OpenRTB

Version 3.0

Released June 2020

Drafted by the Programmatic Supply Chain Technical Working Group.
Please email amit@iabtechlab.com with any feedback.

About IAB Technology Laboratory

Table of contents :

Executive Summary:

This document specifies a standard for the Real-Time Bidding (RTB) Interface. This protocol standard aims to simplify the connection between suppliers of publisher inventory (i.e., exchanges, networks working with publishers, and supply-side platforms) and competitive buyers of that inventory (i.e., bidders, demand side platforms, or networks working with advertisers).

The overall goal of OpenRTB is and has been to create a *lingua franca* for communicating between buyers and sellers. The intent is not to regulate exactly how each party conducts business. As a project, we aim to make integration between parties in the ecosystem easier so that innovation can happen at a deeper-level than the pipes.

OpenMedia Mission

The mission of the OpenMedia project is to spur growth in programmatic marketplaces by providing open industry standards for communication between buyers of advertising and sellers of publisher inventory. There are several aspects to these standards including but not limited to the actual real-time bidding protocol, information taxonomies, offline configuration synchronization, and many more.

History of OpenRTB

OpenRTB was launched as a pilot project between three demand-side platforms (dataxu, MediaMath, and Turn) and three sell-side platforms (Admeld, PubMatic, and The Rubicon Project) in November 2010. The first goal was to standardize communication between parties for exchanging block lists. Version v1.0 of the OpenRTB block list specification was released in December 2010.

After a positive response from the industry, Nexage approached the OpenRTB project with a proposal to create an API specification for OpenRTB focusing on the actual real-time bid request/response protocol and specifically to support mobile advertising. The mobile subcommittee was formed between companies representing the buy-side (dataxu, Fiksu, and [X+1]) and companies representing the sell-side (Nexage, Pubmatic, Smaato, and Jumptap). This project resulted in the OpenRTB v1.0 Mobile specification, which was released in February 2011.

Following the release of the mobile specification, a video subcommittee was formed with video ad exchanges (BrightRoll and Adap.tv) collaborating with dataxu and ContextWeb to incorporate support for video. The goal was to incorporate support for display, video, and mobile in one document. This effort resulted in OpenRTB v2.0, which was released as a unified standard in June 2011.

Due to very widespread adoption by the industry, OpenRTB was established as an IAB standard in January 2012 with the release of version v2.1. Governance over the technical content of the specification remained with the OpenRTB community and its governance rules.

In the years since, programmatic advertising has become a dominant force in the industry. However, this has also led to an increasingly complex supply chain which may increase fraud rates and other risks. This is one of the key motivators driving OpenRTB v3.0.

Regulatory Guidance

OpenRTB implementations will need to ensure compliance on every transaction with all applicable regional legislation.

Architecture

This section describes the underlying model of the ecosystem to which RTB transactions apply and the overall organization of OpenRTB so that specification details have proper context.

OpenRTB Principles

The following points define the guiding principles underlying the OpenRTB specification, some of its basic rules, and its evolution.

- OpenRTB is a living specification. New objects and attributes may be added and enumerated lists may be extended at any time and thus implementers must accept these types of changes without breakage within a version number. See Appendix D: Versioning Policy

    - For example, a demand source with an "OpenRTB 3.0" implementation but must tolerate new fields or enumerated list values it is not expecting, such as from a newer release of OpenRTB 3.0.
    - Likewise, supply sources should freely transmit new fields or enumerated list values (such as from a newer release) and must tolerate bid responses with new fields and enumerated list values it is not expecting.
- Object and attribute names have been made intentionally compact while still trying to balance readability. The reason for this is that these names may be transmitted in plain text extremely frequently.
- OpenRTB imposes no specific representation on its objects. JSON is the default representation, but others may be used. See Representation below.
- All OpenRTB objects may be extended as needed for vendor-specific applications. Extension fields for OpenRTB object must always be placed within a subordinate "ext" object. Most enumerated lists when indicated can also be extended to include vendor-specific codes typically starting at 500.

Terminology

The following terms are used throughout this document specifically in the context of OpenRTB and this specification.

Term	**Definition**
RTB	Bidding for individual items in real-time or near real-time.
Supply Source	The publisher or other audience source selling inventory.
Exchange	An entity (e.g., an SSP) that conducts an auction among bidders per item.
Intermediary	An entity in the path between supply and demand sources (e.g., an exchange).
Supply Chain	A set of 1+ intermediaries between the ultimate supply and demand sources.
Demand Source	An entity (e.g., a DSP) that bids for items into an auction.

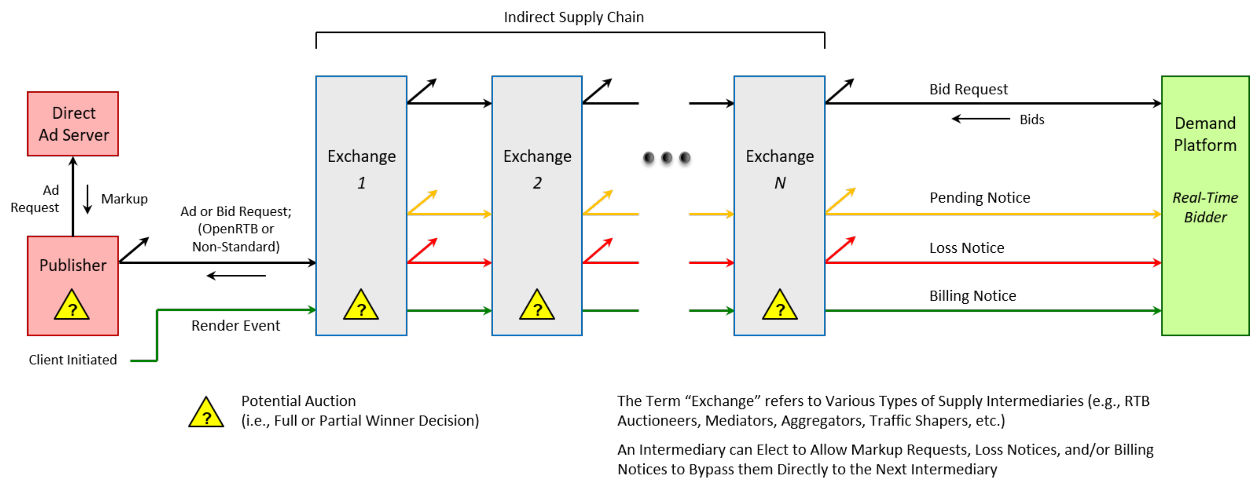
Seat	A buying entity (e.g., advertiser, agency) that wishes to obtain items and use a demand platform to act on its behalf; usually the owner of the buying budget.
Deal	An agreement between sellers and buyers to purchase items under certain terms.

Reference Model

OpenRTB is generally thought of as the interaction between an exchange and its demand sources or sometimes as the auctioneer and its bidders. With that focus, OpenRTB defines a method of selling micro-commodities (e.g., advertising impressions) by broadcasting bid requests from the exchange to its demand partners, collecting bids in response, and determining a winner based on some form of auction rules. An exchange may notify buyers that their bids have been accepted and when a bid becomes billable based on exchange business policy (e.g., rendered markup), the exchange notifies the winning buyer with critical data such as the clearing price. Others may be notified that they lost the auction.

Today's ecosystem is much more complicated, however. Header bidding has become extremely common, which results in a new decision point at or near the publisher or user agent rendering the exchange as no longer the sole decider. It has also become common to have more than one supply chain intermediary; perhaps a mediator or exchange which uses another exchange as a demand source in addition to its own directly integrated demand.

