handle:

- cookie/user sync
- Disabling standard prebid JS targeting keys from this instance of prebid JS to avoid conflicts with your hosted/original ‘pbjs’ version.
- A bespoke/custom function that handles synchronisation of targeting keys to DFP from both prebid JS instances.
- Generic timeout handler to continue with loading ads should auctions exceed timeout.
- Additional code for the newspass bidder to call its bespoke/custom endpoint

```
        pbjs_newspassid.que.push(function() {
            pbjs_newspassid.setConfig({
                userSync: {
                    aliasSyncEnabled: true,
                    filterSettings: {
                        iframe: {
                            bidders: '*',
                            filter: 'include',
                            iframeEnabled: true
                        }
                    },
                    userIds: [
                        {
                            name: "pubCommonId",
                            storage: {
                                type: "cookie",
                                name: "_pubcid",
                                expires: 365
                            }
```

```
                    }
                ]
            },
            newspassid: {
                singleRequest: true,
                enhancedAdserverTargeting: true,
                kvpPrefix: 'np',
                endpointOverride: {
                    origin: 'https://bidder.newspassid.com'
                }
            }

        });
        pbjs_newspassid.addAdUnits(adUnitsNewspassid);
        pbjs_newspassid.requestBids({
            bidsBackHandler: sendAdserverRequestNewpassid
        });

        // for the non-standard pbjs build we need to block the standard keys
        pbjs_newspassid.bidderSettings = {
            standard: {
                sendStandardTargeting: false
            }
        };
    });


      function sendAdserverRequestNewpassid() {
                if (pbjs_newspassid.adserverRequestSent) return;
                pbjs_newspassid.adserverRequestSent = true;
                googletag.cmd.push(function() {
                    pbjs_newspassid.que.push(function() {
                        pbjs_newspassid.setTargetingForGPTAsync();
                        pbjsSync.registerTargetingSet('pbjs_newspassid');
                    });
                });
            }

            var pbjsSync = {
                arrDone: [],
```

```
                registerTargetingSet: function(name) {
                    this.arrDone.push(name);
                    if(this.arrDone.length > 1) {
                        googletag.pubads().refresh();
                    }
                }
            }

            setTimeout(function() {
                sendAdserverRequestNewpassid();
            }, NP_PREBID_TIMEOUT);
```

Step 4 - update your existing 'pbjs' / hosted prebid build

The existing pbjs build will require an edit to the standard code that's being used.

The pbjs instance of bidsBackHandler will typically call a function - initAdServer - which handles set the targeting keys and tells DFP to fetch ads. EG:

```
pbjs.que.push(function() {
    pbjs.addAdUnits(adUnits);
    pbjs.requestBids({
        bidsBackHandler: initAdserver,
        timeout: PREBID_TIMEOUT
    });
});

function initAdserver() {
    if (pbjs.initAdserverSet) return;
    pbjs.initAdserverSet = true;
    googletag.cmd.push(function() {
        pbjs.que.push(function() {
            pbjs.setTargetingForGPTAsync();
            googletag.pubads().refresh();
        });
    });
}
// in case PBJS doesn't load
setTimeout(function() {
    initAdserver();
}, FAILSAFE_TIMEOUT);
```

The initAdserver function will need to be amended so that it no longer calls googletag.pubads().refresh but calls our custom function (pbjsSync) so that both prebid instances targeting can be returned before the page attempts to call DFP/GPT - for example:

```
            pbjs.que.push(function() {
                pbjs.addAdUnits(adUnits);
                pbjs.requestBids({
                    bidsBackHandler: initAdserver,
                    timeout: PREBID_TIMEOUT
                });
            });

            function initAdserver() {
                if (pbjs.initAdserverSet) return;
                pbjs.initAdserverSet = true;
                googletag.cmd.push(function() {
                    pbjs.que.push(function() {
                     if(!window.pbjsSync) {
                    console.error('pbjs ERROR: no pbjsSync object found');
                }
                        pbjs.setTargetingForGPTAsync();
                        pbjsSync.registerTargetingSet('pbjs');
                    });
                });
            }
            // in case PBJS doesn't load
            setTimeout(function() {
                initAdserver();
            }, FAILSAFE_TIMEOUT);
```

Sample Test Page

A working example can be seen here - https://www.ardm.io/ozone/two_pbjs/encapsulated_config/dualprebid.html

---