PRD Prebid Server Price Floors

Updated Feb 20, 2025

Change Log

1. Overview

The document describes the Price Floor feature for Prebid Server (PBS) matching many of the Prebid.js (PBJS) features defined here. Prebid Server will contain its own set of requirements given the additional flexibility PBS has to execute code in a server-side environment.

The workflow will remain the same in PBS as in PBJS:

1. Publisher signs with a Price Floor provider
2. Price Floor provider makes a JSON file available with the data
3. Publisher works with their engineering team or PBS host provider to configure price floors

PBS Floors should support Prebid.js, Prebid AMP, and Prebid SDK integrations.

1.1. Goals

- Provide a Prebid server-side floor solution controlled by the publisher
- Allow for an open framework to set floors manually or work with a floor provider
- Provide a flexible framework to allow floor providers to integrate with the Floor feature

1.2. Assumptions

1. The Floors feature enhances the ORTB2 field imp.bidfloor so server-side bidders immediately get smarter floors without having to update their code.
2. This system does not need to support arbitrary schema fields like PBJS does. Any newly required fields will be added as enhancements in a future project.
3. When validating floor configurations, the general strategy is to allow the auction to happen even if floors cannot be processed:
    a. If there's a problem at PBS **startup**, log a nice error and don't let the server start.
    b. If there's a problem with **account floor configuration**, log a warning and process the auction normally.
    c. If there's a problem with **dynamically-fetched floor rule** syntax, log a warning, reject the update, relying on the previously cached rules until a valid fetch is made.
    d. If there's a problem with **request-based floor rule** syntax, whether with the overall JSON structure of the schema or internal to a rule, reject the block and add a warning to debug mode.

    e. If there's a currency conversion problem, no module floor activity can take place. Use the default request floor if available. Emit a debug warning and log an error at N% sampling.
4. We will define the ORTB2 extensions in camel-case. This differs from our normal convention of lower-case, but this is an existing format, and it would be difficult to force Floors providers to produce different files for PBJS and PBS.
5. Floor currency matters only to the Floor system. It's possible that the ad server currency, the bid currency, and the floor currency are all different, but PBS-core currency processing does not need to take floor currency into account.
6. The high-performance auction threads will never interact directly with the dynamic fetch thread. This implies that the fetch thread configuration cannot be changed with the regular account-config that flows through the auction workflow. Rather, changes to the behavior of the dynamic fetch will be done with host-level config and/or a new config mechanism.
7. Floors should be enforced even on test requests. Getting test bids could be combined with the 'enforcePBS' flag to control whether PBS is rejecting bids for floor reasons.
8. The publisher must be able to set a minimum floor.
9. The following fields are added to the PBJS Floors 'Schema 2':
    a. floorMinCur
    b. enforceRate
    c. enabled
    d. fetchStatus
    e. skipped
    f. location
10. The following fields in PBJS Floors 'Schema 2' are ignored in Prebid Server:
    a. endpoint

1.3. Future Features

1. Detailed floor min feature
2. It's expected that someday we'll generalize some of these features into services that modules could use. For example, Real Time Data providers will need a 'fetch thread' quite similar to the one used by this feature.
3. We could someday add additional dynamic floor dimensions like 'previous bids in a pageview' or 'session depth'.

2. High Level Requirements

1. The attributes used to determine the price floor must be flexible. e.g. one floor provider may determine a specific floor value with a combination of domain and media type, while another provider may utilize a different schema

2. Floors data should be retrieved from a resolved OpenRTB request or via a dynamic fetch from a floors provider.
    a. The schema of attributes used for choosing the price floor may differ in these sources.
    b. The feature must validate floor rules from both sources.
    c. Floors validation failure cannot reject the entire auction.

3. The Floors feature, when active, should set imp.bidfloor based on the rule that best matches known parameters. This allows floors to be utilized by bid adapters that don't explicitly call for custom floors.

4. The PBJS will be responsible for translating schema 1 rules to the supported schema 2 of PBS.

5. There should be a standard method that bidders can utilize to retrieve a custom floor from the Floors feature

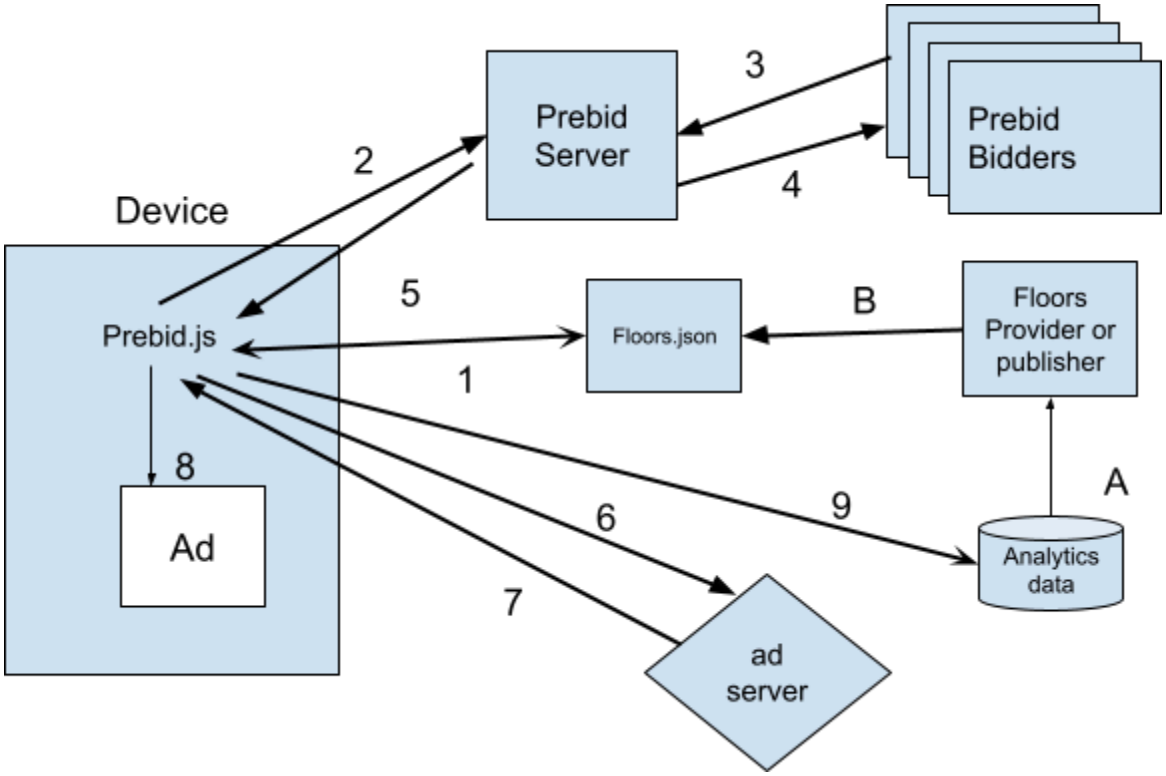
6. The feature should adjust floors where necessary, specifically for currency conversions and / or bid adjustments.
    a. The system must support allowing floor providers to specify the floor currency of their choosing
    b. Bid adapters must be allowed to specify the floor currency for their endpoint
    c. Bidders that require a bid adjustment would have their floor adjusted by original_floor / bidadjustment factor. The intent here is that the bidder's floor needs to be higher so that once their bid is adjusted down it's still above the floor. e.g. say a given bidder's floor was $2, but the publisher lowers their bid by 0.90. The intent is to raise the floor for this bidder to $2.22 so that once their bid is adjusted it's still above the floor

7. Floor rules should be insulated from other features. i.e. adapters or modules should not be able to detect a larger rule set than necessary
8. It should be possible to ease into dynamic server-side floor data, supporting analytics that verifies how dynamic floor rules compare to static floor rules.

3. Floors Feature Integration Flows

3.1. Prebid.js - Scenario 1: Client-enforced Floors

In this baseline scenario, PBJS will retrieve the floors, passing the rules to PBS. PBJS can optionally signal to enforce floors. For this flow, PBS-based fetching of floors has been disabled (or no URL provided), otherwise fetch would take precedence.