These structures are depicted in the reference model diagram that follows, where an exchange may itself act as a demand source bidding into an upstream (to the left) intermediary and/or use other downstream (to the right) exchanges and DSPs as demand sources. Note that although the figure shows the general case of *N* exchanges or supply intermediaries, practical considerations (e.g., latency, successive revenue shares, etc.) will tend to impose natural limits.


Indirect Supply Chain
Direct Ad Server
Ad Request
Markup
Publisher
Client Initiated
Ad or Bid Request; (OpenRTB or Non-Standard)
Render Event
Exchange 1
Exchange 2
Exchange N
Bid Request
Bids
Pending Notice
Loss Notice
Billing Notice
Demand Platform
Real-Time Bidder
Potential Auction (i.e., Full or Partial Winner Decision)
The Term "Exchange" refers to Various Types of Supply Intermediaries (e.g., RTB Auctioneers, Mediators, Aggregators, Traffic Shapers, etc.)
An Intermediary can Elect to Allow Markup Requests, Loss Notices, and/or Billing Notices to Bypass them Directly to the Next Intermediary

OpenRTB bid requests, bid responses, and events can be implemented between any pair of these entities although the publisher integrations, depicted here as publisher to Exchange-1, are usually less standardized. A given entity may or may not be aware of the larger supply chain, but consider that this chain may vary for different publishers. In any case, OpenRTB assumes that a given entity only has a business relationship, at least within the context of a given transaction, with its directly implemented neighbors to the left (supply side; upstream) and the right (demand side; downstream).

The implications of this can be illustrated in event propagation and private marketplace deals. Speaking in the first person from the point of view of an entity in this reference model:

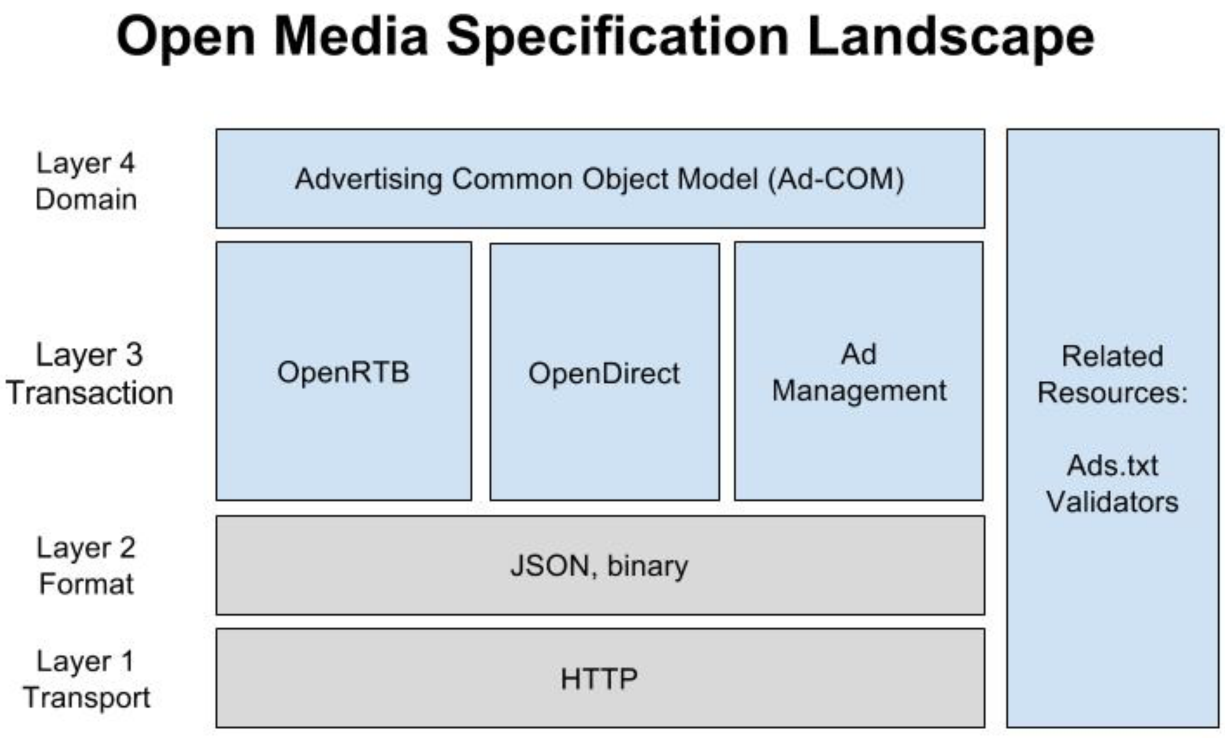
- I assume that all pending, billing, or loss events I receive are either from or with the full authority of my supply side partner (i.e., the entity to my immediate left).
- I am responsible for sending pending, billing, and loss events only to my demand-side partners (i.e., the directly integrated entities to my immediate right).
- I broker and execute deals among my supply and demand partners (i.e., the entities to my immediate left and right). No other entity has any obligation under OpenRTB to extend or propagate my deals further up or down the chain.

Protocol Layers

To assist in reuse of objects across different specifications and to enable various aspects of the specification to evolve at different paces, a layered approach is being adopted as of OpenRTB v3.0. The following illustrates this model. Expressed informally, Layer-1 moves bytes between parties, Layer-2 expresses the language of these bytes, Layer-3 specifies a transaction using this language, and Layer-4 describes the goods being transacted.

Layer 4 Domain	Object Describing the Subject of the Transaction
Layer 3 Transaction	Objects, Events, & Macros that Perform Functions with Domain Objects
Layer 2 Format	Representation; Method of Encoding Data Payloads
Layer 1 Transport	Message Protocol for Communication among Entities

Given this layered concept, the IAB Tech Lab has defined an overall organization of related specifications as "Open Media". The landscape of these specifications and how they may be organized into protocol layers is illustrated as follows.



The following subsections specify these layers as they pertain to the OpenRTB specification. Unless explicitly specified otherwise, annotated as optional, or called out as a best practice, all material aspects of these subsections are required for OpenRTB compliance.

Layer 1: Transport

Communications

The base protocol between an exchange and its demand sources is HTTP. Specifically, HTTP POST is required for bid requests to accommodate greater payloads than HTTP GET and facilitate the use of binary representations. Notification events may be either POST or GET at the discretion of the exchange.

Calls returning content (e.g., a bid response) should return HTTP code 200. Calls returning no content in response to valid requests (e.g., an empty bid response which is one option for indicating no-bid, an event notification) should return HTTP 204. Invalid calls (e.g., a bid request containing a malformed or corrupt payload) should return HTTP 400 with no content.

Version Headers

The OpenRTB version must be passed in the header of a bid request with a custom header parameter. This will allow bidders to recognize the version of the message contained before attempting to parse the request. See Versioning Policy and OpenRTB Principles for more regarding versioning.

```
x-openrtb-version: 3.0
```

Additionally, it is recommended albeit optional that responses include an identically formatted HTTP header with the protocol version implemented by the responder. It is assumed, however, that any response will be compatible with the version of the request and that version support is discussed beforehand between the parties.

BEST PRACTICE: One of the simplest and most effective ways of improving connection performance is to enable HTTP Persistent Connections, also known as Keep-Alive. This has a profound impact on overall performance by reducing connection management overhead as well as CPU utilization on both sides of the interface.

