Search for articles...

NationBuilder v2 API Core Concepts

Capabilities overview of the essential features of the v2 API

Updated over 4 months ago

Table of contents

Table of Contents

Filtering

You can use the **filter** param with any attribute to return a subset of results.

Filter type	Example
Case insensitive match	`/api/v2/signups?filter[note]=my note`
Case sensitive match	`/api/v2/signups?filter[note][eql]=My note`
Prefix	`/api/v2/signups?filter[note][prefix]=my`
Greater than	`/api/v2/signups?filter[donations_amount_in_cents][gt]=500`
Greater than or equal to	`/api/v2/signups?filter[donations_amount_in_cents][gte]=500`
Less than	`/api/v2/signups?filter[donations_amount_in_cents][lt]=1000`
Less than or equal to	`/api/v2/signups?filter[donations_amount_in_cents][lte]=1000`
Exists (is not null)	`/api/v2/signups?filter[middle_name][not_eq]=null`
Does not exist (is null)	`/api/v2/signups?filter[middle_name]=null`

You can also filter sideloaded resources by nesting filter params. The example bellow will return all signups and display any primary, work, and registered addresses with the state code "MT":

```
/api/v2/signups?include=primary_address,work_address,registered_address&filter[addresses]
[state]=MT
```

The first filter param (addresses) is the resource type, not the relationship name.

📌 *Note: The filter does not remove unassociated objects from the response. So, in the example above, signups that do not have addresses matching the criteria of the filter will still be included in the payload.*

Sparse fields

You can use the **fields** param and a comma-delineated list to specify *only* some attributes to return on the resource response.

For example, the following will return only the first name, last name, and email attributes of the signups requested:

```
/api/v2/signups?fields[signups]=first_name,last_name,email
```

As with filters, specification for sparse fields is by the resource type, and **not** the relationship name. So if you wanted to limit sparse fields for a recruiter, for example, you would use `include=recruiter&fields[signups]=first_name,email`.

Sorting

You can use the **sort** param to return the results sorted by ascending or descending.

Sort type	**Example**
Ascending	`/api/v2/signups?sort=first_name`
Descending	`/api/v2/signups?sort=-first_name`

Pagination

Index endpoints return 20 results per response by default. The response includes a `links` field (which is a sibling of `data`) with `self`, `prev` and `next` links. These can be used to paginate results.

The `next` link will always be provided when resources are returned, even if the next page is empty. The last page has been reached once the results returned are empty.

You can also use the `page` parameter to paginate:

- Size per page: `/api/v2/signups?page[size]=10`
- Which page to return: `/api/v2/signups?page[number]=3`

📌 *Note: If you add or delete objects as you are returning sorted results, some of these objects may be skipped or duplicated in your paginated responses.*

Count

You can get the count of any API resource by adding the `stats[total]=count` parameter to your request.

For example, the following will return the count of all donations: `/api/v2/donations?stats[total]=count`

The count will be returned in the `meta` field of the response, which is a sibling to `data`, in the following format:

```
{ meta: { stats: { total: { count: 402 } } } }
```

Relationships between resources

When two endpoints have a relationship with each other, they are referred to as connected resources. You can find a full list of resources, their connected resources, and their relationships, along with an overview of possible relationship types here.

Sideloading

When you're requesting resources from one endpoint, if another resource is connected to the endpoint resource, you can request it by "sideloading" it via the `include` param.

Multiple resources can be sideloaded by using a comma-delineated list.

⚠️ *Please note:* Sideloading resources with a to-many relationship may result in larger-than-expected payloads, which are prone to timeouts, especially when sideloading to-many relationships on index endpoints. For example, a request to `/api/v2/signups?include=taggings` will fetch taggings for each signup within the primary data payload.

Tip: If you find that you're hitting timeouts on these calls, try a combination of reducing the page size of the primary resource and filtering on the sideloaded data to only fetch relevant records (eg. `/api/v2/signups?page[size]=10&include=taggings&filter[taggings][tag_id]=123`)

Paginating sideloads

On **Show** endpoints only, sideloads can be paginated by including the `page[resource_type][size]` and `page[resource_type][number]` params for the resource type you are sideloading and incrementing the page number until an empty response is returned.

The following example will return the 3rd page of petition signatures paginated at 10 signatures per page: `api/v2/petitions/1?include=petition_signatures/&page[petition_signatures][size]=10&page[petition_signatures][number]=3`

Sideposting

You can create and update a resource at the same time as you are creating or updating a connected resource.