Steps for flooring:

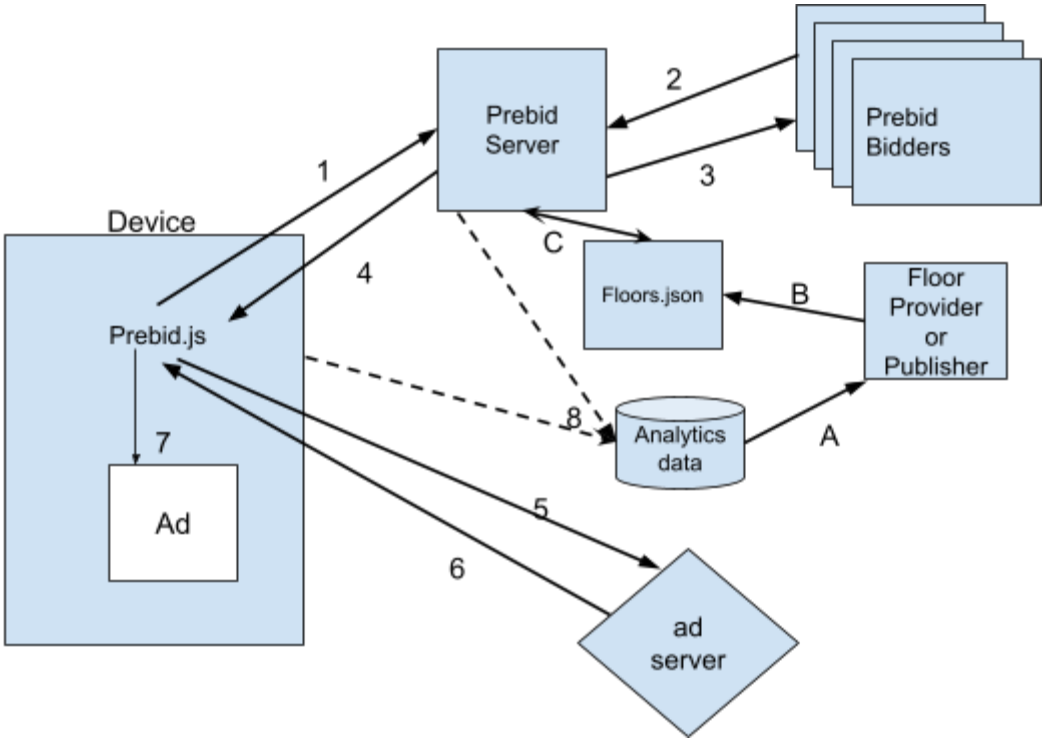
1. PBJS fetches a floor from Floor Provider's endpoint
2. PBJS will make an OpenRTB call to PBS for an ad
3. PBS call bidders
    a. Prebid Server will expose floor data to all bid adapters to read
4. Bidders will respond with bids
5. PBS will pass bids back to the caller
6. pbjs calls GAM with all eligible bids
7. GAM triggers the Prebid line item returning the PUC
8. PUC renders ad
9. PBJS sends all render data to data store to the floor provider to process

Offline Floor Data Sync:

A. Floor Provider reads from its data source historical bid data
B. Floor Provider pushes files to CDN

3.2. Prebid.js - Scenario 2: Server-enforced Floors

In this flow, PBJS will use PBS for rules fetching, signaling and enforcement(optional). Additionally, PBS will be the only demand source for PBJS for visual representation purposes.



Steps for flooring:

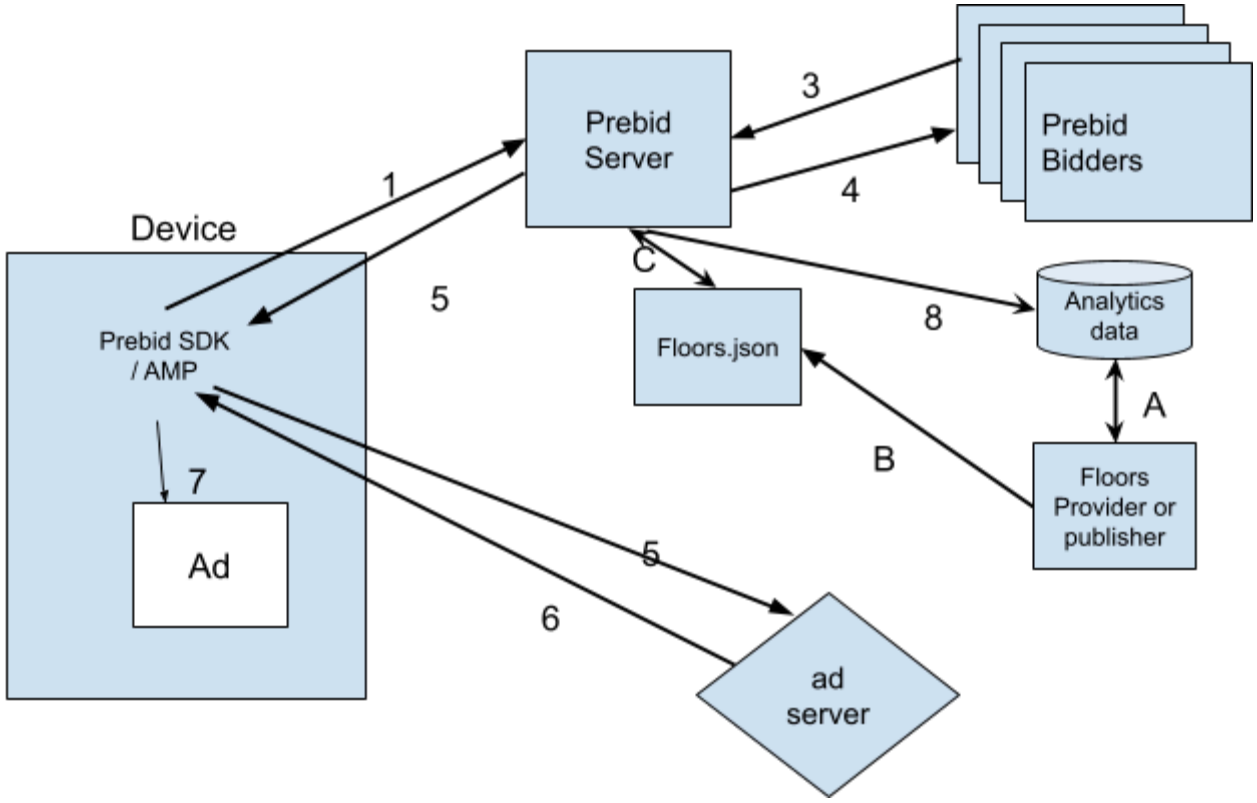
1. PBJS will make an OpenRTB call to PBS for an ad
2. PBS calls bidders
    a. Prebid Server will expose floor data to all bid adapters to read
3. Bidders will respond with bids
4. PBS will pass all bids exceeding the floor (enforcement is optional) back to the caller
5. PBJS calls GAM with all eligible bids
6. GAM triggers the Prebid line item returning the PUC
7. PUC or video player render ad
8. Optionally, PBJS or PBS sends render data to the floor provider. Some implementations may not require detailed analytics.

Offline Floor Data Sync:

A. Floor Provider reads from its data source historical bid data
B. Floor Provider prepares floors data
C. Prebid Server periodically syncs floors data

3.3. Prebid SDK / AMP

The last main scenario, and the one really driving the feature for Prebid Server, is the support of mobile apps and AMP:



Steps for flooring:

1. SDK makes an OpenRTB call to PBS for an ad
2. PBS call bidders
    a. Prebid Server will expose floor data to all bid adapters to read
3. Bidders will respond with bids
4. PBS will pass all bids exceeding the floor back to the caller
5. SDK calls GAM with all eligible bids
6. GAM triggers Prebid line item returns PUC
7. PUC renders ad after fetch from PBC (not show PBC fetch)
8. A PBS analytics adapter sends all auction data to Floor Provider analytics endpoint

Offline Floor Data Sync:

A. Floor Provider reads from its data source historical bid data
B. Floor Provider pushes files to CDN
C. Prebid Server syncs floors data

