Intro to Prebid Programmatic Guaranteed

Dec 11 2019

Overview

This memo outlines the components of the Prebid Programmatic Guaranteed (PG) system.

Programmatic Guaranteed has existed for several years as an ad server-based function. While anchoring PG in the Ad Server was helpful from a deployment expediency perspective, the lack of transparency and configurability in the ad server slowed down product momentum and made it harder for buyers and sellers to flexibly adjust their strategy and configurations for PG deals. The introduction of open-source, standards-based Programmatic Guaranteed, anchored in Prebid Sever, will help both buyers and sellers in several key ways: :

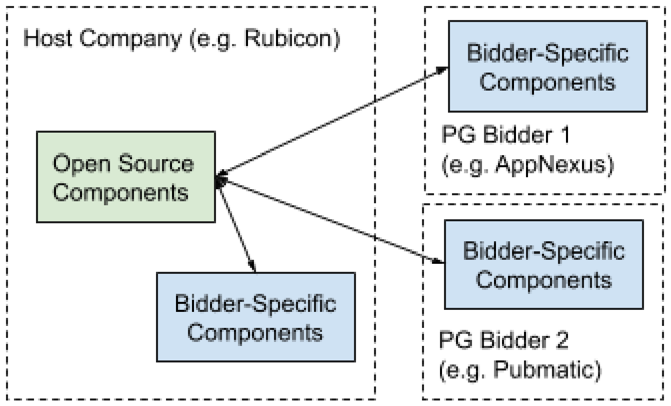
- It allows publishers to utilize preferred partners and data in the Programmatic infrastructure.
- It streamlines the traditional RFP and I/O process between buyers and sellers
- It separates the pacing, capping and forecasting functions from the ad server so publishers can more easily control and modify deals
- Enables a seamless, software-based negotiation process between buyers and sellers
- Ensures easier interoperability with a wider universe of buy-side platforms

There will be separate documents for those wishing to host their own PG system or integrate into an existing PG cluster.

The Framework

At a high level, the system has been designed so that any **host company** running Prebid Server (only the Java version for now) can integrate open source components into their existing UIs and data delivery systems.

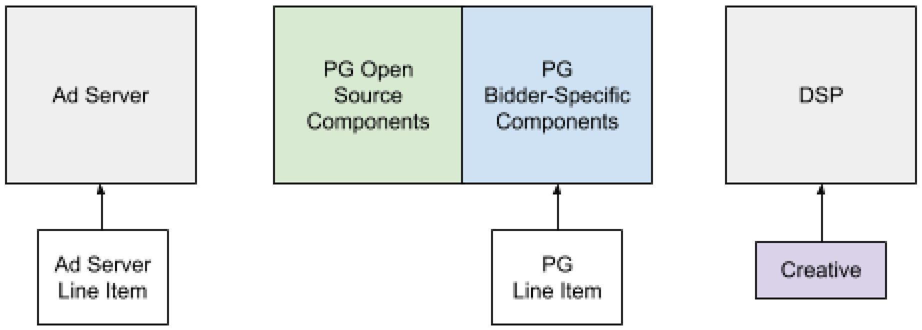
In addition, the host company can support other **PG bidders** connecting into their system to make their environment richer for publishers.


Host Company (e.g. Rubicon)
Open Source Components
Bidder-Specific Components
Bidder-Specific Components
PG Bidder 1 (e.g. AppNexus)
Bidder-Specific Components
PG Bidder 2 (e.g. Pubmatic)

The main idea is to encourage an ecosystem where programmatic vendors can compete on their strengths. Some companies may be excellent at hosting the technical infrastructure, while others may excel at usability, reporting, or delivery algorithms.

