A. How to use the atVenu Public API

Introduction

The atVenu public API is a GraphQL endpoint which enables customers to access to their data contained within the atVenu system. Each organization will be issued an access token that controls the data that can be accessed. An organization may have multiple tokens depending on the API data.

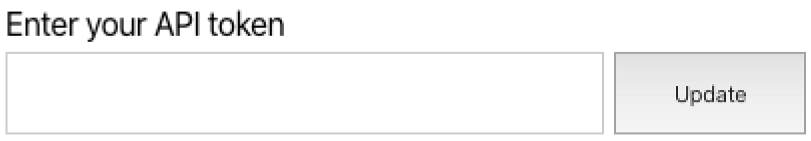
Please contact support for your API token(s). The token is an access key for an organizations data, be sure to keep it safe.

How to browse data via API Browser

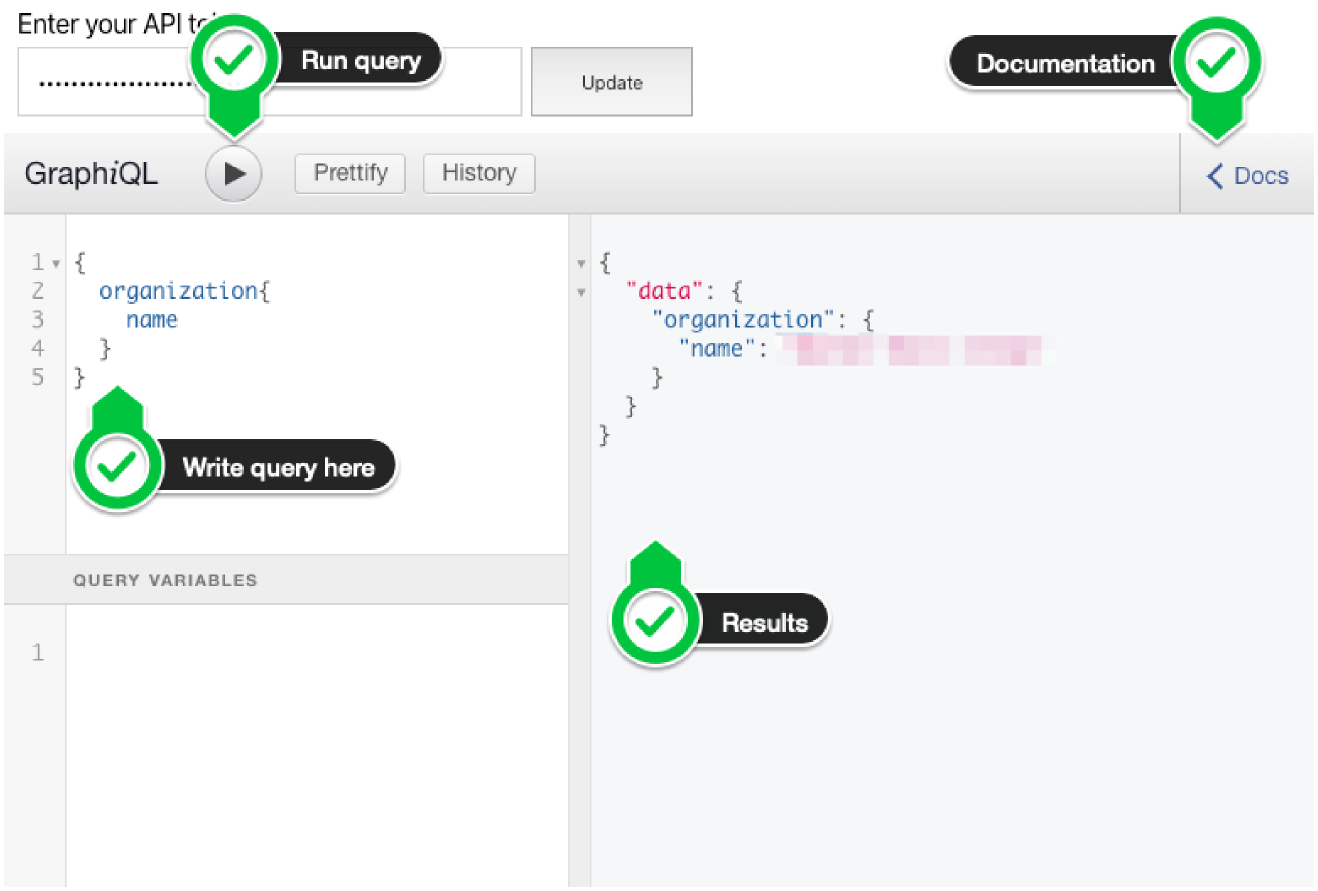
The API Browser is a hosted web page that allows you to experiment with the API viewing your organizations live data. It also contains the documentation on the fields available, and how to query for them.

This is live data, queries are monitored and may be throttled or limited based on complexity.