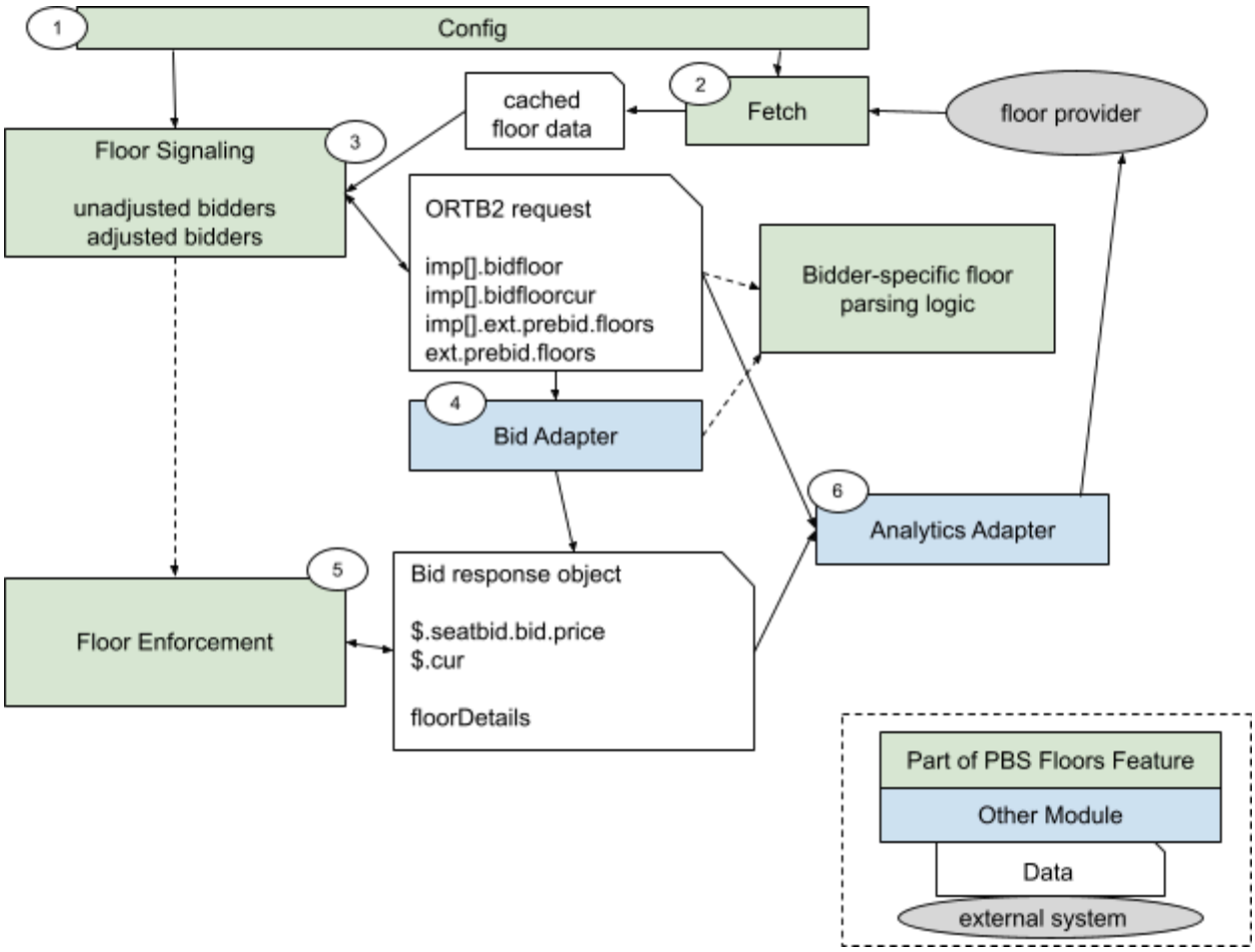
AMP does not support any flooring params. These would all be in the stored request.

4. Functional Requirements

These are the general functions of the Floors feature:

1. **Floor Configurations**: Publisher works with Prebid Server host provider to configure and enable floors for their requests.
    a. Publisher may wish to set floors in a storedRequest, in the ortb2 request, through a Floor Provider or all of these
2. **Fetch Floor Data**: If a Floor Provider is configured, the feature should periodically fetch floor rules files on an account by account basis to be used for floor signaling and enforcement
3. **Floor Signaling**: The feature should evaluate eligible rules for each auction and process rules to determine the appropriate floor to signal to bid adapters
    a. It's possible that bidders get unique floors due to bidCpmAdjustment
4. **Bid Adapter Support** - most adapters will just use the imp.bidfloor determined by the feature. Advanced bid adapters may utilize a 'getFloor' function that can process more advanced flooring rules involving mediaType and size.
5. **Enforcement**: The feature will process each bid to determine if each available bid meets or exceeds the floor for a given auction. If enforcement is turned on, bids with a CPM less than that floor will be rejected. Note that Deals are a special case and have their own flag for enforcement.
6. **Analytics** - pass auction floor information through to the analytics adapters.

This diagram shows the general relationship between the various parts of the system:



4.1. Floor Configurations

This section describes the settings and configurations for the Floors feature.

4.1.1. Global and Account Config

As a general rule of thumb, auction-level flags (provided in the bid stream or set in stored requests) should override **account** configurations, and **global** PBS config is generally used as a default.

All fields in the following table are both global and account level. The idea is that the host company will define a default for every field but that account-level configuration will override as needed.

No.	Config Field	Overall Default	Scope	Definition
1	enabled	true	init. dynamic	Master switch for turning off the floors feature for this account.
2	enforce-floors-rate (should)	100	init, dynamic	Percent chance that PBS should suppress any bids below the matched floor from entering the auction when true. This behavior is dependent upon the choice made for 'skipRate'. i.e. if floors are to be skipped for an auction, enforce-floors-rate doesn't matter.
3	adjust-for-bid-adjustment (must)	true	init, dynamic	Adjust floor passed to Bid Adapters if bid adjustment is passed to PBS and flag set is not set to false. Flag comes in via the top level request
4	enforce-deal-floors (should)	false	init, dynamic	Signals to PBS to suppress deal bids that are below the matched floor. This would only apply if ext.prebid.floors.enforcement.enforcePBS is true or not specified.
5	use-dynamic-data (should)	true	init, dynamic	Tells the signalling code to ignore dynamic fetched data. Could be used when a problem is found or a publisher changes their floor process.
6	fetch.enabled (must)	false	init	Signals to PBS to fetch and cache floors using the fetch URL. Cannot be true without fetch.url also provided.
7	fetch.url (must)	-	init	String containing the URL for PBS to fetch and cache rules files. (this probably doesn't make sense as a host-level

				config field, but but every other field does, so this is supported as well.)
8	fetch.timeout-ms (must)	3000	init	Global timeout for the feature to fetch new rules file, which is done outside of the auction.
9	fetch.max-file-size-kb (should)	100	init	Maximum file size for fetch rules. If the value is 0, it means no limit.
10	fetch.max-rules (should)	1000	init	Maximum fetch rule count per model group. If max-rules is 0, it means there's no maximum.
11	fetch.max-age-sec (must)	86400	init	The TTL/expiration of cached data.
12	fetch.period-sec (must)	3600	init	How often the feature should poll for refreshed data from the floor provider.
13	fetch.max-schema-dims (should)	0	init	Limit the number of concurrent schema dimensions a floor provider can include in a given set of floor rules. A value of 0 means no maximum.
14	max-rules (should)	100	init	Limits the number of rules processed when they come in on the request (or in stored requests). A value of 0 means no maximum.
15	max-schema-dims (should)	3	init	Limit the number of concurrent schema dimensions a request can include in a given set of floor rules. A value of 0 means no maximum.

Config requirements:

1. At initialization, the floors config should support default account-level parameters as noted above, with specific account overrides.
    a. The noted overall defaults should be supported.
2. Errors in the initialization config should cause PBS to fail startup with reasonable error messages. e.g. syntax error, unknown config, bad data type.

3. The only dynamic configuration for fetch that must be supported is the 'enabled' flag. It is expected that floor provider configuration will not change frequently.
    a. It would be nice for the system to be able to support fully dynamic setting of all fetch configuration.
4. Errors in dynamic account configuration should cause the feature to ignore the config entirely, fall back to the default account configuration, log a metric and log an error at N% sampling rate.
    a. Errors include: syntax, unknown values, data types
5. Value validations
    a. enforce-floors-rate >=0 and <= 100
    b. fetch.period-sec must be <= fetch.max-age-sec
    c. fetch.period-sec >= 300 (cannot poll more often than every 5 min)
    d. fetch.max-age-sec >= 600 (must allow data to be valid for at least 10 min)
    e. fetch.max-age-sec < MAXINT
    f. fetch.timeout-ms must be > 10 and < 10,000
    g. fetch.max-rules >= 0 and < MAXINT
    h. fetch.max-file-size-kb >= 0 and < MAXINT
    i. fetch.max-schema-dims >= 0 and < 20
    j. max-rules >= 0 and < MAXINT
    k. max-schema-dims >= 0 and < 20
6. Validation configuration - in order to better support the test environment, it would ideal if the system would allow for configuration of the following validation parameters at the global level:
    a. fetch.min-period-sec - can be set as low as 1 sec for testing purposes. Defaults to 300.
    b. fetch.min-max-age-sec: can be set as low as 1 sec for testing purposes. Defaults to 600. Note: this name seems confusing, but it's correct. It's the minimum (max age) that the floors provider is allowed to supply.
    c. fetch.min-timeout-ms: can be set as low as 1 sec for testing purposes. Defaults to 10.
    d. fetch.max-timeout-ms: Defaults to 10,000.