Transport Security

As of OpenRTB v3.0, HTTPS and Transport Layer Security (TLS) version 1.2+ are required for compliance and thus all connections over which the OpenRTB protocol operates must be HTTPS. Legacy integrations that still use unsecured HTTP are no longer considered compliant.

Layer 2: Format

Representation

JSON (JavaScript Object Notation) is the default format for bid request and bid response data payloads. JSON was chosen for its combination of human readability and relative compactness.

Optionally, an exchange may also offer binary representations (e.g., compressed JSON, ProtoBuf, Avro, etc.), which can be more efficient in terms of transmission time and bandwidth. The IAB Tech Lab may offer reference implementations for these or other formats. When available, the use of these IAB reference implementations is highly recommended to reduce integration variations.

The bid request specifies the representation as a mime type using the Content-Type HTTP header. The mime type for the standard JSON representation is `application/json` as shown. The format of the bid response must be the same as the bid request.

```
Content-Type: application/json
```

If alternative binary representations are used, the exchange or SSP should specify the Content-Type appropriately. For example: `avro/binary` or `application/x-protobuf`. If the content-type is missing, the bidder should assume the type is `application/json`, unless a different default has been selected by an exchange.

Encoding

Compressing data sent between exchanges and demand sources can be very beneficial. Compression greatly reduces the size of data transferred and thus saves network bandwidth for both exchanges and demand sources. To realize this savings fully, compression should be enabled for both the bid request sent by the exchange and the bid response returned by the demand source.

Compression can be enabled on the bid response using standard HTTP 1.1 mechanisms. Most web servers already support gzip compression of response content and as such it is an ideal choice. For an exchange to signal they would like the response to be compressed, it should set the standard HTTP 1.1 Accept-Encoding header. The encoding value used should be `gzip`.

```
Accept-Encoding: gzip
```

This header represents to demand sources an indication by the exchange that it is capable of accepting gzip encoding for the response. If a demand source server supports this and is correctly configured, it will automatically respond with content that is gzip encoded. This will be indicated using the standard HTTP 1.1 Content-Encoding header.

```
Content-Encoding: gzip
```

To enable compression on the bid request, it must first be agreed upon between the exchange and the demand source that this is supported. This is similar to when a custom data format is used since the exchange has to know both format and encoding before sending the bid request. If the demand source supports it, the exchange should indicate it is sending a gzip compressed bid request by setting the HTTP 1.1 Content-Encoding header. The encoding value used should be `gzip`.

```
Content-Encoding: gzip
```

If this header is not set then it is assumed that the request content isn't encoded. In HTTP 1.1, the Content-Encoding header is usually only used for response content. However by using this header for the request content as well we are able to indicate a request is compressed regardless of the data format used. This is useful since even binary data formats can benefit from being compressed.

Layer 3: Transaction

The Transaction Layer is the heart of the real-time bidding protocol that comprises OpenRTB. It defines the commerce protocol between an exchange and its bidders, or more generally a supply chain intermediary and its integrated demand sources.

Refer to the Specification section of this document for full details.

Layer 4: Domain

The Domain Layer defines the objects on which the Transaction Layer operates; the media exchange being transacted. In a typical advertising auction, the bid request would contain domain objects in two places. First, the overall request would contain domain objects that describe the context for the sale such as the site or app, the device, and the user. Second, each item being offered for sale would contain domain objects that define the item such as impression and placement details, specifications, and restrictions. Each bid in a response would include domain objects that define the media to be delivered to the user if the auction is won.

Since the version of the Domain Layer specification can vary independent of the Transaction Layer, the root object in the Transaction Layer includes domain specification and version information. This is also critical since support for different versions will vary over time by exchange and/or demand source.

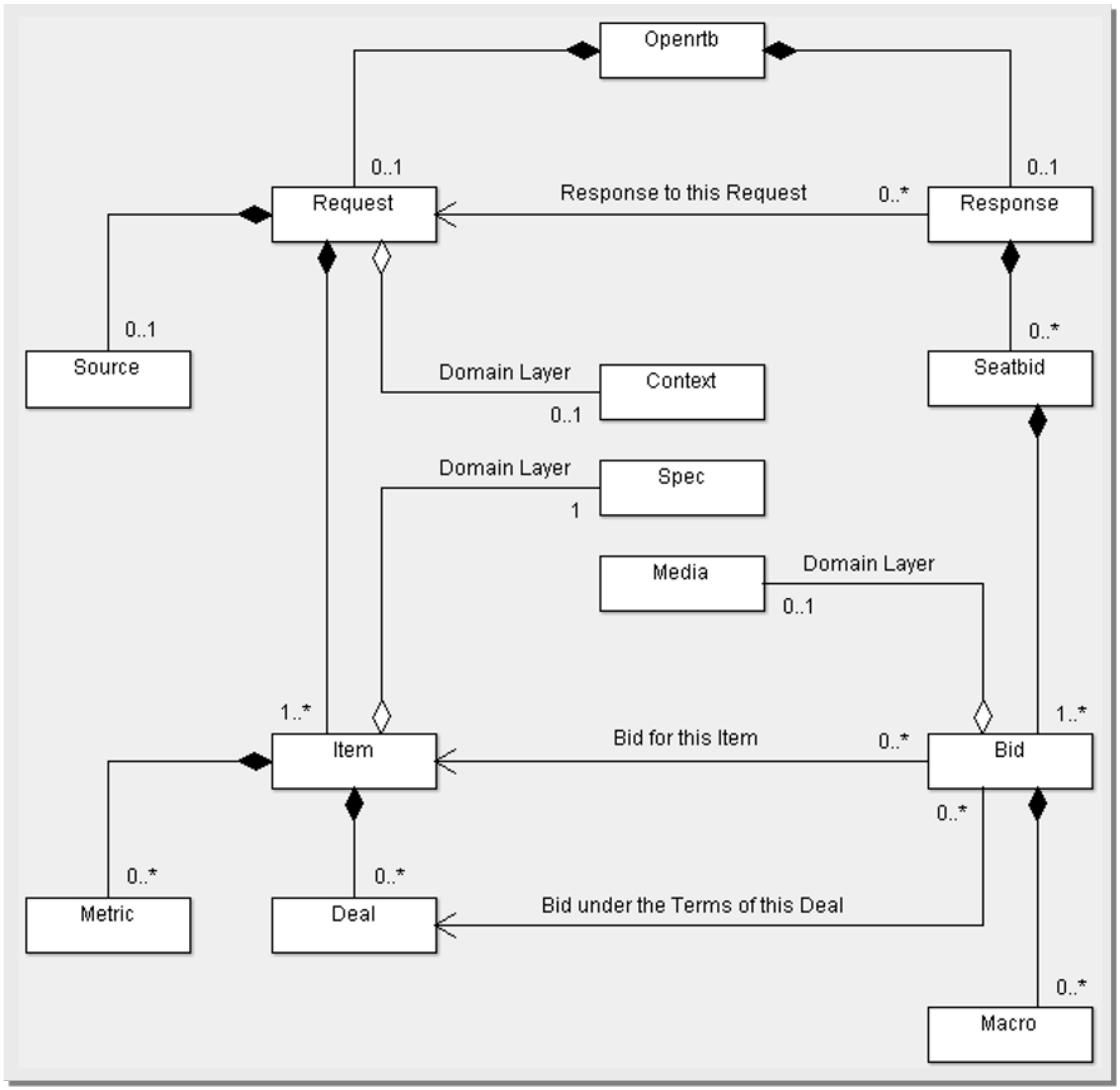
For OpenRTB, the Advertising Common Object Model (AdCOM) is the default Domain Layer.

