API Semantics

API Semantics

This page explains the semantics of our REST API. It includes information on:

- How to ask a service about itself: what fields it supports, which fields are filterable
- How to get only the information you want by filtering and sorting
- The "shape" of our JSON responses in different scenarios

This document assumes you have completed the API Onboarding Process.

On This Page

HTTP Protocol

The AppNexus API supports HTTP Protocol version 1.1 or later. While some calls may work with the deprecated 1.0 version, this is not guaranteed. Please ensure that your client communicates using at least version 1.1.

API Endpoints

The URL for the production API endpoint is: https://api.appnexus.com. Please note that non-secure access to the product API (HTTP) is not available.

Changes made with this API affect the production AppNexus Console. Only authorized users should alter information or settings in this environment.

The URL for the testing API endpoint is: https://api-test.appnexus.com.

This environment replicates the production codebase and is kept up to date on a monthly or shorter release schedule. The environment is made available expressly for clients to test their integrations without having to interfere with production data. See Client Testing Environment for details.

REST Semantics

Our API services are *RESTful*. REST (Representational State Transfer) is a type of software architecture in which requests model the communication from a web browser to a web server. Below are the central REST methods used in our API services, and their uses:

POST	Create
GET	Read
PUT	Update
DELETE	Delete

When making a POST or PUT request, you must include a JSON file with the data to create or update.

PUT overwrites arrays unless 'append=true' is added to query string
For PUT requests, only the fields included in the JSON file will be updated, **except in the case of arrays**. When updating an array using PUT, all fields in the array are **overwritten with the contents of the new array** you upload, unless you append the following to your request query string: `"append=true"`.

>> Example "legacy" PUT request for updating an array
This example walks you through the process of properly updating the `pixels` array of creative ID 503577 using the "legacy" method; in other words, with the "overwrite arrays on PUT" behavior that occurs unless you append the string `"append=true"` to the query string of your request.

First, let's look at the creative. Note that the `pixels` array already includes one pixel.

```
$ curl -b cookies 'https://api.appnexus.com/creative?id=503577'

{
    "response": {
        "status": "OK",
        "count": 1,
        "id": "503577",
        "start_element": 0,
        "num_elements": 100,
        "creative": {
            "name": null,
            "brand_id": 1,
            "media_url": "http://creative.com/300x250",
            "id": 503577,
            ...
            "pixels": [
                {
                    "id": 196,
                    "pixel_template_id": null,
                    "param_1": null,
                    "param_2": null,
                    "param_3": null,
                    "param_4": null,
                    "param_5": null,
                    "format": "url-js",
                    "url":
"http://50.16.221.228/render_js?cb=${CACHEBUSTER}&uid=${USER_ID}"
                }
            ],
            ...
        }
    }
}
```

Next, we create the JSON file for adding a new pixel to the creative. In the file, we include both the new pixel that we want to add **and** t he pixel that was already attached to the creative.

> If we don't include the existing pixel in the JSON file, our update will delete that pixel from the creative.

```
$ cat creative_update

{
    "creative": {
        "pixels": [
            {
                "format": "url-js",

"url":"http://12.19.232.312/render_js?cb=${CACHEBUSTER}&uid=${USER_ID}&ref=${REF
ERER_URL}"
            },
            {
                "id": 196,
                "format": "url-js",
                "url":
"http://50.18.232.228/render_js?cb=${CACHEBUSTER}&uid=${USER_ID}"
            }
        ]
    }
}
```

Then we make a `PUT` call to update the creative with the information in the JSON file. Note that the `pixels` array in the response includes both the new and old pixels.

```
$ curl -b cookies -X PUT -d @creative_update
'https://api.appnexus.com/creative?id=503577'

{
    "response": {
        "status": "OK",
        "count": 1,
        "id": "503577",
        "start_element": 0,
        "num_elements": 100,
        "creative": {
            "name": null,
            "brand_id": 1,
            "media_url": "http://creative.com/300x250",
            "id": 503577,
            ...
            "pixels": [
                {
                    "id": 196,
                    "pixel_template_id": null,
                    "param_1": null,
                    "param_2": null,
                    "param_3": null,
                    "param_4": null,
                    "param_5": null,
                    "format": "url-js",
                    "url":
"http://50.16.221.228/render_js?cb=${CACHEBUSTER}&uid=${USER_ID}
                     &ref=${REFERER_URL}&campaign_id=147"
                },
                {
                    "id": 197,
                    "pixel_template_id": null,
                    "param_1": null,
                    "param_2": null,
                    "param_3": null,
                    "param_4": null,
                    "param_5": null,
                    "format": "url-js",

"url":"http://12.13.234.312/render_js?cb=${CACHEBUSTER}&uid=${USER_ID}&ref=${REF
ERER_URL}"
                }
            ],
            ...
        }
    }
}
```