Setting Up a Line Item

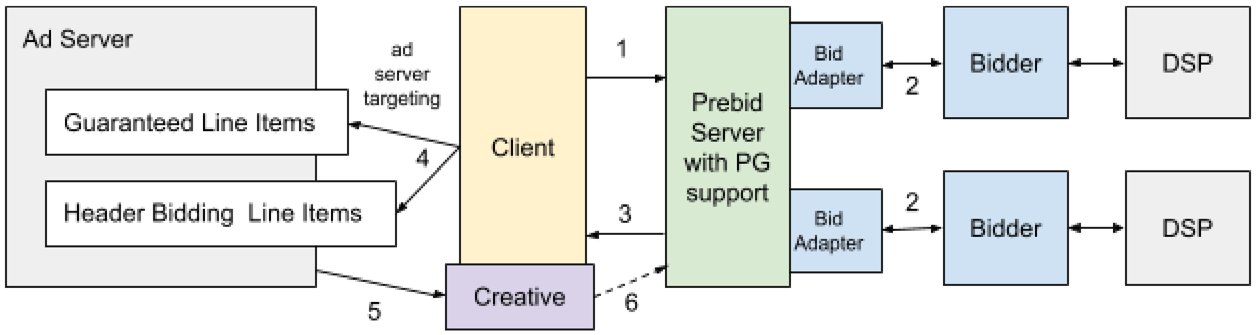
The data objects recognized by the PG system are Deals, Line Items, and Creatives. A **Deal** is a contract that may have multiple Line Items. A **Line Item** is the component of a Deal that has a start date, end date, target, goal, and other specific options. A **Creative** is the artwork that shows up in the browser, app, or video. The expected usage model is that a Line Item will be entered into both the PG system and the ad server, while the creative is entered into the DSP.



Line Item data entered into the PG system and the Ad Server may differ -- for example, the targeting information entered into the PG UI will be comprehensive: geographic, user, device, time, etc. While the targeting entered into the ad server will likely be simplified, just looking for the presence of a deal signal.

End to End

End-to-end, the system works largely like it does for regular open market header bidding:

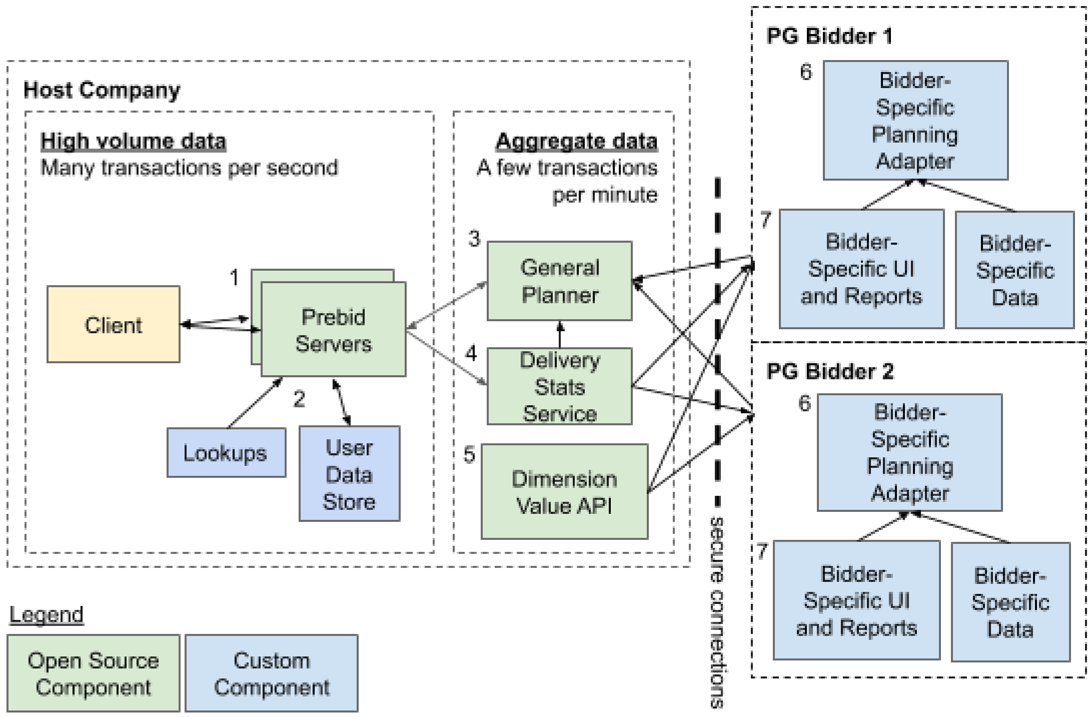


1) The client (browser or app) makes a request to Prebid Server using Prebid.js or Prebid SDK using standard Prebid.js AdUnits.