Specification

This section contains the detailed OpenRTB transaction layer specification. Unless explicitly specified otherwise, annotated as optional, or called out as a best practice, all material aspects of this section are required for OpenRTB compliance.

Object Model

The UML class diagram that follows illustrates the overall payload structure including both request and response objects. Payloads are rooted in named objects; `Openrtb` as a common root and `Request` and `Response` as subordinate roots to identify the payload type.



Throughout the object model subsections, attributes may be indicated as “Required” or “Recommended”. Attributes are deemed *required* only if their omission would break the protocol and is not necessarily an indicator of business value otherwise. Attributes are *recommended* when their omission would not break the protocol but would dramatically diminish business value.

From a specification compliance perspective, any attribute not denoted *required* is optional, whether *recommended* or not. An optional attribute may have a default value to be assumed if omitted. If no default is indicated, then by convention its absence should be interpreted as *unknown*, unless otherwise specified. Empty strings or null values should be interpreted the same as omitted (i.e., the default if one is specified or *unknown* otherwise).

BEST PRACTICE: Exchanges and demand sources are encouraged to publish to their partners the set of optional objects and attributes they support along with any extensions to the specification.

Object: Openrtb

This top-level object is the root for both request and response payloads. It includes versioning information and references to the Layer-4 domain model on which transactions are based. By default, the domain model used by OpenRTB is the Advertising Common Object Model (AdCOM).

Note: As a convention in this document, objects being defined are denoted with uppercase first letter in deference to the common convention for class names in programming languages such as Java, whereas actual instances of objects and references thereto in payloads are lowercase.

Attribute	Type	Definition
`ver`	string	Version of the Layer-3 OpenRTB specification (e.g., "3.0").
`domainspec`	string; default “adcom”	Identifier of the Layer-4 domain model used to define items for sale, media associated with bids, etc.
`domainver`	string; required	Specification version of the Layer-4 domain model referenced in the `domainspec` attribute.
`request`	object; required *	Bid request container. * Required only for request payloads. Refer to Object: Request.

`response`	object; required *	Bid response container. * Required only for response payloads. Refer to Object: Response.

Some of these attributes are optional. The `ver` attribute, for example, indicates the OpenRTB specification version to which this payload conforms. This is also conveyed in Layer-1 via an HTTP header. Its utility here is more to assist in diagnostics by making the payload more self-documenting outside the context of a runtime transaction.

The `domainver` attribute, however, does have runtime utility since the structures of Layer-4 objects may vary over time based on their specification versions. This attribute can assist in invoking the correct domain object parser or unmarshalling code.

Bid Request Payload

The request object contains minimal high level attributes (e.g., its ID, test mode, auction type, maximum auction time, buyer restrictions, etc.) and subordinate objects that cover the source of the request and the actual offer of sale. The latter includes the item(s) being offered and any applicable deals.

There are two points in this model that interface to Layer-4 domain objects: the `Request` object and the `Item` object. Domain objects included under `Request` would include those that provide context for the overall offer. These would include objects that describe the site or app, the device, the user, and others. Domain objects included in an `Item` object would specify details about the item being offered (e.g., the impression opportunity) and specifications and restrictions on the media that can be associated with acceptable bids.

Object: Request

The `Request` object contains a globally unique bid request ID. This `id` attribute is required as is an `Item` array with at least one object (i.e., at least one item for sale). Other attributes establish rules and restrictions that apply to all items being offered. This object also interfaces to Layer-4 domain objects for context such as the user, device, site or app, etc.

Attribute	Type	Definition

`id`	string; required	Unique ID of the bid request; provided by the exchange.
`test`	integer; default 0	Indicator of test mode in which auctions are not billable, where 0 = live mode, 1 = test mode.
`tmax`	integer	Maximum time in milliseconds the exchange allows for bids to be received including Internet latency to avoid timeout. This value supersedes any general guidance from the exchange. If an exchange acts as an intermediary, it should decrease the outbound `tmax` value from what it received to account for its latency and the additional internet hop.
`at`	integer; default 2	Auction type, where 1 = First Price, 2 = Second Price Plus. Values greater than 500 can be used for exchange-specific auction types.
`cur`	string array; default [“USD”]	Array of accepted currencies for bids on this bid request using ISO-4217 alpha codes. Recommended if the exchange accepts multiple currencies. If omitted, the single currency of “USD” is assumed.
`seat`	string array	Restriction list of buyer seats for bidding on this item. Knowledge of buyer’s customers and their seat IDs must be coordinated between parties beforehand. Omission implies no restrictions.
`wseat`	integer; default 1	Flag that determines the restriction interpretation of the `seat` array, where 0 = block list, 1 = whitelist.
`cdata`	string	Allows bidder to retrieve data set on its behalf in the exchange’s cookie (refer to `cdata` in Object: Response) if supported by the exchange. The string must be in base85 cookie-safe characters.

`Source`	object	A `Source` object that provides data about the inventory source and which entity makes the final decision. Refer to Object: Source.
`item`	object array; required	Array of `Item` objects (at least one) that constitute the set of goods being offered for sale. Refer to Object: Item.
`package`	integer	Flag to indicate if the Exchange can verify that the items offered represent all of the items available in context (e.g., all impressions on a web page, all video spots such as pre/mid/post roll) to support road-blocking, where 0 = no, 1 = yes.
`context`	object; recommended	Layer-4 domain object structure that provides context for the items being offered conforming to the specification and version referenced in **`openrtb.`**`domainspec` and **`openrtb.`**`domainver`. For AdCOM v1.x, the objects allowed here all of which are optional are one of the `DistributionChannel` subtypes (i.e., `Site`, `App`, or `Dooh`), **`User`**, `Device`, `Regs`, `Restrictions`, and any objects subordinate to these as specified by AdCOM.
`ext`	object	Optional exchange-specific extensions.

Object: Source

This object carries data about the source of the transaction including the unique ID of the transaction itself, source authentication information, and the chain of custody.

NOTE: Attributes `ds`, `dsmap`, `cert`, and `digest` support digitally signed bid requests as defined by the Ads.cert: Signed Bid Requests specification. As the Ads.cert specification is still in its BETA state, these attributes should be considered to be in a similar state.

Attribute	Type	Definition
`tid`	string; recommended	Transaction ID that must be common across all participants throughout the entire supply chain of this transaction. This also applies across all participating exchanges in a header bidding or similar publisher-centric broadcast scenario.
`ts`	integer; recommended	Timestamp when the request originated at the beginning of the supply chain in Unix format (i.e., milliseconds since the epoch). This value must be held as immutable throughout subsequent intermediaries.
`ds`	string; recommended	Digital signature used to authenticate the origin of this request computed by the publisher or its trusted agent from a digest string composed of a set of immutable attributes found in the bid request. Refer to Section "Inventory Authentication" for more details.
`dsmap`	string	An ordered list of identifiers that indicates the attributes used to create the digest. This map provides the essential instructions for recreating the digest from the bid request, which is a necessary step in validating the digital signature in the `ds` attribute. Refer to Section "Inventory Authentication" for more details.
`cert`	string; recommended	File name of the certificate (i.e., the public key) used to generate the digital signature in the `ds` attribute. Refer to Section "Inventory Authentication" for more details.