1. Navigate to https://api-browser.atvenu.com/
2. Login with your provided API token



3. Browse documentation, write queries and view your results



```
{
  organization{
    name
  }
}
```

```
{
  "data": {
    "organization": {
      "name": 
    }
  }
}
```

How to access the API directly

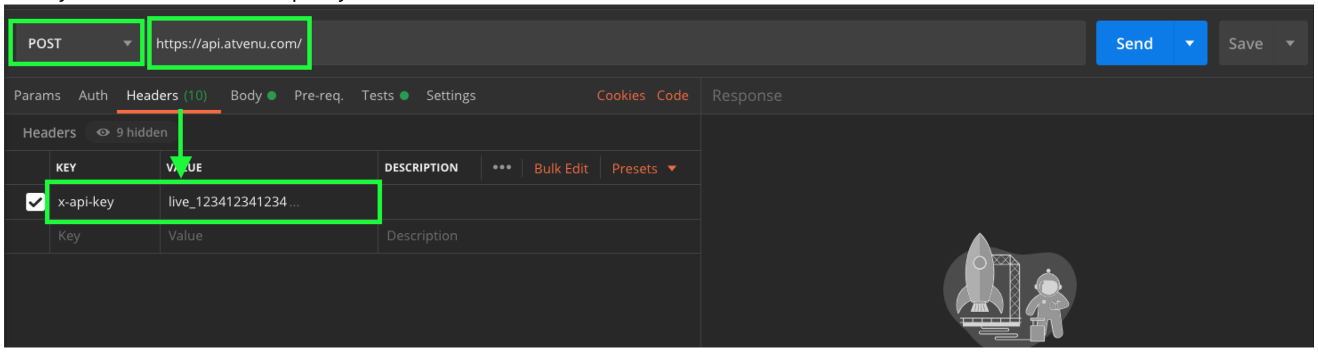
The endpoint for the API is available at `https://api.atvenu.com`.

Provide organization's token as part of the `x-api-key` header.

✓ You can query graphQL in any language. atVenu maintains an example implementation in Javascript (https://github.com/atvenu/atvenu-api-example) If you would like access, please provide a github account.

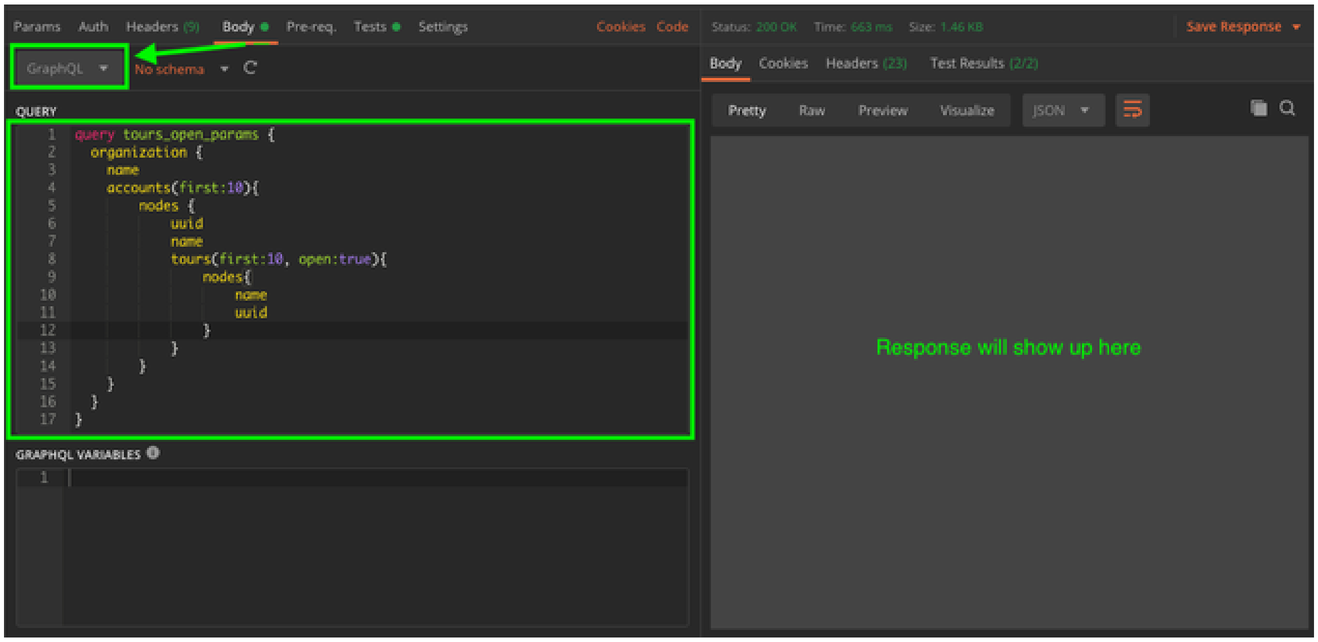
Example: Postman

1. Create a new **POST** request to https://api.atvenu.com/ .
2. Enter your API token as an x-api-key header.



	KEY	VALUE	DESCRIPTION
✓	x-api-key	live_123412341234...	
	Key	Value	Description

3. Provide your query and variables as a GraphQL request.



QUERY

```
query tours_open_params {
  organization {
    name
    accounts(first:10){
        nodes {
            uuid
            name
            tours(first:10, open:true){
                nodes{
                    name
                    uuid
                }
            }
        }
    }
  }
}
```

GRAPHQL VARIABLES

4. Hit ‘Send’ to query

How to page through collections

atVenu follows the GraphQL Connection specification for paging through large collections.