2) A multi-bidder auction takes place as usual, with special PG processing as appropriate. Bidder adapters that know how to handle PG line items forward the information to their endpoints.
3) Prebid Server returns PG line items as deals, possibly mixed with open market bids.
4) Targeting variables are returned to the ad server as usual, where ad server line items have been set up in advance. The best PG line item from each bidder is returned.
5) If one of these line items wins in the ad server, the Prebid Universal Creative is returned so the ad can be displayed.
6) If a PG line item won, the creative notifies Prebid Server of the win so it can adjust pacing as needed.

Detailed Architecture

Drilling deeper, this diagram shows the interaction between specific PG components:



Each component has a specific role within the lifecycle of a PG line item:

1) **Prebid Server** (PBS) needs to make decisions within milliseconds. Prebid.js or Prebid SDK create PBS-flavored OpenRTB requests, to which Prebid Server has to respond as quickly as possible with relevant PG and open market bids. These are the tasks it performs:
    a) For requests that may have PG line items, perform lookups against various data sources to enhance the request with geographic, device, and user info.

    b) Quickly determine which line items match incoming requests.
    c) Execute all line item pacing plans as evenly across the time period as possible.
    d) Periodically poll the General Planner to receive one or more **'plans'** for each Line Item. A plan is a number of **'tokens'** that need to be spent by this particular server over a particular time period (e.g. 5-minutes).
    e) Periodically report metrics to the Data Service.
2) Prebid Server connects to several local lookup services including a local **User Data Store** to obtain both user segment and frequency capping information. This component is not currently part of the open source release, as its assumed that most potential host companies will already have a method of supplying user segments.
3) The **General Planner** is a centralized component that coordinates communication with planner adapters and manages the messy reality of a global network of Prebid Servers. It has these responsibilities:
    a) Poll the bidder-specific Planning Adapters for updated plans and line items.
    b) Track how each Prebid Server is delivering each line item, and whether servers come online or disappear.
    c) Allocate each plan across Prebid Servers.
    d) Respond to Prebid Server requests.
4) The job of the **Delivery Stats Service** is to collect and aggregate key metrics for use by other components:
    a) Receive aggregated metrics from each Prebid Server
    b) Provide metrics to the General Planner
    c) Provide metrics to Bidder-Specific Planning Adapters
5) The **Dimension Value API** is a support component that provides targeting values to bidder-specific components. More on this below.
6) The **Bidder-Specific Planning Adapter** does the strategic pacing work for the line items hosted by that bidder. It's jobs are to:
    a) Query the Delivery Stats Service and internal sources for line item delivery status.
    b) Create the global plan for each line item.
    c) Respond to requests from the General Planner for line item data and plans.
7) Finally, the **Bidder-Specific UI** is where users enter line item data. It needs to:
    a) Query the Dimension Value API to present targetable values to users.
    b) Query the Delivery Stats Service and internal sources for reporting numbers.
    c) Provide line item data to the Bidder-Specific Planning Adapter.

Impressions vs Tokens

In the contract, Programmatic Guaranteed line items are generally committed to deliver a certain number of impressions -- meaning that the creative was rendered in the user's browser or mobile app.

However, a line item may have to be offered to the ad server several times as the preferred bid choice before it receives an actual billable impression. A **token** is a metric that indicates how many times PG needs to give that line item to the ad server in order to obtain an impression.

For example, the line item may require 3,000,000 impressions over 30 days, or about 100,000 imps per day. Let's say that PG line items are being chosen 50% of the time right now in the ad server and that once chosen, the render rate is 90%. Here's a sample calculation:

- In order to wind up with 100,000 final impressions
- we need to account for the render rate of 0.9, which means it needs to be chosen by the ad server 111,111 times
- and then to accommodate the ad server success rate of 0.5, we will need to offer the line item 222,222 times to the ad server today.