`digest`	string	The full digest string that was signed to produce the digital signature. Refer to Section "Inventory Authentication" for more details. **NOTE:** This is only intended for debugging purposes as needed. It is not intended for normal Production traffic due to the bandwidth impact.
`pchain`	string	Payment ID chain string containing embedded syntax described in the TAG Payment ID Protocol. **NOTE:** Authentication features in this Source object combined with the "ads.txt" specification may lead to the deprecation of this attribute.
`ext`	object	Optional exchange-specific extensions.

Object: Item

This object represents a unit of goods being offered for sale either on the open market or in relation to a private marketplace deal. The `id` attribute is required since there may be multiple items being offered in the same bid request and bids must reference the specific item of interest. This object interfaces to Layer-4 domain objects for deeper specification of the item being offered (e.g., an impression).

Attribute	**Type**	**Definition**
`id`	string; required	A unique identifier for this item within the context of the offer (typically starts with "1" and increments).

`qty`	integer; default 1	The quantity of billable events which will be deemed to have occurred if this item is purchased. In most cases, this represents impressions. For example, a single display of an ad on a DOOH placement may count as multiple impressions on the basis of expected viewership. In such a case, qty would be greater than 1. Only one of 'qty' or 'qtyflt' may be present.
`qtyflt`	float	The quantity of billable events which will be deemed to have occurred if this item is purchased. This version of the fields exists for cases where the quantity is not expressed as a whole number. For example, a DOOH opportunity may be considered to be 14.2 impressions. Only one of 'qty' or 'qtyflt' may be present.
`seq`	integer	If multiple items are offered in the same bid request, the sequence number allows for the coordinated delivery.
`flr`	float	Minimum bid price for this item expressed in CPM.
`flrcur`	string; default "USD"	Currency of the `flr` attribute specified using ISO-4217 alpha codes.
`exp`	integer	Advisory as to the number of seconds that may elapse between auction and fulfilment.
`dt`	integer	Timestamp when the item is expected to be fulfilled (e.g. when a DOOH impression will be displayed) in Unix format (i.e., milliseconds since the epoch).

`dlvy`	integer; default 0	Item (e.g., an Ad object) delivery method required, where 0 = either method, 1 = the item must be sent as part of the transaction (e.g., by value in the bid itself, fetched by URL included in the bid), and 2 = an item previously uploaded to the exchange must be referenced by its ID. Note that if an exchange does not support prior upload, then the default of 0 is effectively the same as 1 since there can be no items to reference.
`metric`	object array	An array of `Metric` objects. Refer to Object: Metric.
`deal`	object array	Array of `Deal` objects that convey special terms applicable to this item. Refer to Object: Deal.
`private`	integer; default 0	Indicator of auction eligibility to seats named in `Deal` objects, where 0 = all bids are accepted, 1 = bids are restricted to the deals specified and the terms thereof.
`spec`	object; required	Layer-4 domain object structure that provides specifies the item being offered conforming to the specification and version referenced in **`openrtb`**`.domainspec` and **`openrtb`**`.domainver`. For AdCOM v1.x, the objects allowed here are `Placement` and any objects subordinate to these as specified by AdCOM.
`ext`	object	Optional exchange-specific extensions.

Object: Deal

This object constitutes a specific deal that was struck in advance between a seller and a buyer. Its presence indicates that this item is available under the terms of that deal.

Attribute	Type	Definition

`id`	string; required	A unique identifier for the deal.
`flr`	float	Minimum deal price for this item expressed in CPM.
`flrcur`	string; default "USD"	Currency of the `flr` attribute specified using ISO-4217 alpha codes.
`at`	integer	Optional override of the overall auction type of the request, where 1 = First Price, 2 = Second Price Plus, 3 = the value passed in `flr` is the agreed upon deal price. Additional auction types can be defined by the exchange using 500+ values.
`wseat`	string array	Whitelist of buyer seats allowed to bid on this deal. IDs of seats and the buyer's customers to which they refer must be coordinated between bidders and the exchange beforehand. Omission implies no restrictions.
`wadomain`	string array	Array of advertiser domains (e.g., advertiser.com) allowed to bid on this deal. Omission implies no restrictions.
`ext`	object	Optional exchange-specific extensions.

Object: Metric

This object is associated with an item as an array of metrics. These metrics can offer insight to assist with decisioning such as average recent viewability, click-through rate, etc. Each metric is identified by its type, reports the value of the metric, and optionally identifies the source or vendor measuring the value.

Attribute	**Type**	**Definition**

`type`	string; required	Type of metric being presented using exchange curated string names which should be published to bidders beforehand.
`value`	float; required	Number representing the value of the metric. Probabilities must be in the range 0.0 – 1.0.
`vendor`	string; recommended	Source of the value using exchange curated string names which should be published to bidders beforehand. If the exchange itself is the source versus a third party, "EXCHANGE" is recommended.
`ext`	object	Optional exchange-specific extensions.

Bid Response Payload

The response object contains minimal high level attributes (e.g., reference to the request ID, bid currency, etc.) and an array of seat bids, each of which is a set of bids on behalf of a buyer seat.

The individual bid references the item in the request to which it pertains and buying information such as the price, a deal ID if applicable, and notification URLs. The media related to a bid is conveyed via Layer-4 domain objects (i.e., ad creative, markup) included in each bid.

Object: Response

This object is the bid response object under the `Openrtb` root. Its `id` attribute is a reflection of the bid request ID. The `bidid` attribute is an optional response tracking ID for bidders. If specified, it will be available for use in substitution macros placed in markup and notification URLs. At least one `Seatbid` object is required, which contains at least one `Bid` for an item. Other attributes are optional.

To express a “no-bid”, the most compact option is simply to return an empty response with HTTP 204. However, if the bidder wishes to convey a reason for not bidding, a `Response` object can be returned with just a reason code in the `nbr` attribute.

Attribute	Type	Definition
`id`	string; required	ID of the bid request to which this is a response; must match the `request.id` attribute.
`bidid`	string	Bidder generated response ID to assist with logging/tracking.
`nbr`	integer	Reason for not bidding if applicable (see List: No-Bid Reason Codes). Note that while many exchanges prefer a simple HTTP 204 response to indicate a no-bid, responses indicating a reason code can be useful in debugging scenarios.
`cur`	string; default “USD”	Bid currency using ISO-4217 alpha codes.
`cdata`	string	Allows the bidder to set data in the exchange’s cookie, which can be retrieved on bid requests (refer to `cdata` in Object: Request) if supported by the exchange. The string must be in base85 cookie-safe characters.
`seatbid`	object array	Array of `Seatbid` objects; 1+ required if a bid is to be made. Refer to Object: Seatbid.
`ext`	object	Optional demand source specific extensions.

Object: Seatbid

A bid response can contain multiple `Seatbid` objects, each on behalf of a different buyer seat and each containing one or more individual bids. If multiple items are presented in the request offer, the **`package`** attribute can be used to specify if a seat is willing to accept any impressions that it can win (default) or if it is interested in winning any only if it can win them all as a group.

Attribute	Type	Definition
`seat`	string, recommended	ID of the buyer seat on whose behalf this bid is made.
`package`	integer; default 0	For offers with multiple items, this flag Indicates if the bidder is willing to accept wins on a subset of bids or requires the full group as a package, where 0 = individual wins accepted; 1 = package win or loss only.
`bid`	object array; required	Array of 1+ `Bid` objects each related to an item. Multiple bids can relate to the same item. Refer to Object: Bid.
`ext`	object	Optional demand source specific extensions.