Some NationBuilder resources require sideposting a resource with another resource. For example, creating a petition requires the creation of a page at the same time. When this is relevant, it will be noted in the interactive API documentation available in your control panel under **Developer > Api testing**.

Sideposting examples

Sideposting enables API consumers to persist new resources and connect existing resources with the object in the primary "data" payload, all newly created resources within the payload are persisted together within the nation in the same request.

Creating a new signup with sideposting

The payload below creates a new signup with the name 'Kim Possible' and an email, 'test@example.com'. The additional values seen in "relationships" and "included" portions of the payload do the following:

- Creates a new signup, 'Joy Palmer' with an email 'recruiter@example.com' which will be associated with the primary payload as a recruiter.
- Tags the signup with three signup tags, one that already exists and two others that will be created by this request.

```
{ "data": { "type": "signups", "attributes": { "first_name": "Kim", "last_name": "Possible'
```

Note that the already existing signup tag with ID 123 did not need a corresponding payload in the "included" array. The inclusion of the signup tag's data in the relationship section of the payload is sufficient to tag the signup with the provided signup tag.

Updating data with sideposting

In this next example, the signup's first_name and the petition signature's comment are being updated at the same time. Additionally, a previously created petition signature associated with the signup is being deleted in this request and a signup tag is being removed from the signup.

```
{ "data": { "id": "48", "type": "signups", "attributes": { "first_name": "Lucy" }, "relati
```

Note here the use of the "disassociate" method for signup_tags vs the use of the "destroy" method for the petition signature.

- The "disassociate" sidepost method is used to remove a connection between resources without destroying either resource.
- The "destroy" sidepost method is used to delete the associated object from the nation.

Additionally, the inclusion of a petition signature payload in the included array is being used to update the existing petition signature associated with the signup.

Sideposting methods

As highlighted in the examples above, there are multiple methods that can be specified within the 'relationships' section of the payload.

Method	Description
Create	Creates the specified resource as part of the sideposted payload. The payload must have a temp-id that is unique within the entire reque
Update	Updates the relationship and sideposted resource. If only relationship data is provided, `update` will associate the sideposted data with the primary resource in the request. Requires a `id` in the sidepost payload
Destroy	Destroys the resource based on the id given in the relationship section of the payload.
Disassociate	Removes the association between the primary resource and the sideposted resource in the payload.

Understanding error codes

Below are many of the possible error statuses, codes, and messages you may receive from the API. Please note that while we consider changing or removing an error status or code a breaking change, the error message may change at any time and you should not rely on the message text.

Validation Errors

Validation errors when creating or updating a resource are returned in an `errors` array. A validation error will look something like this:

```
{ "errors": [ { "code": "unprocessable_entity", "status": "422", "title": "Validation Error
```

Non-Validation Errors

All other errors will return a body like:

```
{ "code": "not_found", "message": "Record not found" }
```

Below are the possible statuses and codes returned with these errors.

Error code	Type	Description
400	`bad_request`	Invalid include, filter value, attribute, sort, operator, filter, sideload, or other issue with the request.
400	`invalid_custom_field`	Your request had an unknown custom field.
401	`unauthorized`	You are not authorized to access this content. Your access token may be missing. The resource owner also may not have a permission level sufficient to grant access.
401	`token_expired`	Your access token has expired.
401	`token_upgrade_required`	A JWT is required to access this resource. API V1 tokens do not work with API V2.
404	`not_found`	Record not found. Check the ID.
422	`unprocessable_entity`	There was a validation error with the data provided in the request. These errors have a different body shape. See "Validation Errors" above.
500	`server_error`	You have encountered a server error. Double check your request and/or try again later.
502	`bad_gateway`	Backend service timed out.
503	`service_unavailable`	The service is temporarily unavailable. Please retry later. Note that you may see this error if sideloading too much data.

Troubleshooting FAQs

Data access constraints

Access to data through the API is governed by the resource owner's permission set. The data accessible to an API request is limited by the resource owner's permissions within the nation. When encountered missing data for a given API call, ensure that there has not been a recent change in the resource owner's permission level within that nation.

403 forbidden errors for valid access tokens

API applications may encounter a 403 despite the API access token's expiration date having not lapsed. The most common source of this issue is that the resource owner has lost access to the nation's control panel. When the resource owner loses access, their associated access token becomes unusable for API calls.

Related Articles

- NationBuilder API QuickStart Guide
- API v2 walkthrough
- Relationships between Resources in v2 API
- Using parameters to interact with the NationBuilder API
- Interactive API Documentation | Endpoint Explorer