These 222,222 opportunities-to-become-impressions are what we call 'tokens' in PG. They're not 'impressions' because we don't want to serve that many impressions at the end of the day. The relationship between tokens and an impression is an ever-changing ratio. e.g. it may be taking 2.2 tokens to get an impression right now for this line item, but that number may change anytime without warning, perhaps in the next few minutes as campaigns in the ad server are updated.

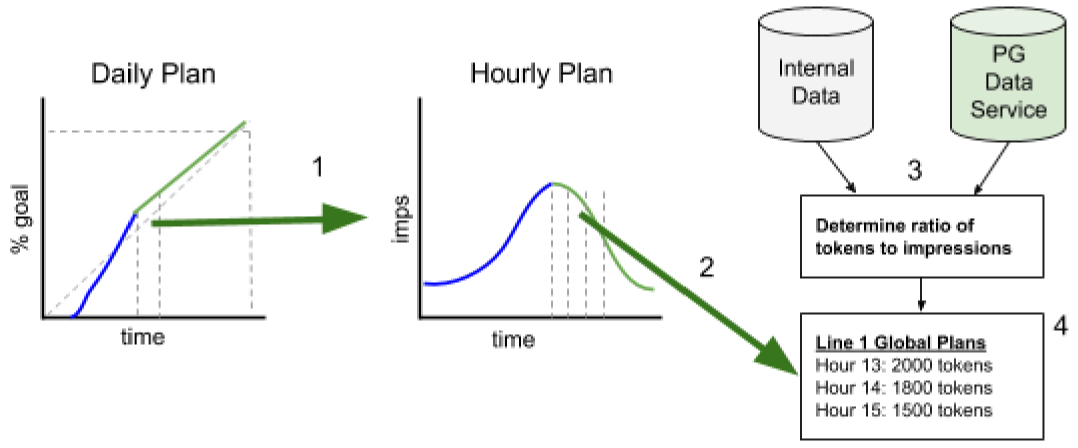
The most important job of the Bidder-Specific Planning adapter is to determine the number of tokens needed for each plan. More on that in the next section.

Plans

A delivery plan is a description for how many tokens a given line item needs to get over a specific time period. The plan starts with the bidder-specific planning adapter and is used by downstream components to pace each line item.

Bidder Planning Adapter

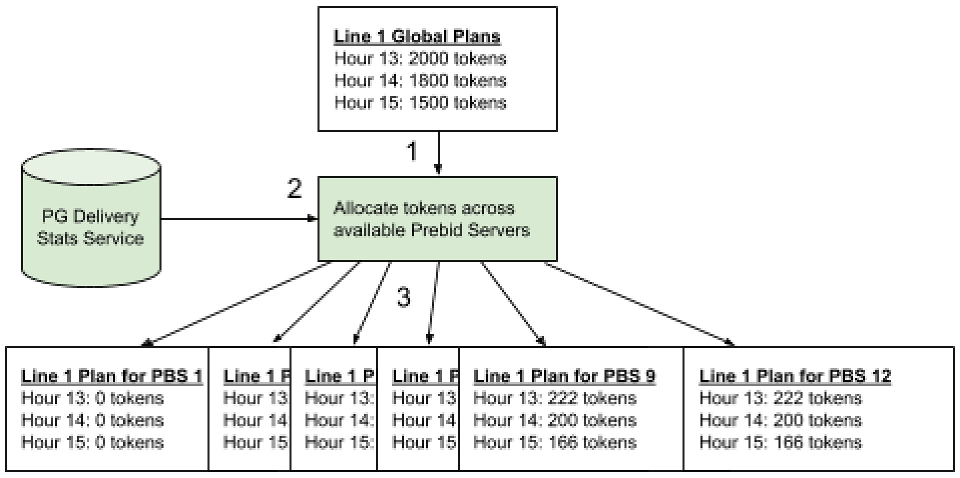
Each bidder planning adapter may have a different approach in how they create plans, but in general the process will have several internal steps:



1) The planning adapter will need to know where each line item is its overall lifecycle.
2) And how traffic matching the target is expected to come in over the near future
3) It will need to keep tabs on how many tokens it's currently taking to get each impression.
4) And it needs to combine that information into a series of time chunks that Prebid Servers can act on.