Using cURL

In our documentation we use curl to make HTTP requests. Curl is a command-line tool for transferring files with URL syntax, supporting FTP, FTPS, HTTP, HTTPS, SCP, SFTP, TFTP, TELNET, DICT, LDAP, LDAPS, and more. Example scripts have been provided on each API wiki page

to illustrate the structure of the `curl` commands you will need to run AppNexus API services. In addition, an example of how to a make a generic POST request is shown below. This example uses the Authentication Service:

```
$ curl -b cookies -c cookies -X POST -d @auth.json 'https://api.appnexus.com/auth'
```

Chunk of Request	What it Means
`-c cookies`	Creates a text file called "cookies" and stores your session token (assigned by the Authentication Service). This is not a required argument to `curl` after the initial authentication, but it doesn't affect subsequent calls if it is included.
`-b cookies`	Retrieves the authentication token that you previously stored in the "cookies" text file.
`-X`	Indicates that you are going to make a certain type of request, in this case "POST".
`-d`	Indicates that you are going to upload a file, in this case "auth.json".
`'https://api.appnexus.com/auth'`	The URL of the service you are making the request to. Use quotes in case you have any special characters in your URL.

> **Use Single Quotes Around Your Request URL**
> Some requests require single quotes around your request URL, as in the above `curl` request. If you get an error message from your UNIX shell, make sure your request URL has single quotes before troubleshooting further. For more information on how UNIX shell quotes and escaping work, see this documentation on quotes and escaping in shells.

Filtering and Sorting

Most API Services support filtering and sorting. Filtering allows you to specify a subset of objects to be returned. Sorting allows you to control the order of the objects returned.

> Please also see the Search Service and Lookup Service for ways of looking up objects across your member.

Get Multiple Objects by ID

You can get multiple specific objects by ID by passing a comma-separated list of IDs. The result object will contain an array holding just those

specific objects. In the example below, we ask the Campaign Service for just the campaigns with IDs 1, 2, and 3.

```
$ curl -bc -cc 'https://api.appnexus.com/campaign?id=1,2,3

{
  "response" : {
     "count" : 3,
     "status" : "OK",
     "campaigns" : [ ... ]
  }
}
```

Filter by IDs

Pass a query string parameter for the field with a comma-separated list of IDs.

Example: Request all campaigns for certain line items.

```
$ curl -b cookies
'https://api.appnexus.com/campaign?advertiser_id=40&line_item_id=1,2,3'
```

Example: Request certain advertisers

```
$ curl -b cookies 'https://api.appnexus.com/advertiser?id=3'
```

> **Only 100 objects will be returned per request**
> The maximum number of objects that can be returned, regardless of pagination, is 100. If you request over 100 objects, we will only return the first 100 and will not provide an error message. For more information on how to paginate API results, see Paging.

Filter by Min and Max Values

Fields that are of the type `int`, `double`, `date`, or `money` can be filtered by `min` and `max`. For example:

```
$ curl -b cookies 'https://api.appnexus.com/campaign?min_id=47'
$ curl -b cookies 'https://api.appnexus.com/campaign?min_advertiser_id=20'
```

Fields of the type `date` can be filtered by `nmin` and `nmax` as well. The `nmin` filter lets you find dates that are either `null` or after the specified date, and the `nmax` filter lets you find dates that are either `null` or before the specified date. For example:

```
$ curl -b cookies
'https://api.appnexus.com/campaign?nmax_start_date=2012-12-20+00:00:00'
$ curl -b cookies
'https://api.appnexus.com/campaign?nmin_end_date=2013-01-01+00:00:00'
```

Note the required date/time syntax in the preceding example: YYYY-MM-DD+HH:MM:SS

Another option for filtering by date is to use the `min_last_modified` filter:

```
$ curl -b cookies
'https://api.appnexus.com/campaign?min_last_modified=2017-10-27+21:00:00'
```