Here's a proposed initialization config:

```
settings:
  default-account-config: >
    {
      "auction": {
        "price-floors": {
          "enabled": true,
          "fetch": {
            "enabled": false,        // cannot be true as a default
                                     // (needs account-specific url)
            "timeout-ms": 5000,      // ms
            "max-rules": 0,          // no max
            "max-file-size-kb": 200, // KB
            "max-age-sec": 86400,    // 1 day
```

```
            "period-sec": 3600,      // 1 hour
            "max-schema-dims": 5    // no more than 5 dimensions
          },
          "enforce-floors-rate": 100,
          "adjust-for-bid-adjustment": true,
          "enforce-deal-floors": true,
          "use-dynamic-data": true,
          "max-rules": 100,
          "max-schema-dims": 3
        }
      }
    }

price-floors:
  enabled: true
```

It would be ideal for the floors feature to immediately fetch the defined URLs upon PBS startup, but this can come as a later improvement

Here's a proposed dynamic account-level config in DB or file:

```
{
  "auction": {
    "price-floors": {
      "use-dynamic-data": true,
      "enforce-floors-rate": 100,
      "adjust-for-bid-adjustment": true,
      "enforce-deal-floors": true
    }
  }
}
```

4.1.2. Floor Schema Syntax

The supported fields for the PBS floor schema is equivalent to the PBJS version 2 schema syntax that can be set in a resolved stored request or through a Floor Provider. See https://docs.prebid.org/dev-docs/modules/floors.html#schema-2 . Each key is stored in ext.prebid.floors.

Note: floors provider services are assumed to supply only the "data" portion of the schema.

For example:

```
pbjs.setConfig({
    floors: {
        enforcement: { ... },
        ...
        data: {
          "currency": "EUR",
          "skipRate": 20,
          "floorsSchemaVersion":2,
          ...
        }
    }
});
```

Would be sent to Prebid Server (or in a stored request) as:

```
ext.prebid.floors: {
        enforcement: { ... },
        ...
        data: {
          "currency": "EUR",
          "skipRate": 20,
          "floorsSchemaVersion":2,
          ...
        }

}
```

These are the assumptions this PRD makes about the Prebid Floors schema v2:

1. Floors providers **must** supply the modelGroup array
2. The following fields in 'data' may be used as defaults by modelGroups: currency and skipRate

4.2. Dynamic Fetch of Floor Data

This section will cover the behavior of the Floors feature fetch, used to retrieve Floor Provider rules files. See the configuration section above for config details.

1. Upon system startup, PBS should invoke the Floors feature with configuration that enables it to immediately begin filling the floors cache for each configured account.

2. The Floors feature must periodically fetch rules for each account when these conditions are true for an account:
    a. account auction.price-floors.enabled is true
    b. ext.prebid.floors.enabled is true if specified

c. fetch.enabled is true
d. fetch.url is configured

3. The fetch mechanism must support the HTTP GET method to retrieve floors.

4. Floor provider URLs are expected to return valid JSON in the PBS floors schema as defined in the "version 2 schema for PBJS". Note that floor provider responses are always considered to be under the 'data' object as noted in req 10 below. https://docs.prebid.org/dev-docs/modules/floors.html#schema-2
    a. The fetch should be made every **fetch.period-sec**, even if the past fetch failed.
    b. The JSON response must pass all validations.
        i. Validations
            1. HTTP request succeeds with HTTP code 200
            2. valid JSON
            3. at least one model
            4. at least one rule
            5. Overall size of the file is less than **fetch.max-file-size-kb**
            6. No modelGroup should have more than **fetch.max-rules**
            7. Fetch request exceeds **fetch.timeout-ms**
            8. If any modelGroup[].modelWeight field is present, then it must be an integer greater than or equal 1 and less than or equal to 100.
            9. If any skipRate field is present, then it must be an integer greater than or equal 0 and less than or equal to 100
            10. If any modelGroup[].default field is present, then it must be a float greater than or equal to 0.
            11. If the floorMin field is present, then it must be a float greater than or equal 0.
            12. No modelGroup should have more schema dimensions than defined in **fetch.max-schema-dims**.
            13. If useFetchDataRate is present, it must be an integer between 0 and 100 inclusive. The default value is 100.
        ii. Validation exceptions
            1. unknown attributes will be ignored
            2. duplicate attributes will use the second value
        iii. The feature must not replace a previously good schema in the cache, instead rejecting the invalid update.
        iv. A metric should be logged: price-floors.fetch.failure
        v. An error log entry should be made that includes the fetch.url and a general description of the validation issue.
            1. The intention is that the Prebid Server host company can set up operational alerts so they can work with the floors provider to resolve the issue.

5. The Floors feature must cache successful and valid fetched results to be used for fast retrieval at auction run-time
    a. Updating cache data must be done atomically in a thread-safe way since requests may be consuming previously cached data.
    b. PBS should cache each account's rules file up to the period defined by the max-age HTTP header of the floor provider response JSON.
        i. Validations on the max-age header: must be greater than fetch.min-max-age-sec. Must be greater than fetch.period-sec. Must be less than MAXINT.
        ii. If the header fails the validation, ignore it.
        iii. Note that the cache-control header may contain other controls, e.g. "no-cache, no-store, max-age=0, must-revalidate". The only entry in the header relevant to floors is max-age. All other entries are ignored.
    c. If no max-age HTTP header is set, use **fetch.max-age-sec** defined by account config or 86400 sec as an overall default.
        i. **fetch.max-age-sec** config is in seconds. It must be a positive integer greater than 0.
            1. If it's invalid in startup config, PBS should fail to start with a reasonable warning.
            2. If it's invalid in account config, log a warning and fall back to the 24-hour overall default
    d. Once max-age is reached, the system may clear that cache entry in a thread-safe way.
        i. An error log entry should be made that includes fetch.url and a general description of the validation issue.
            1. The intention is that the Prebid Server host company can set up operational alerts so they can work with the floors provider to resolve the issue.
        ii. This requirements document does not define whether the cache aging happens at read-time or in a separate thread.

6. Requirements for the **fetch.enabled** config:
    a. The fetch.enabled flag is a host provider configuration. It allows the host to temporarily turn off the periodic floor data fetch without losing the rest of the data like the fetch.url.
    b. Even if the fetch.enabled flag is toggled to false, the feature should continue to hold onto previously fetched rules files until they expire.
    c. If the fetch.enabled flag is toggled to true but no fetch.url is available, it's an error.

7. Requirements for the **fetch.url** config
    a. Only one fetch.url must be configured at a time for each account
    b. Each account must support its own fetch.url
    c. No specific URL validation is required. If the HTTP request fails for any reason, see the requirement above about validation.

    d. It's fine for the system to attempt to retrieve the bad URL every fetch.period.

8. Requirements for the **fetch.max-file-size-kb** config
    a. A value of 0 is possible, which means no maximum size.

9. Requirements for the **fetch.max-rules** config
    a. Rule count should be the max count of rules for a modelGroup. No modelGroup should have more than maxFetchRules. This is to protect the host provider on modelGroups being too large to store in memory.
    b. If a fetched rule exceeds the maxFetchRules count, do not cache rules and log a warning
    c. The default maxFetchRules value is 0, which indicates there's no max rule count.

10. When data is retrieved from a dynamic fetch, it must be assumed to be entirely under the 'data' element of Schema 2. As described in the signaling section below, fetch data will be merged into the floors request config under 'data' before validating. The idea is that the request/storedrequest will define some pieces of the overall floor config (e.g. floorMin, enabled) and the floor provider supplies the actual floor rules under the 'data' section.

11. Requirements for the fetch.max-schema-dims config:
    a. No modelGroup can have more than this number of dimensions. This is to protect the host provider on modelGroups being too large to store in memory.
    b. If a fetched file exceeds this count, do not cache the data set, and log a warning
    c. The default fetch.max-schema-dims value is 0, which indicates there's no max rule count.

4.3. Floor Signaling

This section will cover the inbound request and bid adapter flow, including how the Floors feature should handle the ingestion of rules to be used for signaling to bid adapters as well the recording of inbound rule results to the bid request object for analytics.