General Planner

The General Planner's job is to break the 'global' plan across the pool of Prebid Servers. To do this, it needs to know which line items are delivering well on which servers. Delivery may be uneven across servers for many reasons, including geographic distribution of website users, line item targeting, or different hardware capabilities.



1) Accept the global plans coming in from each bidder planning adapter
2) Utilize the line token delivery data from the PG Delivery Stats Service
3) Respond to each Prebid Server request with instructions for how it should deliver each line item.

Prebid Server

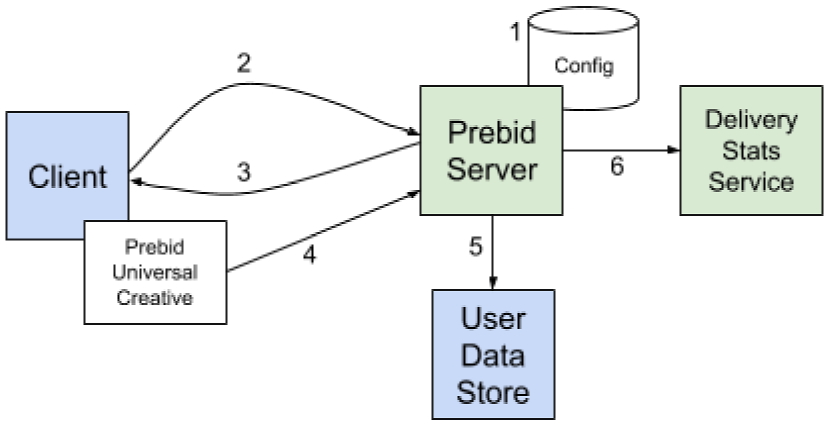
In the Prebid Server world, milliseconds count. The server's job is to spend tokens evenly across each plan without spending a lot of time calculating what to do next. So for instance, if we need to serve 222 tokens in the next hour for Line Item 1, it needs to be the top bid sent to the ad server once every 16 seconds. Once a token is considered spent, that line item will pause for a while until it's time for the next attempt.

Prebid Server counts a token as 'spent' when a PG Line Item is the winning bid sent back to the ad server in the targeting variables hb_pb, hb_bidder, etc.

1) If one or more PG line items exist, there will be at most one from each bidder. Choose randomly between them to determine the 'winner'. Decrement the token count for that line item.
2) If there are no PG line items in the response, the highest CPM open market wins. No tokens are spent.

Events

A responsive PG system cannot wait for offline reporting of impression numbers that could be delayed for many hours. Also, it must support user frequency capping. Both of these features require a real-time feedback loop provided by Prebid events. Currently events are only available in the Java version of Prebid Server. Here's how the event system works:



1) Prebid Server account configuration is set up to enable events.
2) Incoming OpenRTB requests for PG line items need to enable targeting options: ext.prebid.targeting.includewinners and ext.prebid.targeting.includebidderkeys
3) Prebid Server will return the 'hb_winurl' targeting variable, which is sent through to the ad server.

4) When a PG line item wins in the ad server, the Prebid Universal Creative (v1.6 and up) will hit the 'hb_winurl' which informs Prebid Server the ad server has chosen this line item to be displayed.
5) If there's a frequency cap defined on the Line Item, Prebid Server immediately informs the User Data Store of the win so that it can manage the frequency cap.
6) Prebid Server includes this information as part of its periodic report to the Delivery Stats Service. This data should be utilized by the Planning Adapters as part of token calculation.

Note that another future enhancement will be to enable click-tracking within the Prebid stack.

Dimension Value API

That covers the delivery components, but there's an important support server.

Each Prebid Server host company will need to determine which geographic vendor service it uses (Netacuity, MaxMind, etc.) and which device information vendor (DeviceAtlas, 51 Degrees, WURFL, etc.) In addition, each host company will have access to a different set of user segment data.