Filter by Field Names

To limit the response to specific fields of an object, pass the `fields` query string parameter with a comma-separated list of field names. For example:

```
$ curl -b cookies "https://api.appnexus.com/user?current&fields=username,user_type,id"

{
    "response":{
        "status":"OK",
        "count":1,
        "start_element":0,
        "num_elements":100,
        "user":{
            "id":14311,
             "username":"rloveland",
             "user_type":"admin"
        }
    }
}
```

Misc Filters on Field

We support the following additional field-based filters on API responses:

- `not_*`
- `like_*`
- `min_*`
- `max_*`
- `nmin_*`
- `nmax_*`
- `having_*`
- `having_min_*`
- `having_max_*`

Example:

```
$ curl -b cookies 'https://api.appnexus.com/placement?like_[fieldName]=partialValue'
```

Search

Some services support `search` as a query string parameter to look for ID or name. For example:

```
$ curl -b cookies 'https://api.appnexus.com/placement?search=17'
```

Sorting

To sort use the `sort` query string parameter and pass in a list of fields you'd like to sort by and whether you want them ascending (`asc`) or descending (`desc`). For example:

```
$ curl -b cookies 'https://api.appnexus.com/campaign?advertiser_id=1&sort=id.desc'
```

Paging

To page, use the `start_element` and `num_elements` parameters. If `num_elements` is not supplied, it defaults to 100 (which is also the maximum value).

```
$ curl -b cookies 'https://api.appnexus.com/campaign?start_element=20&num_elements=10'
```

Append on PUT

By including `append=true` in the query string of a PUT call, a user can update only a particular child object instead of replacing all child objects. In other words, rather than overwriting an entire array with a new one on a PUT call, you can use `append=true` on the query string to add a single element to a long array.

In this example, we'll use `append=true` on a PUT call to toggle the `is_available` flag of an object in the `member_availabilities` array of the Plugin Service. Without the `append=true` flag on the query string, the new item would replace the entire array. In this example, it's only added.

First let's look at the object we'll be modifying (these examples use jq to slice and dice the JSON). Both of the availabilities are set to `true`:

We'll send the following JSON to turn off the `is_available` flag on one of the `member_availability` objects:

```
$ cat plugin-update.json
{
    "plugin" : {
        "developer" : {
            "id" : 1
        },
         "member_availabilities" : [
            {
                "is_available" : false,
                 "id" : 4
            }
        ],
         "name" : "ccc"
    }
}
```

Normally, sending the JSON above on a PUT call would overwrite the whole `member_availabilities` array. However, this time we'll add `"append=true"` to the query string of the call. This tells the API to change just the object whose `id` is 4. We can verify that it's done so by inspecting the output.

```
$ curl -bc -X PUT -d @plugin-update.json
'https://api.appnexus.com/plugin?id=13&append=true' | jq
'.response.plugin.member_availabilities'
[
  {
    "is_available": false,
    "id": 4
  },
  {
    "is_available": true,
    "id": 7
  }
]
```

JSON Basic Structure

Below are the syntax of the components of a JSON object and what they mean.

An object:

```
{. . . }
```

An array:

```
[. . . ]
```

A string:

```
". . ."
```

Associate a key with an alphanumeric string value:

```
"key":"string"
```

Associate a key with a numeric value:

```
"key":int
```

An example that puts them together:

```
{
    "campaign": {
        "name": "my campaign",
        "id": 1434,
        "creatives": [
                {
                    "id": 4162,
                    "state": "active"
                }
            ],
}
}
```

JSON Field Types

POST and PUT requests require JSON data. For PUT requests, only the JSON fields included in a request will be updated. All other fields will be unchanged.

Different fields require different types of values. The table of types below extends those defined in the JSON standard.

Type	Description	Example
boolean	True or false.	`true`
string(100)	A string of 100 characters or less.	`"Homepage Pixel"`
int	An integer.	`87`
decimal	A generic decimal number.	`3.0`
float	A floating-point number with 32-bit precision.	`3.14...`
double	A floating-point number with 64-bit precision.	`3.14...`
enum	One of a number of predetermined values.	`"male"` or `"female"`
money	A floating-point numeric value used to represent money. For more information, see Vertica's Numeric Data Types.	`19.23`

timestamp	A date and time string in the form YYYY-MM-DD HH:MM:SS. All timezones are in UTC unless otherwise noted.	`"2009-01-14 05:41:04"`
date	See `timestamp` above.	
object	A wrapper for any sub-fields under the current field. In the example that follows, the field `"brand"` is a multi-object.	`"brand": { "id": 466, "name": "PKR" }`
array	A list containing one or more values. In our API, arrays most often contain lists of objects, integers, or strings.	`"members" : [ { "id": 1234, "member_use_deal_floor": true, "member_ask_price": 2.15, "name": "Buyer 1" }, { "id": 5561, "member_use_deal_floor": true, "member_ask_price": 2.25, "name": "Buyer 2" } ]`

How and Why Reporting APIs are Different

The reporting APIs available via the Report Service work differently than our other APIs. They have their own multi-step request and response flow. This is required because they process large amounts of data; this processing needs to be performed asynchronously.