At a high level, this activity takes place after the auction ortb has been resolved:

- make sure floors are enabled for Prebid Server, the account, and the request. If any of them are disabled, skip signaling.
- determine which floor data to use
- determine which floor model to use
- overwrite the request **ext.prebid.floors** with the specific source and model being used so downstream entities all use the same data and model
- process floor rules
- update imp[].bidfloor/cur with the appropriate value

- update imp[].ext.prebid.floors with impression-specific values: the floorRule and floorRuleValue used to set the imp.bidfloor
- for adjusted bidders, update imp[].bidfloor/cur with the appropriate value

Requirements

1. Determine whether the floors feature is **enabled** for Prebid Server, account, and request. If not, no signaling is done. Verify that both account enabled is true and ext.prebid.floors.enabled is not false.
2. The Floors feature must determine at runtime the location of floor data to use, either resolved in the bid request to PBS or via fetch from a floors provider using the following priority: dynamic data from fetch (after considering useFetchDataRate) > ortb2 request (ext.prebid.floors) > imp.bidfloor > no floor as described below
    a. Dynamically fetched floors should take first priority over floors resolved in the request to PBS.
        i. As noted in the dynamic fetch section above, data from floors providers is always considered part of the 'data' section of Schema 2.
        ii. If data.useFetchDataRate is supplied in the dynamic data, is an integer, and is less than 100, choose a random number from 0-100. If that number is greater or equal to data.useFetchDataRate, then skip the dynamically fetched floor data. Make sure ext.prebid.floors.location reflects that dynamic data is not being used. This supports a use case where a floors provider has been asked to prove their data is better than static data.
    b. If no valid fetched data is found or data.useFetchDataRate skipped dynamic data, the Floors feature should look for floors in the resolved request
        i. The feature will only see the results of the merging of **ext.prebid.floors**, so will not know whether this value came from the original request or a stored request.
        ii. PBS-core will use ext.prebid.floors from the original request over any value from the stored request.
        iii. Validation on ext.prebid.floors should be done the same as 4.2.3.b for dynamic fetches.
            1. If an error is found during validation, and 'test' mode is off, place an error message in the response noting "Error in request floors data: Invalid modelWeight."
            2. Also, log validation error at N%.
            3. Floor processing can proceed with any supplied imp.bidfloor.
    c. If neither valid fetched floors or ext.prebid.floors exist, the feature does nothing for signaling or enforcement -- just pass "location: noData" to analytics adapters.

3. Once the location of which floor data to use is determined, the Floors feature should evaluate the skipRate from that floor data set. skipRate determines if the Floors feature should signal and enforce floors.
    a. skipRate can be declared within floors.data.modelGroups object
        i. If skipRate is supplied in both the floors.data object and within the floors.data.modelGroups array, the skipRate configuration within the floors.data.modelGroups array prevails
    b. skipRate must be an integer whose values should be between 0 and 100 inclusive
    c. skipRate should default to 0 if no value is supplied
    d. If a valid skipRate is supplied, the Floors feature should randomly skip floor usage by the rate provided in the derived skipRate field
        i. all imp objects in the auction are treated the same. i.e. the system should not pick different random numbers for different imps.
        ii. If floors are to be skipped:
            1. imp.bidfloor and imp.bidfloorcur will be left as they were originally
            2. The Floors feature should still signal to Analytics providers with data from the auction. See the analytics section below.
            3. No enforcement of bid responses takes place.

4. Assuming floors are active, the Floors feature must determine which model should be used based on the weight supplied in the ruleset:
    a. A single model is selected and applied to the whole auction (e.g. all imp objects)
    b. The floor provider is expected to provide model weights as a positive integer value
        i. If no modelWeight is provided, assume the weight is 1.
        ii. If specified, modelWeight must be an integer and > 0 and <= 100
            1. If specified and invalid, the entire ruleset should be rejected as noted above.
        iii. The system should divide the weights proportionally when choosing which model to use.

5. After all the previous conditions are met (floors are not disabled, floor location determined with valid data and skipRate is false) the Floors feature should remove non-selected models before updating ext.prebid.floors in the resolved bid request object.

6. The system must be designed so that newly fetched floor data does not overwrite the floor data resolved for an auction in-progress.

7. Field exceptions. The following fields from the original request (or stored request) must take precedence over any data from a dynamic fetch:
    a. floorMin - this is the publisher control to make sure that the floor provider is within acceptable bounds.
    b. floorMinCur

8. Schema version validation. The feature should validate that if defined, floorsSchemaVersion is 2. This field may be missing, and if so, it's assumed that it will conform to schema 2. Unknown fields may be ignored.

9. Analytics adapters and other downstream entities depend on the signaling component to set the following data:
    a. **ext.prebid.floors.data.modelGroups[0]** - this is the only entry of the original modelGroups array that should be seen by downstream code. It's the model group chosen based on weight.
    b. **ext.prebid.floors.floorProvider** - if ext.prebid.floors.data.floorProvider is set, copy it here. Otherwise, leave it to the original value.
    c. **ext.prebid.floors.skiprate** - this is set from the first available of:
        i. ext.prebid.floors.data.modelGroups[CHOSEN].skiprate
        ii. ext.prebid.floors.data.skiprate
        iii. otherwise leave it as the original value
    d. **ext.prebid.floors.floorMin** - not set by signaling. Takes whatever value comes in on the original floors data
    e. **imp[].ext.prebid.floors.floorMin** - not set by signaling. Takes whatever value comes in on the original request
    f. **ext.prebid.floors.data.modelTimestamp** - not set by signaling. Takes whatever value comes in on the original floors data
    g. **ext.prebid.floors.enabled** - boolean set to whether the entire floors feature (signaling and enforcement) is enabled for this request
    h. **ext.prebid.floors.fetchStatus (*)** - The last fetch status for this account. Valid values are: 'none' (when dynamic fetch is not enabled for the account), ‘success’ (when fetch returned an http 200 status), ‘timeout’ (when fetch results not returned before either auction delay or prebid timeout) ‘error’ (any http status other than 200 or other error condition), or 'inprogress' (when no valid fetch data is present but the request is outstanding)
    i. **ext.prebid.floors.skipped (*)** - Whether the skipRate resolved to be true or false
    j. **ext.prebid.floors.location (*)** - Where the module derived the rule set. Values are one of 'request’, ‘fetch’ or ‘noData’. If the module code is invoked and no floors object is able to be found (either by error or other condition) the floorsModule will set location to ‘noData’.

Note that the fields flagged by a (*) are output fields only: they are intended for use by analytics adapters and other downstream modules. They are not defined as Schema v2 input fields.

4.3.1. Schema Processing

4.3.1.1. Rules Requirements

1. Each key in data.modelGroups[].values is considered a rule within a set of one or more modelGroups, where the key is a set of targetable attributes and the value is the CPM floor
    a. Keys must be string values, otherwise reject the rule.
    b. CPM must be a float value, otherwise reject the rule.
    c. Rule fields are case insensitive
    d. In both rule selection and enforcement, all fields and values should be treated as lower case

2. The Floors feature should split each rule using the applied delimiter (default to "|" if none supplied), to match 1:1 for each data.modelGroups[].schema.fields string supplied in the order provided in the fields object
    a. If the total number of split entries in a given rule does not match the number of fields, ignore the rule and log a warning
    b. The request-level **max-rules** config cannot come in on the request - it must be specified as part of the account config, i.e. ignore ext.prebid.floors.max-rules. If there are more rules than allowed, stop processing and emit a warning to the debug output. This helps prevent host companies from dealing with accidental or malicious rule counts. A value of 0 indicates no limit.
    c. The request-level **max-schema-dims** config cannot come in on the request - it must be specified as part of the account config, i.e. ignore ext.prebid.floors.max-schema-dims. If there are more dimensions than allowed, stop processing and emit a warning to the debug output. This helps prevent host companies from dealing with accidental or malicious floors config. A value of 0 indicates no limit.
    d. A given rule can have one or more attributes within values that can be a "*" to signify any (or catch-all) for that specific field

3. The Floors feature must support a pre-defined set of schema fields defined in the data.modelGroups[].schema.fields object that correspond to the data.modelGroups[].values objects. Below are the fields and their corresponding values:

Rules Dimensions Table

Dimension	Type	Example	How the Floors Feature and the getFloor function Resolve
siteDomain	string	level4.level3.example.com	Compare to site.domain or app.domain

pubDomain	string	example.com	Compare to site.publisher.domain or app.publisher.domain
domain	string	example.com	Compare to (site.domain and site.publisher.domain) or (app.domain and app.publisher.domain). If any of them match this part of the rule matches.
bundle	string	org.prebid.drprebid	app.bundle
channel	string	pbjs, amp	ext.prebid.channel.name
mediaType	string	banner	set to "*": if more than one of these: imp.banner, imp.video, imp.native, imp.audio banner: true if imp.banner exists video-outstream: true if imp.video exists and imp.video.placement is not 1 video-instream: true if imp.video exists and imp.video placement exists and is 1 video: an alias for video-instream native: true if imp.native exists audio: true if imp.audio exists
size	string	300x250	if mediaType banner and only one size exists in imp.banner.format, then match against imp.banner.format[0].w and imp.banner.format[0].h if mediaType banner and there's no imp.banner.format, then use imp.banner.w and imp.banner.h if mediaType video, use imp.video.w and imp.video.h Otherwise size only matches the * condition
gptSlot	string	/1111/adslot	if imp.ext.data.adserver.name=="gam" then compare against imp.ext.data.adserver.adslot otherwise also compare against imp.ext.data.pbadslot