In order for another company to plug into the PG system, they will need to specify line item targets using the set of targeting attributes that will be available at runtime. For instance, the city code for Pittsburgh, PA is '126' for Netacuity, but '5206379' in MaxMind. So if the host company is using NetAcuity as its geo-lookup service, line item targets will need to refer to device.ext.netacuity.city=126 to target users in Pittsburgh.

The Dimension Value API is populated by each PG host company to contain all attributes and their respective values that can be targeted in their environment. Bidder-specific planning adapters or user interfaces can use this data to present their users with the available targeting options.

Companies connecting to the API can query for:

- A list of supported targeting attributes. This includes the targeting name of the attribute (e.g. 'device.ext.maxmind.connspeed') and a name that can be displayed to users (e.g. 'Connection Speed').
- A list of values supported for a particular attribute. This includes the value needed in the line item target (e.g. '126') and a name that can be displayed to users (e.g. 'Pittsburgh, PA')

See Appendix A for an example list of targeting attributes supported by Rubicon's hosted PG environment. Example spreadsheets to upload into the API will be made available in the open source repository.

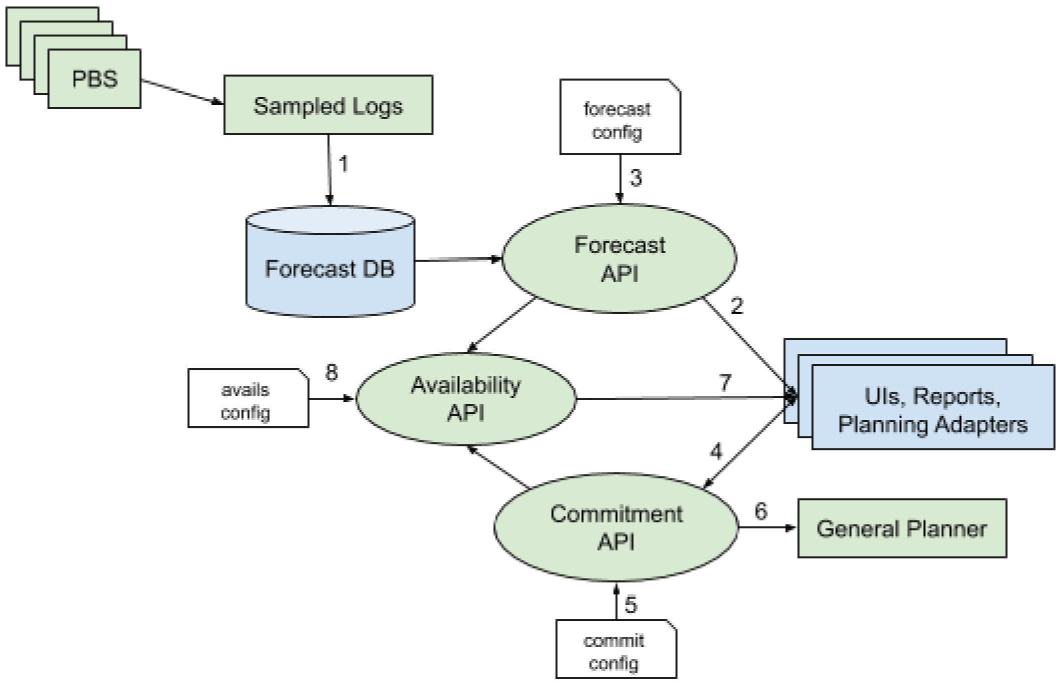
Future Features

Forecast and Availability

Those familiar with guaranteed ad delivery will have noticed that there's been no mention so far of inventory numbers. Since 'Guaranteed' and 'Availability' are two sides of the same coin, that will seem like a glaring omission.

However, in this first phase of Prebid's Programmatic Guaranteed, it is assumed that target availability numbers will be estimated from other systems, presumably the ad server or bidder-specific data sources.

These systems will be built in the next major phases of PG. Here's a preview.