For instructions on how to retrieve reports, see the Report Service.

For a tutorial that explains how to use our reporting APIs effectively, see Report Pagination.

A Note on Underscores and Hyphens

JSON fields and values use underscores, e.g., `audit_type_direct`.

API service names in URLs are hyphenated, e.g., `https://api.appnexus.com/insertion-order`.

Response Codes

All API Services return JSON data. When Service calls are successful, the JSON response will include a `"status"` field set to `"OK"`. The response to `POST` and `PUT` calls will also include the ID of the relevant object, as well as any relevant attributes of that object. Every response will include a `"dbg_info"` object that conveys information about the API call and response, such as the API machine that processed the request and the version of the API you're using.

In the example below, we are using cookies to store our authentication token and adding the file "creative" to advertiser 123 with the Creative Service.

```
$ curl -b cookies -X POST -d @creative
'https://api.appnexus.com/creative?advertiser_id=123'
{
    "response": {
        "status": "OK",
        "id": "242",
        "dbg_info": {
            ...
        }
    }
}
```

The table below lists the fields of the `dbg_info` object and their definitions:

Field	Type	Description
`instance`	string	The API machine which processed the request.
`slave_hit`	Boolean	Whether or not the API machine ran SQL queries on a database slave.
`db`	string	The database the query was executed on.
`reads`	int	The number of reads made.
`read_limit`	int	The limit on the number of reads.
`read_limit_seconds`	int	The time period over which the `read_limit` is enforced.
`writes`	int	The number of writes made.
`write_limit`	int	The limit on the number of writes.

`write_limit_seconds`	int	The time period over which the `write_limit` is enforced.
`time`	decimal	The amount of time it took to process the API request, expressed in milliseconds.
`start_microtime`	decimal	The POSIX timestamp of the start time of processing, including milliseconds (right side of the decimal point).
`version`	string	The version of the API.

Error Messages

When invalid input is sent to the API (for example, an incorrect password), a JSON response will be returned with `"error"` and `"error_id"` fiel ds.

```
$ cat auth
{
  "auth": {
     "username":"user1",
     "password":"Wr0ngP@ss"
  }
}

$ curl -b cookies -c cookies -X POST -d @auth 'https://api.appnexus.com/auth'
{
    "response": {
        "error_id": "NOAUTH"
        "error": "No match found for user\/pass",
        "dbg_info": {
            ...
        }
    }
}
```

The `"error"` field is useful for debugging purposes, as it contains a verbose description of the error. The `"error_id"` field can be used programmatically as described in the table below.

> For an in-depth discussion of the errors that your API script should be able to handle, see KB - API Authentication (customer login required).

Error_ID	Meaning	How to Respond

`INTEGRITY`	A client request is inconsistent; for example, a request attempts to delete a default creative attached to an active placement.	Check the request logic for consistency.
`LIMIT`	The user has reached the maximum number of allowed objects of a certain type.	Delete unnecessary objects to get under the limit. If you cannot delete any object, please contact your AppNexus representative.
`NOAUTH`	The user is not logged in, or the login credentials are invalid.	Use the Authentication Service to get a token, or check the username and password in your request.
`NOAUTH_DISABLED`	The user's account has been deactivated.	Login with a different user, or create a user account specifically for API access.
`NOAUTH_EXPIRED`	The user's password has expired and needs to be reset.	Use the Authentication Service to get a new token.
`SYNTAX`	The syntax of the request is incorrect.	Use the `"error"` message to identify the issue and fix the code.
`SYSTEM`	A system error has occurred.	Contact your AppNexus representative.
`UNAUTH`	The user is not authorized to take the requested action.	Check the `"error"` message and make sure the logic in your code is correct.

Related Topics

- API Usage Constraints
- API Best Practices