Object: Bid

A `Seatbid` object contains one or more `Bid` objects, each of which relates to a specific item in the bid request offer via the "item" attribute and constitutes an offer to buy that item for a given price.

Attribute	Type	Definition
`id`	string; recommended	Bidder generated bid ID to assist with logging/tracking.

`item`	string; required	ID of the item object in the related bid request; specifically `item.id`.
`price`	float; required	Bid price expressed as CPM although the actual transaction is for a unit item only. Note that while the type indicates float, integer math is highly recommended when handling currencies (e.g., BigDecimal in Java).
`deal`	string	Reference to a deal from the bid request if this bid pertains to a private marketplace deal; specifically `deal.id`.
`cid`	string	Campaign ID or other similar grouping of brand-related ads. Typically used to increase the efficiency of audit processes.
`tactic`	string	Tactic ID to enable buyers to label bids for reporting to the exchange the tactic through which their bid was submitted. The specific usage and meaning of the tactic ID should be communicated between buyer and exchanges beforehand.
`purl`	string	Pending notice URL called by the exchange when a bid has been declared the winner within the scope of an OpenRTB compliant supply chain (i.e., there may still be non-compliant decisioning such as header bidding). Substitution macros may be included.
`burl`	string; recommended	Billing notice URL called by the exchange when a winning bid becomes billable based on exchange-specific business policy (e.g., markup rendered). Substitution macros may be included.

`lurl`	string	Loss notice URL called by the exchange when a bid is known to have been lost. Substitution macros may be included. Exchange-specific policy may preclude support for loss notices or the disclosure of winning clearing prices resulting in `${OPENRTB_PRICE}` macros being removed (i.e., replaced with a zero-length string).
`exp`	integer	Advisory as to the number of seconds the buyer is willing to wait between auction and fulfilment.
`mid`	string	ID to enable media to be specified by reference if previously uploaded to the exchange rather than including it by value in the domain objects.
`macro`	object array	Array of **`Macro`** objects that enable bid specific values to be substituted into markup; especially useful for previously uploaded media referenced via the `mid` attribute. Refer to Object: Macro.
`media`	object	Layer-4 domain object structure that specifies the media to be presented if the bid is won conforming to the specification and version referenced in **`openrtb.`**`domainspec` and **`openrtb.`**`domainver`. For AdCOM v1.x, the objects allowed here are “Ad” and any objects subordinate thereto as specified by AdCOM.
`ext`	object	Optional demand source specific extensions.

Object: Macro

This object constitutes a buyer defined key/value pair used to inject dynamic values into media markup. While they apply to any media markup irrespective of how it is conveyed, the principle use case is for media that was uploaded to the exchange prior to the transaction (e.g., pre-registered for creative quality review) and referenced in bid. The full form of the macro to be substituted at runtime is `${CUSTOM_KEY}`, where “**`KEY`**” is the name supplied in the `key` attribute. This ensures no conflict with standard OpenRTB macros.

Attribute	Type	Definition
`key`	string; required	Name of a buyer specific macro.
`value`	string	Value to substitute for each instance of the macro found in markup.
`ext`	object	Optional demand source specific extensions.

Substitution Macros

Event notification URLs and their formats are defined by the demand source. In order for the exchange to convey certain information (e.g., the clearing price), a number of macros can be inserted into these URLs which the exchange is responsible for resolving prior to invoking. Substitution is assumed to be simple in the sense that wherever a legal macro is found, it will be replaced without regard for syntax correctness. Furthermore, if the source value is an optional parameter that was not specified, the macro will simply be removed (i.e., replaced with a zero-length string).

Custom macros may also be supported at the bidder-implementer's discretion. If bidding is done by reference to pre-uploaded ads, support for custom macros to inject dynamic data is generally necessary. Custom macros are used in the form of `${CUSTOM_KEY}` where "**`KEY`**" is the exact string specified in the "Macro" object with no alteration to case, etc. See the "Macro" object for more details. Note that an exchange may also elect whether or not to support custom macros, but it is expected that any exchange that offers the ability to serve pre-uploaded ads will do so.

These same substitution macros can also be placed in the ad markup. The exchange will perform the same data substitutions as in the aforementioned notice URLs. This occurs irrespective of how the markup is obtained. An example use case is for injecting data such as clearing price into tracking URLs that are embedded in the markup.

The following table defines the standard substitution macros. Note that OpenRTB compliant exchanges must support all macros for which data is available and support substitution in both markup and notification URLs.

Macro	Definition
`${OPENRTB_ID}`	ID of the bid request; from `request.id` attribute.
`${OPENRTB_BID_ID}`	ID of the bid; from `response.bidid` attribute.
`${OPENRTB_ITEM_ID}`	ID of the item just won; from `item.id` attribute.
`${OPENRTB_ITEM_QTY}`	Quantity of the item just won; from `item.qty` or `item.qtyflt` attribute.
`${OPENRTB_SEAT_ID}`	ID of the bidder seat; from `seatbid.seat` attribute.
`${OPENRTB_MEDIA_ID}`	ID of previously registered media to be served; from `bid.mid` attribute.
`${OPENRTB_PRICE}`	Clearing price using the same currency as the bid.
`${OPENRTB_CURRENCY}`	The currency used in the bid (explicit or implied); for confirmation only.
`${OPENRTB_MBR}`	Market Bid Ratio defined as: clearance price / bid price.
`${OPENRTB_LOSS}`	Loss reason codes (see List: Loss Reason Codes).

When rendering markup for test or ad quality purposes, some macro values (e.g., clearing price) may not be known. In these cases, substitute `AUDIT` as the macro value.

Prior to substitution, macro data values can be encoded for security purposes using various obfuscation or encryption algorithms. This may be of particular interest for use cases where price information is carried beyond the exchange, through the publisher, and into the device browser via a tracking pixel in the markup.

To specify that a particular macro is to be encoded, the suffix `:X` should be appended to the macro name, where X is a string that indicates the algorithm to be used. Algorithms choices are not defined by this specification and must be mutually agreed upon between parties. As an example, suppose that the price macro is to be encoded using

Base64 and that its code is agreed to be `B64`. The macro would then be written as follows:

```
${OPENRTB_PRICE:B64}
```

Event Notification

The following events are defined within this specification. Additional events may be defined in Layer-4 specific to domain objects. Any entity that invokes an event URL must first ensure the resolution of all standard OpenRTB substitution macros.

Event: Pending

A pending notice indicates that a bid has been selected as the winner within the scope of an OpenRTB compliant supply chain. In a simple supply chain comprising an exchange as the sole decision maker, this is synonymous with winning the auction. For a complex supply chain, this event is initiated by the exchange closest to the publisher who is not conveying its results via OpenRTB. However, since there may still be upstream decisioning that is not OpenRTB compliant (e.g., header bidding), this event cannot always be equated to a win. The exchange conveys this event by invoking the URL provided by the demand source in the `bid.purl` attribute.

NOTE: This event should be interpreted for guidance only (i.e., the bid has passed all OpenRTB compliant gates in the supply chain). It is never an indication of any form of fulfilment or billing.

Event: Billing

A billing event is when a transaction results in a monetary charge from the exchange or other intermediary to one of their demand-side partners. This event is subject to exchange-specific business policies that should be conveyed clearly to their demand partners. For a DSP, this event signals that they can decrement spend against the related campaign. The exchange conveys this event by invoking the URL provided by the demand source in the `bid.burl` attribute.

NOTE: While the clearing price macro may be resolved in other events as well as DSP trackers in markup, this URL properly implemented is the ONLY official method of communicating a billing event in the OpenRTB protocol.