~~pbAdSlot~~	~~string~~	~~/1111/homepage#div1~~	~~imp.ext.data.pbadslot~~
adUnitCode	string	/1111/homepage#div1	1) imp.ext.gpid 2) imp.tagid 3) imp.ext.data.pbadslot 4) imp.ext.prebid.storedrequest.id
country	string	usa	compare against device.geo.country (ISO-3166-1-alpha-3)
deviceType	string	desktop, phone, tablet	if device.ua is not present, only rules specifying a wildcard deviceType will match. In other words, there's no default value unless device.ua exists. if device.ua is present, resolve deviceType to: 'phone' if UA matches one of these patterns: "Phone", "iPhone", "Android.*Mobile", "Mobile.*Android" else 'tablet' if UA matches one of these: "tablet", "iPad", "Windows NT.*touch", "touch.*Windows NT", "Android" else 'desktop'

The last column in this table is used in two places:

1. When the feature is determining the imp.bidfloor for each imp in the request, it could just call getFloor() which uses these values as defaults.
2. When a bid adapter calls getFloor() without specifying a value, the function will use these values as defaults

4. If a rules file contains an unrecognized dimension in the schema, the rules file should be considered invalid.

4.3.1.2. Rule Selection Algorithm

See https://docs.prebid.org/dev-docs/modules/floors.html#rule-selection-process

1. After a rule (and thus floor value) is selected for each imp, the Floors feature must validate the chosen **floorRuleValue** against any provided floorMin:
    a. Look for a floorMin. First check for imp[].ext.prebid.floors.floorMin. If that doesn't exist, check ext.prebid.floors.floorMin.
    b. If a floorMin was found, next, find the floorMin currency: First check for imp[].ext.prebid.floors.floorMinCur. If that doesn't exist, check for

ext.prebid.floors.floorMinCur. If that doesn't exist, check the floors data for data.currency or data.modelGroups[].currency
        i. If both imp[].ext.prebid.floors.floorMinCur and ext.prebid.floors.floorMinCur exist and they're different, emit a warning in debug mode.
    c. If the floorMinCur is different from the floor currency, convert the floorMin value.
    d. If a currency conversion is required but not available, the system should not override imp.bidfloor. It should log a warning in debug mode and to the error log at N% sampling.
    e. If the floorRuleValue is less than the floorMin, then floorValue is set to the converted floorMin. Else floorValue is set to floorRuleValue.
    f. Floor values should be rounded to 4 digits of precision.
2. Overwrite the relevant imp.bidfloor with floorValue and imp.bidfloorcur with the floorProvider's currency. Note: the original imp.bidfloor is a default, not a min. So if a rule matched, it might overwrite the original imp.bidfloor value with a lower value.
3. If no rule matched, the original imp.bidfloor and imp.bidfloorcur must remain untouched.
4. There's no floorMin check if no floorRuleValue is chosen. The idea behind floorMin is that it's a fence for dynamic floors. If the pub sends in a default imp.bidfloor, we can assume it's above the floorMin

4.3.1.3. Bidder Floor Adjustment

If ext.prebid.bidadjustmentfactors is specified for any bidder, the feature must be able to adjust the floor as seen by that bidder. There are two places where this needs to be accommodated:

- Updating the bidder-specific ORTB imp.bidfloor for named bidders
- Adjusting the output of the getFloor() function for named bidders

Requirements

1. If the ext.prebid.bidadjustmentfactors array contains the current bidder, the feature must divide the floor by the adjustment value. For example, if the auction-wide floor is 1.00 USD and bidderA has an adjustment factor of 0.85. the system should update imp.bidfloor as seen by that adapter to 1.1765.
    a. If mediatype level adjustments are defined and the impression defines more than one mediatype, choose the most aggressive adjustment, i.e. the lowest one.
2. Likewise, if ext.prebid.bidadjustments contains the current bidder, the feature must get the relevant adjustment array and can walk the adjustments backwards.
    a. If mediatype level adjustments are defined and the impression defines more than one mediatype, choose the most aggressive adjustment, i.e. the lowest one.
3. The floor function should be able to recognize which bidder is calling it and similarly adjust the floor provided to that bidder.

4.4. Bid Adapter

See section 7.3, the bid adapter interface.

4.5. Enforcement

Note: The floor enforcement options supported by PBS are the same as defined in Schema 2 for Prebid.js with one exception: PBS adds the "enforceRate" option, which corresponds to the 'enforce-floors-rate' configuration option.

1. The price floor must be enforced before pricegranularity rounding occurs but after regular bidresponse-to-requested-currency conversion.

2. The system must determine whether to enforce floors by considering all of these factors. If all of them are true, it should enforce the bid:
    a. If the account config **enabled** flag is true
    b. If request ext.prebid.floors.enabled is defined and true. If not defined by the request, the default is true.
    c. If **ext.prebid.floors.enforcement.enforcePBS** is not defined or if **ext.prebid.floors.enforcement.enforcePBS** is defined and true
        i. This field is used as a flag from Prebid.js telling PBS that the client-side floors module is in effect.
    d. If the bid response contains a seatbid.bid.dealid and (**enforce-deal-floors** config is set to true and **ext.prebid.floors.enforcement.floorDeals** is true)
    e. If both account.auction.price-floors.enforce-floors-rate and request.ext.prebid.floors.enforcement.enforceRate are satisfied

3. If enforcement is to take place, the Floors feature must compare the bid against the appropriate floor for the specific bid after normalizing for currency.
    a. If currency cannot be normalized, no floor enforcement can take place. Log a warning in debug mode and to the error log at N% sampling.
    b. Bid adapters may have utilized floors more sophisticated than simply imp.bidfloor – they may have called the getFloors() function to get a specific rule combination. The enforcement component must be able to enforce against the floors ruleset, not just the possibly over-simplified imp.bidfloor.
    c. If bid is below the floor
        i. mark the bid as bidRejected and suppress it from auction.
            1. It should not be cached in PBC or present in the client response.
        ii. create a record for analytics adapters to be aware of this rejection and reason.
        iii. Log a warning in debug mode and to the error log at N% sampling. The system should use a specific error code for this log entry.
    d. If the bid meets or exceeds the floor, it is not modified

4. If multi-bid is enabled, each bid should be enforced independently.

5. Requirements for **enforce-floors-rate** config:
    d. When specified and valid, for each auction request that is not skipped by the skipRate setting, the system should pick a random number between 0 and 100. If the number chosen is less than the enforce-floors-rate, set ext.prebid.floors.enforcement.enforcePBS=true. Else set ext.prebid.floors.enforcement.enforcePBS=false.
    e. Note: imp.bidfloor on the original request is a default passed through to bidders even if the floors feature is skipped. In this scenario, no enforcement happens if either skipped=true OR if (skipped=false and enforce-floors-rate turns out false)

6. If enabled per above rules, enforcement should happen even if the PBS feature did not add any signaling to the bidfloor/bidfloorcur already present in the request.

6. Prebid.js Server-to-Server Adapter

1. The PBJS pbsBidAdapter should send floor configuration data to PBS in **ext.prebid.floors**.
2. If there are multiple models and PBJS has chosen a particular model, only that model should be sent to PBS.
3. It should also resolve the client-side match into imp.bidfloor/cur as a default.
4. There should be an s2sConfig config value that allows the publisher to turn off floors data, as it can be large and if PBS doesn't support floors, it would be wasted bandwidth.

If the PBS host company has set up the Floors feature it will be able to parse the data. If not, the data will be ignored and imp.bidfloor will be passed through to the adapters.