1) Prebid Server will log a sample of requests that are added to a search index.
2) A Forecast API can be contacted to estimate how many requests might match a provided target over a specific time period. This server queries the request index to obtain those estimates.
3) Administrative users will be able to influence the results of forecast queries to account for seasonality, upcoming events, etc. They will also be able to control which bidders have access to which publisher data.

4) A Commitment API will allow Bidder-Specific systems to register already-committed line items.
5) Administrative users will be able to control which commitments can be registered based on bidder, publisher, date range, size, and targeting attributes.
6) The General Planner will consult the Commitment API and discard any incoming plans that are not listed as commitments.
7) The Availability Server API will allow reports and interactive tools to query how many requests could be sold for a certain target after considering forecasts and existing commitments.
8) Administrative users will be able to control availability buffers and which bidders can access which data.

Simulation Test Framework

The open source components of the Prebid PG system are built to operate in a 'simulation' mode that allows testing of the algorithm behavior under different circumstances.

The main idea of this kind of testing is that scenarios are set up and played in "fast-forward". Sample traffic that would come in over 24-hours can be pushed through the system within a few minutes while letting the natural feedback loops operate as they would if things were happening at normal speed.

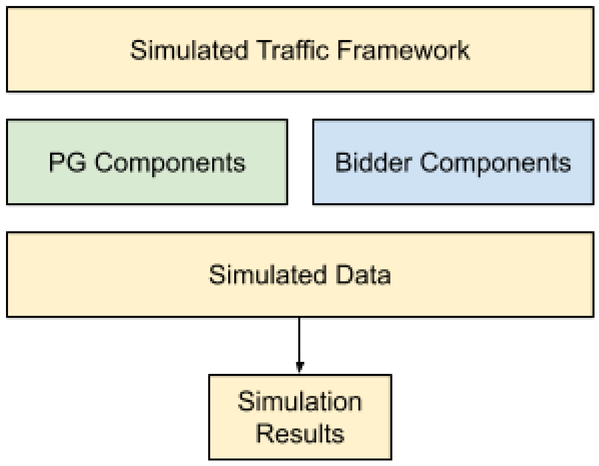
Some characteristics of Line Items that are confirmed by simulation testing:

- The simulation crosses a couple of day boundaries
- Overlapping Line items that have different priorities
- Line items that start and end at different points within the simulation range
- Some line items have have a late start
- Lines cover all the delivery types: even delivery and as-soon-as-possible
- Line have different-sized goals, some big, some very small
- Line items start in an over-delivered state, simulating a change in goal or date

Some characteristics of incoming traffic that are covered:

- Plenty of inventory for all line items
- Oversold
- Unexpected traffic spikes and dropoffs
- Expected traffic spikes and dropoffs
- Changes in conversion rates from bids to impressions

At a high level, the simulation framework surrounds the software with controlled input. At the end of the run, a report indicates how each Line Item in the scenario fared.



The simulation framework and test cases will be open sourced as well, and we will be encouraging Bidder Planning Adapters to support the necessary features so they can be tested under different scenarios. Host companies may insist that planning adapters pass certain tests so they don't adversely affect other bidders by over-pacing or over-delivering.

Appendix A - Sample Target Attributes

These are the targeting attributes Rubicon intends to support in the initial version of PG:

1. adunit.size
2. adunit.mediatype
3. adunit.adslot
4. site.ext.domain
5. site.referrer
6. app.bundle
7. device.geo.ext.netacuity.country
8. device.geo.ext.netacuity.region
9. device.geo.ext.netacuity.metro
10. device.ext.netacuity.connspeed
11. device.ext.deviceatlas.type
12. device.ext.deviceatlas.osfamily
13. device.ext.deviceatlas.browser
14. device.ext.deviceatlas.make
15. device.ext.deviceatlas.model
16. device.ext.deviceatlas.language
17. user.ext.time.userdow
18. user.ext.time.userhour

19. ufpd.ATTR   // user first party data specific to each publisher plugged into the system
20. sfpd.ATTR   // site first party data specific to each publisher plugged into the system
21. bidp.BIDDER.ATTR // specific to the bidders and their parameters
22. segment.SOURCE   // specific to the host company and publisher