BEST PRACTICE: Firing the billing URL represents the fulfillment of a business transaction between the exchange and its demand partner. This should not be delegated to another party including a client-side pixel, although a pixel may be the initiating signal for billing to the exchange.

BEST PRACTICE: Upon determining a billable event has occurred (e.g., receipt of client-initiated render signal), an exchange should invoke the billing notice from server-side and as "close" as possible to where the exchange books revenue in order to minimize discrepancies between itself and its demand sources.

BEST PRACTICE: Exchanges are highly encouraged to standardize on a client-initiated render event or equivalent as the basis for the billing event as the most consistent approach in a complex supply chain scenario with potentially multiple auction decision points.

BEST PRACTICE: The public internet is noisy and this event is financial in nature. If an entity calling a billing notice receives a response other than HTTP 200 or 204, it should consider a retry scheme (e.g., every 10 seconds for the next minute). Conversely, an entity receiving billing notices should endeavor to make their endpoint idempotent to avoid double counting.

For VAST Video, the IAB prescribes that the VAST impression event is the official signal that the impression is billable. If the `bid.burl` attribute is specified, it too should be fired at the same time if the exchange is adhering to this policy. However, subtle technical issues may lead to additional discrepancies and bidders are cautioned to avoid this scenario. One option is for exchanges nearest a video supply source to use the impression tracker as its billing signal and then use the billing notice as described.

Event: Loss

A loss event is when an exchange determines that a bid has no chance of winning. At this point, an exchange may invoke the URL provided by the demand source in the `bid.lurl` attribute.

NOTE: Absence of a loss notice does not imply a win.

Exchange support for firing loss notifications is not required for OpenRTB compliance. Demand partners are encouraged to check with their supply-side partners. Furthermore, exchange-specific policy may preclude support for loss notices or the disclosure of winning clearing prices resulting in `${OPENRTB_PRICE}` macros being removed (i.e., replaced with a zero-length string).

Complex Event Chains

In the complex supply chain scenario, an intermediary can choose to participate in these events (i.e., receive them, process them, and pass them on) or allow them to bypass. For example, an intermediary such as a bid request traffic shaper that may elect to allow events to bypass directly downstream, whereas a full exchange especially one engaging in revenue-share may need to intercept.

Motivation to intercept versus bypass may also vary by event type. For example, an intermediary will almost always choose to intercept the billing notice. For loss notices, however, some may be interested and others not. Consider an exchange that decides to drop certain bids (e.g., below the floor, blocklist violations, etc.) and include the rest in its bid response, some of which will lose upstream. It should call loss notices for those it drops since it's essentially declaring these losses, but it may elect to allow the upstream entity to invoke loss notices directly downstream when it decides to drop the next bid subset.

To perform interception of an event on a given bid, an intermediary would resolve macros in the original URL, encode that URL as a parameter on a new URL that points back to itself with potentially a new set of macros (e.g., to learn an upstream clearing price), and specify this new URL in the bid returned upstream. Upon receipt of this event URL, the intermediary would process the event, then retrieve, decode, and fire the original URL. To elect the event to bypass, the intermediary would simply preserve the original URL in the bid returned upstream.

Inventory Authentication

The capability to sign a bid request is to provide assurance to the buyer that this unit of inventory is what it claims to be (e.g., to prevent **`domain-spoofer`**`.com` from posing as **`domain`**`.com`). This practice becomes particularly important in complex supply chains where a buyer may not have a trust relationship with all intermediaries. This feature referred to as “ads.cert” can be used on its own, but is most effective in conjunction with the IAB ads.txt specification (refer to iabtechlab.com/ads-txt).

When used together, an exchange can present a bid request to buyers with the assurances *"I am authorized to sell inventory for the publisher of* `domain.com` *AND this unit of inventory is in fact from* `domain.com`*."*

Ads.cert Primer

Digital signatures must be produced either by the publisher or a trusted agent thereof (e.g., a primary technology provider). Hereinafter, the signing party will be referred to simply as the publisher. Prior to authenticating inventory, a publisher must generate a public and private key pair. The private key is used by the publisher for signing and must be kept secret. The public key must be placed in the same publicly accessible web directory as specified by the ads.txt standard.

The entire bid request cannot be signed since there are legitimate use cases for portions of the request to be altered by intermediaries (e.g., new data to enrich the inventory, reducing precision of IP addresses or location information based on privacy policies, etc.). Instead, specific attributes in the request are chosen that are material to inventory authentication and which should be immutable by any intermediary (e.g., the publisher’s site domain or application bundle). These attributes are assembled into a *digest* string in a simple, but very specific manner.

In addition, the digest must contain an element to tie the digital signature to this and only this transaction in order to avoid replay attacks. Therefore, the digest must include the transaction ID (i.e., the `Source.tid` attribute) and/or at least the request origination timestamp (i.e., the `Source.ts` attribute).

On each impression request, the publisher composes the digest and uses it along with their private key to generate a digital signature string. This signature is then made available to the exchange or more generally to the first supply chain entity for placement in the `Source.ds` attribute. The `Source` object must also contain the name of the publisher’s public key located in their ads.txt directory (i.e., the `Source.cert` attribute) and a map of the digest structure used to create the string that was signed (i.e., the `Source.dsmap` attribute).

With this information, a buyer or downstream intermediary wishing to authenticate the request can recreate the digest from fields in the bid request as prescribed by the digest map and validate it against the digital signature using the public key.

Any intermediary receiving a bid request with this attribute is obligated to include it along with all of the material attributes comprising this signature unaltered in all downstream bid requests. Failure to do so will render this unit of inventory unable to be authenticated and subject to rejection by buyers.

Detailed Implementation Guide

To put these concepts into practice, publishers or their trusted agents must know how to generate their public/private key pairs, how to sign transaction digest strings, and how to communicate this and related information to their immediate exchange partners. Both publishers and those vendors wishing to authenticate requests must know the precise method of creating a digest string from bid request attributes and how to validate a signature given a public key.

For these details, please refer to ads.cert Beta Specification.

Enumerations

The following lists define enumerations referenced by attributes in the payload objects.

List: No-Bid Reason Codes

The following table lists the options for a bidder to signal the exchange as to why it did not offer a bid for the item.

Value	Definition
0	Unknown Error
1	Technical Error
2	Invalid Request
3	Known Web Crawler

4	Suspected Non-Human Traffic
5	Cloud, Data Center, or Proxy IP
6	Unsupported Device
7	Blocked Publisher or Site
8	Unmatched User
9	Daily User Cap Met
10	Daily Domain Cap Met
11	Ads.txt Authorization Unavailable
12	Ads.txt Authorization Violation
13	Ads.cert Authentication Unavailable
14	Ads.cert Authentication Violation
15	Insufficient Auction Time
16	Incomplete SupplyChain
17	Blocked SupplyChain Node
500+	Exchange specific values; should be communicated with buyers beforehand.

List: Loss Reason Codes

The following table lists the options for an exchange to inform a bidder as to the reason why they did not win an item.