> API queries are limited due to live data requests. This may result in a different max page size.

The following is an example of how to page through all accounts, in graphiQL.

Query

1. Write a query to request data, and info on next page.

```
query getAccounts ($first: Int!, $after: String){
        organization{
            name
            accounts(first: $first, after: $after){
                pageInfo {
                    endCursor
                    hasNextPage
                }
                nodes{
                    uuid
                    name
                }
            }
        }
    }
```

2. Set variables to get the first page.

```
{
      "first": 20,
      "after": null
}
```

3. Run query and get the `endCursor` .

```
{
  "data": {
    "organization": {
      "name": "Your Cool Org",
      "accounts": {
        "pageInfo": {
          "endCursor": "ED=3",
          "hasNextPage": true
        },
        "nodes": [
          {
            "uuid": "acct_a1111-1111-1111-1111-11111111111",
            "name": "A Band"
          },
          ...
```

4. Use this `endCursor` value to update the variables to get the second page

```
{
      "first": 20,
      "after": "ED=3"
}
```

Paginating through collections ***nested*** within paginated collections is not recommended because you can only specify one cursor per collection for the next page, but a nested collection returns many end cursors.

How to access individual objects

This is a variation of the Global Object Identification specification.

Certain types implement the `NodeInterface` including:

- Account
- Tour
- Show

Location is not included at this time.

```
interface NodeInterface {
  "Unique atVenu ID"
  uuid: UUID
}
```

These types can be retrieved by selecting `node` with a `UUID` value on the root query type.

Use fragments to select fields specific to the type being retrieved.

Query

```
query {
  node(uuid: "show_XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX") {
    uuid
    ...on Show {
      showDate
    }
  }
}
```

Numerical Values

Whole Numbers

Values that represent whole numbers are of the built-in `Integer` type and are expressed as Integers in the atVenu API.

Decimal Numbers

Values that can represent fractional values are of the type `scalar Decimal` in the atVenu API. These values are strings expressed in exponential notation.

The atVenu API does not use the builtin `Float` type and discourages using floating point math in your integration. We suggest using an arbitrary arithmetic library.

Examples:

The value 123.456 would be returned as `"0.123456e3"`
The value 0 would be returned as `"0.0"`
The value 0.01 would be returned as `"0.1e-1"`
The value -50,000 would be returned as `"-0.5e5"`

Monetary Values and Currency Format

Monetary amounts are expressed as a `scalar Decimal` in the major unit of the amount's currency.

The atVenu platform is used worldwide with currencies that vary in scale and places kept after the decimal point. We use scientific notation and major currency units to avoid the ambiguity of the amount's value and discourage floating point conversions.

The currency format of the Show indicates the currency used to collect sales at the show. It also describes the formatting of the currency and the number of places after the decimal point retained (`subunits`).

Examples:

A net sold amount of $35.25 in USD would be returned as `"netSoldAmount": "0.3525e2"` in the atVenu API.
The currency format for USD has `2` subunits.

A net sold amount of ¥450 in JPY would be returned as `"netSoldAmount": "0.45e3"` in the atVenu API.
The currency format for JPY has `0` subunits.

Limits

atVenu imposes limits on requests to the API to protect the integrity of the system and may be changed at any time.

Response Time

All requests are limited to a maximum of `30` seconds to respond.

Query Depth

Queries have a maximum depth of `14`.

Complexity

Queries have a maximum complexity of `76 520`.

Complexity is calculated by the number of potential nodes returned by a paginated connection field or node query.

Every paginated connection requires `first` or `last` to be specified and limits the number of nodes returned. This limit plus the limit multiplied by the complexity of child nodes determines the complexity of the query.

Each query to `RootQueries.node` contributes 1 plus the complexity of child nodes.

Query

```
query getShows {
  organization {
    accounts(first: 20) {
      accountNodes: nodes {
        name
        uuid
        tours(first: 1) {
          tourNodes: nodes {
            shows(first: 50) {
              showNodes: nodes {
                showDate
                showEndDate
                uuid
              }
            }
          }
        }
      }
    }
  }
}
```

Variables

```
  20 Accounts
+ 20 * 1 Tour
+ 20 * 1 * 50 Shows
= 1040 Nodes
```

Connection Limits

The `first` and `last` arguments on connection fields are limited to a maximum of `200` except for `Organization.accounts` which are limited to a maximum of `20` and has a depth of `4` and a complexity of `9`.

Query

```
query getAccounts {
    organization{
        name
        accounts(first: 20){
            pageInfo {
                endCursor
                hasNextPage
            }
            nodes{
                uuid
                name
            }
        }
    }
}
```