7. Interfaces

7.1. Stored Request Interface

Stored Requests can contain ext.prebid.floors and imp[].bidfloor{cur}

7.1.1. Example Configuration 1

Floor data is configured on the top-level stored-request. A simple one modelGroup configuration:

```
{
   "ext": {
```

```
      "prebid": {
         "floors": {
            "data": {
               "currency": "USD",
               "modelGroups": [
                  {
                     "schema": {
                        "fields": [
                           "gptSlot",
                           "mediaType"
                        ]
                     },
                     "values": {
                        "*|*": 0.1,
                        "/1111/homepage/left-nav|banner": 0.9,
                        "/1111/homepage/top-rect|banner": 0.5,
                        "/1111/homepage/top-rect|video": 5,
                        "/1111/homepage/top-rect|*": 5,
                        "/1111/tech/left-nav|banner": 1.5
                     },
                     "default": 1.0
                  }
               ]
            }
         },
         "targeting": {
            "includebidderkeys": true,
            "includewinners": true,
            "pricegranularity": "med"
         }
      }
   }
}
```

7.2. Floors Provider Interface

The data format is a subset of what is supported for Prebid.js. Floor data providers endpoints must return JSON data, and all of that data is assumed to be under the "data" field of the main floors schema. Essentially, floors providers have a different schema.

Example response from the fetch.url:

```
{
   "currency": "USD",
   "modelGroups": [
      {
         "modelVersion": "new model 1.0",
         "modelWeight": 50,
         "schema": {
```

```
            "fields": [
               "country",
               "bundle",
               "mediaType",
               "size"
            ]
         },
         "values": {
            "*|*|*|*": 0.8,
            "*|org.prebid.mobile.demoapp|banner|*": 0.75,
            "*|org.prebid.mobile.demoapp|banner|300x50": 0.95,
            "GBR|*|video|480x600": 1.5,
            "usa|*|banner|*": 0.9,
            "usa|*|banner|300x250": 0.8,
            "usa|*|banner|300x600": 1.2
         }
      },
      {
         "modelVersion": "new model 1.0",
         "modelWeight": 50,
         "schema": {
            "fields": [
               "gptSlot",
               "mediaType"
            ]
         },
         "values": {
            "*|*": 0.1,
            "/1111/homepage/top-rect|banner": 0.8,
            "/1111/homepage/top-rect|video": 1.2,
            "/1111/homepage/top-rect|*": 1.2,
            "/1111/tech/left-nav|banner": 1.5
         }
      }
   ],
   "skipRate": 5
}
```

7.3. Bid Adapter Interface

While OpenRTB has a clean method to express a bidfloor and bidfloorcurrency, they exist at the top level of the bidRequest object. There are several scenarios, including floors for multiple sizes and / or multi-format bid requests, where more than one price floor can be matched for a given auction.

To avoid ambiguity, Prebid.js exposes a 'getFloor()' function each bidder can utilize on every bidrequest object, media type, and size combination with the desired currency to derive the appropriate floor.

There are 4 scenarios that cover server-side bid adapters:

1. The most common scenario is expected to be bidders that just forward imp.bidfloor and imp.bidfloorcur through to their endpoints, not supporting advanced floors.
2. Some bid adapters may not support floors at all, completely ignoring imp.bidfloor.
3. A bid adapter may support special floor logic using getFloor(). Specific examples:
    a. if the adapter selects just one mediatype in a multi-format imp, it should call getFloor() with the utilized mediatype.
    b. if the adapter creates separate requests for each size in a multi-size imp, it should call getFloor() with the relevant size.
4. If the bidder endpoint supports Prebid-level floor granularity (i.e. multi-format and multi-size), the adapter can call getFloor() to parse the floor schema or it can translate the schema in its own way.

7.3.1. Assumptions

1. Bid adapters should not have access to floor configuration such as skip rate, enforce rate, etc.

7.3.2. The Floor Function

A floor function (aka "getFloor") enables Bid Adapters to retrieve a floor for each auction. Due to the complexity of the rule system, deriving the correct floor can be a difficult task. The function simplifies the retrieval process.

1. A method must be available to allow each Bid Adapter to query for special floors. Further requirements assume this will be implemented as a "getFloor()" function, but any other approach consistent with the requirements is acceptable.

2. If a bid adapter needs special floor details, it must call the floor function on each imp object to retrieve a floor

3. The floor function should take these parameters:
    a. **biddercode:** needed to handle bid adjustments
    b. **currency**: the three letter currency code for the output price
    c. **bidRequest or impId**: used to identify which imp object to operate on
    d. **mediaType**: the provided bidRequest/impID will be scanned to get the list of relevant mediaTypes.:
        i. banner

        ii. video
        iii. video-instream
        iv. video-outstream
        v. native
        vi. * // the default
    e. **size**: the provided bidRequest/impID will be scanned to get the list of relevant sizes
        i. Array of w, h (`[ w, h]`)
        ii. * // the default

4. BidRequest/impId is the only required parameter for the floor function, all other parameters are optional
    a. If BidRequest/impId is not provided, do not return a floor
    b. If only BidRequest/impId is provided, the result should be the same as imp.bidfloor that was already set by the feature.

5. MediaType or Size parameters that are not defined or are explicitly “*” will match to the rule whose field contains a * value

6. If no rule matches, use a 'default' floor, else, use no floor.

7. The floor function should parse the floors rules and return the following parameters:
    a. **currency**: the three letter currency code the returned floor is in
    b. **floor**: the CPM floor evaluated by the Floors feature
    c. **floorDetails**: a block of JSON that the bid adapter will attach to the bid response for use in analytics. It should contain:
        i. floorRule
        ii. floorRuleValue

8. The Floors system does not need to enforce that bid adapters are implemented correctly with respect to calling the floor function and attaching the floorDetails to the bid response -- that will be the responsibility of the code reviewers. It is expected that only a few bidders will make use of this advanced feature.

9. The floor function is expected to return only one floor per call.

10. The floor function should respond with a max CPM precision of 4 decimal points

11. The floor function should treat the OpenRTB imp.bidfloor as a floor minimum of any selected rule floor. Note that the imp.bidfloor at this point may not be the original request bidfloor - it may have been replaced by a rule value.
    a. The getFloor function should take the greater of imp.bidfloor and the selected floor data floor when signaling to bid adapters

12. If PBS has a bidAdjustment factor for this particular bidder (ext.prebid.bidadjustmentfactors), the floor function must adjust the floor by the inverse of the incoming bidAdjustmentFactor. **This is an important feature** - here's the scenario:
    a. Publisher has applied an adjustment factor on bidderA of 0.9.
    b. The floor is 1.10.
    c. If we allow bidderA to bid 1.00, their bid will get adjusted down to 0.90 and it will then get thrown out due to the floor.
    d. Therefore, the floors feature needs to apply the inverse of the adjustment and tells bidderA the floor is 1.00/0.9=1.11 so that when they do bid, those bids will be above the actual floor.

13. Similarly, if ext.prebid.bidadjustments exists, the floor function must adjust the bidder's floor by the inverse of the incoming bidAdjustments. For example:
    a. imp[].bidfloor: 0.98,
    b. imp[].bidfloorcur: "USD"
    c.
```
"bidadjustments": {
```
    d.
```
  "mediatype": {
```
    e.
```
    "banner": {
```
    f.
```
      "bidderA": {
```
    g.
```
        "*": [{
```
    h.
```
            "adjtype": "multiplier",
```
    i.
```
            "value": 0.8
```
    j.
```
          },{
```
    k.
```
            "adjtype": "cpm",
```
    l.
```
            "value": 0.18,
```
    m.
```
            "currency": "USD"
```
    n.
```
          }]
```
    o.
```
      }
```
    p.
```
    }
```
    q. The resulting call to the priceFloorResolver for banner should return (0.98 + 0.18) / 0.8 = 1.45

14. The floor function should perform any currency conversion if there's a currency difference between the rule floor and the requested currency.

15. The floor function must respect the 'skipped' status as chosen by the Floors Signaling component.

16. The floor function must check whether the floors feature is enabled for this request. i.e. the request may turn it off by setting ext.prebid.floors.enabled:false.

Example Format

```
getFloor({
     bidder: "bidderA",
    currency: string,   //default to currency specified in data
    impId: string       //Required, value consisting of the imp id to
associate to the appropriate imp object
    mediatType: string  //Required,one of “banner”, “video”, “native” or *
    size : [ w, h] OR ‘*’  //default size is “*”
});
```

The job of the floor function is:

1) Encapsulate full rules file by matching the auction floor rule set against attributes of the bidRequest object
    a) Bidders will be required to call getFloor for each mediaType, ad size and imp.id combination or once per mediaType per imp object if they don’t support floors per size
    b) getFloor will return a single price floor and floor currency specific for that imp, media type and size combination of the bidrequest object for that bidder context
2) Handle floor adjustments from the bidder ext.prebid.bidadjustmentfactors if relevant
3) Convert the currency into the desired currency using the currency conversion feature, otherwise, return specified currency in utilized data set
    a) If a currency conversion is not available, return to the caller with an error. They should not set any floor at all rather than set an incorrect floor.