Value	Definition
0	Bid Won

1	Internal Error
2	Impression Opportunity Expired
3	Invalid Bid Response
4	Invalid Deal ID
5	Invalid Auction ID
6	Invalid Advertiser Domain
7	Missing Markup
8	Missing Creative ID
9	Missing Bid Price
10	Missing Minimum Creative Approval Data
100	Bid was Below Auction Floor
101	Bid was Below Deal Floor
102	Lost to Higher Bid
103	Lost to a Bid for a Deal
104	Buyer Seat Blocked
200	Creative Filtered - General; Reason Unknown
201	Creative Filtered - Pending Processing by Exchange (e.g., approval, transcoding, etc.)
202	Creative Filtered - Disapproved by Exchange
203	Creative Filtered - Size Not Allowed
204	Creative Filtered - Incorrect Creative Format
205	Creative Filtered - Advertiser Exclusions

206	Creative Filtered - Not Secure
207	Creative Filtered - Language Exclusions
208	Creative Filtered - Category Exclusions
209	Creative Filtered - Creative Attribute Exclusions
210	Creative Filtered - Ad Type Exclusions
211	Creative Filtered - Animation Too Long
212	Creative Filtered - Not Allowed in Deal
500+	Exchange specific values; should be communicated with buyers beforehand.

Examples

The following are examples of Layer-3 request/response payloads expressed using JSON.

Bid Request

The following is an example of Layer-3 of a bid request with a single item offered for sale and a single private marketplace deal associated with it. Some optional attributes have been omitted for brevity. Notice that `spec` and `context` are the interfaces to domain objects specified in AdCOM. The `spec` object should have one `placement` object that carries the details of the impression being offered under this item. The `context` object can have any of `device`, `user`, `regs`, `restrictions`, and at most one of `site` (shown in the example), `app`, or `dooh`.

```
{
   "openrtb": {
      "ver": "3.0",
      "domainspec": "adcom",
      "domainver": "1.0",
      "request": {
         "id": "0123456789ABCDEF",
         "tmax": 150,
         "at": 2,
         "cur": [ "USD", "EUR" ],
         "source": {
            "tid": "FEDCBA9876543210",
            "ts": 1541796182157,
            "ds": "AE23865DF890100BECCD76579DD4769DBBA9812CEE8ED90BF",
            "dsmap": "...",
            "cert": "ads-cert.1.txt",
            "pchain": "..."
         },
         "package": 0,
         "item": [
            {
               "id": "1",
               "qty": 1,
               "private": 0,
               "deal": [
                  {
                     "id": "1234",
                     "flr": 1.50
                  }
               ],
               "spec": {
                  "placement": {  Refer to the AdCOM Specification.  }
               }
            }
         ],
         "context": {
            "site": {  Refer to the AdCOM Specification.  },
            "user": {  Refer to the AdCOM Specification.  },
            "device": {  Refer to the AdCOM Specification.  },
            "regs": {  Refer to the AdCOM Specification.  },
            "restrictions": {  Refer to the AdCOM Specification.  }
         }
      }
   }
}
```

Bid Response

The following is an example of Layer-3 of a bid response, which refers to the previous bid request example in terms of `response.id` matching `request.id`, the `bid.item` attribute in the bid referring to request `item.id`, and the deal reference in **`bid`**`.deal` pointing to the offered deal `deal.id` in the request. Notice that `media` is the interface to domain objects specified in AdCOM. The "media" object should have one `ad` object that carries the details of the ad to be served if the bid wins.

For illustration purposes, this example shows both the `mid` parameter to reference previously uploaded media with some macros for dynamic values and a `domain` object reference. In practice, media would be passed either by value (i.e., details included as domain objects) or pass by reference (i.e., using the media ID and optional macros).

```
{
   "openrtb": {
      "ver": "3.0",
      "domainspec": "adcom",
      "domainver": "1.0",
      "response": {
         "id": "0123456789ABCDEF",
         "bidid": "0011223344AABBCC",
         "seatbid": [
            {
               "seat": "XYZ",
               "bid": [
                  {
                     "id": "yaddayadda",
                     "item": "1",
                     "deal": "1234",
                     "price": 1.50,
                     "tactic": "...",
                     "purl": "...",
                     "burl": "...",
                     "lurl": "...",
                     "mid": "...",
                     "macro": [
                        {
                           "key": "TIMESTAMP",
                           "value": "1127987134"
                        },
                        {
                           "key": "CLICKTOKEN",
                           "value": "A7D800F2716DB"
                        }
                     ],
                     "media": {
                        "ad": {  Refer to the AdCOM Specification.  }
                     }
                  }
               ]
            }
         ]
      }
   }
}
```

Appendix A: Additional Resources

Interactive Advertising Bureau Technology Laboratory (IAB Tech Lab)

www.iabtechlab.com

Creative Commons / Attribution License

creativecommons.org/licenses/by/3.0

AdCOM

https://tools.iabtechlab.com/docs/spec/adcom

ads.text Specification

iabtechlab.com/ads-txt

JavaScript Object Notation (JSON)

www.json.org

Apache Avro

Avro.apache.org

Protocol Buffers (Protobuf)

github.com/google/protobuf

Appendix B: Change Log

This appendix serves as a brief summary of changes to the specification. These changes pertain only to the substance of the specification and not routine document formatting, information organization, or content without technical impact. For that, see Appendix C: Errata.

Version	Release	Changes
3.0	June 2020	**No bid reasons:** Reasons associated with SupplyChain have been added. **DOOH improvements:** A field has been added to represent quantity of impressions as a float, since DOOH impressions may not be a whole number. A corresponding macro has been added too.
3.0	February 2020	**Regulatory guidance:** A section has been added to call attention to the expectation that implementers comply with applicable laws or regulations.
3.0	November 2019	**Versioning policy:** Elaborated on the versioning of this specification.
3.0	November 2018	**Initial release of OpenRTB 3.0:** Complex supply chains, layered specification architecture, and digital signing for inventory authenticity.
2.5	N/A	Support for header bidding, billing and loss notifications, Flex Ads, Payment ID, tactic ID, impression metrics, out-stream video, and many more minor enhancements.

2.4	N/A	Support for Audio ad units and the largest set of minor to moderate enhancements in v2.x history.
2.3	N/A	Support for Native ad units and multiple minor enhancements.
2.2	N/A	New enhancements for private marketplace direct deals, video, mobile, and regulatory signals.
2.1	N/A	Revisions for IQG compliance, minor enhancements, and corrections.
2.0	N/A	Combines display, mobile, and video standards into a unified specification.
1.0	N/A	Original release of OpenRTB Mobile.

Appendix C: Errata

This appendix catalogues any error corrections which have been made to this document after its versioned release. The body of the document has been updated accordingly.

Only minor fixes, such as clarifications or corrections to descriptions, may be treated as errata. Improvements or material changes are summarized in the change log.

Link fixes, language improvements: Some broken links have been fixed. Word choice has been improved in places for clarity. (2020/02/14).

Appendix D: Versioning Policy

As of OpenRTB 3.0, OpenRTB's version number is only incremented on breaking changes. In other words, OpenRTB 3.1 should be considered a distinct version from OpenRTB 3.0 when there is a need for distinguishing versions. For example, a demand source might regard the version header when parsing a bid request received from a supply source. See OpenRTB Principles.

The current version of the OpenRTB specification is updated approximately once a month if there are non-breaking improvements to be released such as new fields, objects, or values in enumerated lists. Errata, such as clarifications or corrections to descriptions not materially impacting the specification itself, are also addressed during monthly updates. See Errata.

Release branches are created for each monthly release and the history of these can be reviewed on GitHub. The master branch for the repository will always reflect the most recent release, whereas ongoing development work occurs in the 'develop' branch.

This versioning policy is a break from historical practice for OpenRTB. In versions of OpenRTB prior to 3.0, major version numbers represent breaking changes and minor version numbers represent non-breaking changes.