7.3.2.1 Example Rules file

Below is an example of valid rules for a given auction to be used in the following getFloor() examples:

```
{
   "data": {
         "currency": "USD",
         "schema": {
            "fields": [ "gptSlot", "mediaType", "size"]
         },
         "values": {
            "/1111/homepage/top-rect|banner|300x250" : 0.60,
            "/1111/homepage/top-rect|banner|300x600" : 1.78,
            "/1111/homepage/top-rect|banner|*" : 1.10,
            "/1111/homepage/top-rect|video|480x600" : 3.20,
            "/1111/homepage/top-leaderboard|banner|728x90" : 1.50
         },
         "default": 0.75
```

```
    }
}
```

7.3.2.2. Example getFloor 1

getFloor for media type Banner for a bid request in the context of the gpt slot “/1111/homepage/top-rect” where the bid adapter does not support multiple sizes.

```
getFloor({
    bidder: "bidderA",
    bidRequest: object
    currency: 'USD',
    impId: “1”,
    mediatType: ‘banner’,
    size: "*"
});
```

Example response

```
{
    currency: 'USD',
    floor: 1.01,
    floorDetails: {
       floorRule: "/1111/homepage/top-rect|banner|*"
       floorRuleValue: 1.10,
       floorValue: 1.01,
       currency: 'USD'
    }
}
```

The bid adapter does two things with this response:

1. uses the 'floor' value in the call to their endpoint
2. attaches the 'floorDetails' object to the bidResponse object they create

7.3.2.2. Example getFloor 2

getFloor for media type Banner for a bid request in the context of the gpt slot “/1111/homepage/top-rect” with size of 300x600 where the bid adapter does support multiple sizes.

```
getFloor({
    bidder: "bidderA",
```

```
    currency: 'USD',
    impId: "1",
    mediaType: 'banner',
    size: [300, 600]
});
```

Response

```
{
    currency: 'USD',
    floor: 1.78,
    floorDetails: {
        floorRule: "/1111/homepage/top-rect|banner|300x600",
        floorRuleValue: 1.78,
        floorValue: 1.78,
        currency: 'USD'
    }
}
```

The bid adapter does two things with this response:

1. uses the 'floor' value in the call to their endpoint
2. attaches the 'floorDetails' object to the bidResponse object they create

7.4. Analytics Interface

Price Floors providers will rely heavily on the associated Prebid analytics feature in order to make the most informed price floor rule decisions. Because of this, the price floors feature needs to relay important information about the flooring and decisions made in the lifecycle of an auction.

The price floors feature will do this by leveraging the already existing implementation for Prebid analytics by exposing information onto the bidRequest, bidResponse, and possibly analytics tags objects. Thus, when an analytics adapter hooks into these Prebid events, it will be able to pick out the price floors data and pass it along to their servers.

7.4.1. bidRequest Object

Bid Requests objects contain all floors data. It may be used by bidder adapters and analytics adapters as needed.

Parameter	Type	Description

imp.bidfloor	float	The floor defined by the feature for bidders that just look here.
imp.bidfloorcur	string	Currency of the imp.bidfloor. If the floors feature is disabled, then this value is whatever was set on the original request. If enabled, then this value is whatever currency is defined in the chosen rule: the fetch file or the incoming floor data.
imp.ext.prebid.floors		Impression-level Floor values set by the Floors feature. This block will be empty if no valid rule set was found.
floorRule	string	The specific rule that set the imp.bidfloor. (e.g. "/1111/homepage|*". This will be empty when the skipRate feature fires.
floorRuleValue	float	The value of the rule. Could differ from imp.bidfloor based on bid adjustments or the floorMin setting. This will be empty when the skipRate feature fires.
floorValue	float	This is the final imp.bidfloor for the specific bidder. which is the max of (floorRuleValue, orig imp.bidfloor, ext.prebid.floors.floorMin, and imp[].ext.prebid.floors.floorMin) Note: the currency here is the same as imp.bidfloorcur.
floorMin	float	the lowest value floor the publisher will allow for this ad unit
floorMinCur	string	the currency in which the lowest floor is specified
ext.prebid.floors		Auction-level floors values. This block comes from the source of floors data: dynamic fetch, stored request, or original ortb2 request. It may be enhanced by the Floors feature.
enabled	boolean	Turn off the server-side floors feature. Defaults to 'true'. If the feature is disabled, all functionality in the feature should be turned off. This is intended only to be set by the request in emergency situations where the account config cannot be updated.
fetchStatus	string	The last fetch status for this account. Valid values are: 'none' (when dynamic fetch is not enabled for

		the account), ‘success’ (when fetch returned an http 200 status), ‘timeout’ (when fetch results not returned before either auction delay or prebid timeout) ‘error’ (any http status other than 200 or other error condition), or 'inprogress' (when no valid fetch data is present but the request is outstanding)
location	string	Where the module derived the rule set. Values are one of 'request’, ‘fetch’ or ‘noData’. If the module code is invoked and no floors object is able to be found (either by error or other condition) the floorsModule will set location to ‘noData’.
skipped	boolean	Whether the skipRate resolved to be true or false
floorMinCur	string	Defines the currency of the floorMin as defined by the publisher. Defaults to the currency of the floors provider (data.currency). Note: this is an addition to the PBS implementation and will eventually be added to the PBJS module.
ALL FIELDS IN SCHEMA 2		See https://docs.prebid.org/dev-docs/modules/floors.html#schema-2

There are several fields in the PBJS schema 2 that are NOT recognized by Prebid Server. These fields are ignored:

- endpoint.url
- enforcement.enforceJS

7.4.2. bidResponse Object

Most floor data is common across all bidders and bid responses for a given imp. There are several cases where bidder-response-level data is needed:

- when the Floors feature determines that it needs to remove a bid response for enforcement reasons
- when a bidder's floor was adjusted due to bidAdjustmentFactor
- when a bidder uses 'getFloor' logic to determine a custom floor

When a bid response is being processed it is important for analytics adapters to know the decision which was made and the context of the rule selection. Here is the floor data that needs to be attached to each bidResponse:

Parameter	Type	Description

bidAdjustment	boolean	Used to record if the bid floor was CPM adjusted based on the bidAdjustmentFactor provided in the bidRequest
floorCurrency	string	Currency of the floor matched. Only specified if the bidder calls getFloor.
floorRule	string	The matching rule for the given bidResponse. Only needed if the bidder calls getFloor().
floorRuleValue	float	Rule floor selected. This is to differentiate between the floor bound to the selected rule and the OpenRTB bidfloor (if available). Only needed if the bidder calls getFloor().
floorValue	float	The value of the floor enforced for this bidder. This will be the greater of the OpenRTB bidfloor and floorRuleValue. Only needed if the bidder calls getFloor or if the floor was adjusted due to bidCpmAdjustments.

It must be possible to determine how to communicate that a bid response has been rejected due to floors. If the existing PBS BidResponse object cannot be used, the system could make use of the Modularity system's Analytics Tags field.

7.5. Prebid SDK Interface

Given the static nature of apps, setting floors at the SDK level would be a difficult task. For this reason, floors for Prebid SDK traffic should be set using floor providers (fetch) or in stored requests.

7.6. AMP

Similar to Prebid SDK traffic, passing floors real-time for AMP traffic can be a challenge. Floors in this mode should be also set using floor providers (fetch) or in stored requests.

Change Log

Date	Change	Person
June 30, 2022	Added support for imp-level floorMin	bretg
July 12, 2022	Added adUnitCode targeting dimension	bretg

July 25, 2022	Updated imp-level floorMinCur default in 4.3.1.2.1	bretg
July 27, 2022	Refined imp-level floorMinCur default in 4.3.1.2.1	bretg
Aug 3, 2022	Added imp.tagid to the adUnitCode targeting dimension resolution	bretg
Sept 12, 2022	Made it explicit that skipped: true also means that enforcement doesn't happen.	bretg
Dec 9, 2022	Addressed clarification questions.	bretg
Mar 20, 2023	Added fetch.max-schema-dims and request-level max-rules and max-schema-dims	bretg
Mar 22, 2023	Added floorMin, floorMinCur to the analytics table	bretg
May 10, 2023	Added fetch.use-data-rate feature. Added description of why bidadjustments affect floors.	bretg
June 12, 2023	Dropped mention of the floorMinCur default from the field exception requirement	bretg
June 15, 2023	Changed fetch.use-data-rate to data.useFetchDataRate	bretg
Feb 20, 2025	Added the new bidadjustments feature, as distinct from the older bidadjustmentfactors feature.	bretg