Open States API Documentation

Release 1.0

Open States

Jan 23, 2020

Contents:

Documentation for Open States, its API, and how to contribute.

CHAPTER 1

Open States v2 GraphQL API

- If you are using the API please visit our Discourse and make use of our issue tracker to file comments/issues.
- API keys are required. You can register for an API key and once activated, you'll pass your API key via the `X-API-KEY` header.
- You can also check out our introductory blog post for more details.

1.1 Basics

This is a GraphQL API, and some of the concepts may seem unfamiliar at first.

There is in essence, only one endpoint: https://openstates.org/graphql.

This endpoint, when accessed in a browser, will provide an interface that allows you to experiment with queries in the browser, it features autocomplete and a way to browse the full graph (click the 'Docs' link in the upper right corner).

A GraphQL query mirrors the structure of the data that you'd like to obtain. For example, to obtain a list of legislators you'd pass something like:

```
{
    people {
        edges {
            node {
                name
            }
        }
    }
}
```

Note: If you are using the API programatically it is recommended you send the data as part of the POST body, e.g.:

```
curl -X POST https://openstates.org/graphql -d "query={people{edges{node{name}}}}"
```

Of course, if you try this you'll see it doesn't work since there are some basic limits on how much data you can request at once. We paginate with the `first`, `last`, `before` and `after` parameters to a root node. So let's try that again:

```
{
    people(first: 3) {
        edges {
            node {
                name
            }
        }
    }
}
```

And you'd get back JSON like:

```
{
  "data": {
    "people": {
      "edges": [
        {
          "node": {
            "name": "Lydia Brasch"
          }
        },
        {
          "node": {
            "name": "Matt Williams"
          }
        },
        {
          "node": {
            "name": "Merv Riepe"
          }
        }
      ]
    }
  }
}
```

Ah, much better. Nodes also can take other parameters to filter the returned content. Let's try the "name" filter which restricts our search to people named Lydia:

```
{
    people(first: 3, name: "Lydia") {
        edges {
            node {
                name
            }
        }
    }
}
```

Results in:

```
{
  "data": {
```

(continues on next page)

(continued from previous page)

```
    "people": {
      "edges": [
        {
          "node": {
            "name": "Lydia Brasch"
          }
        },
        {
          "node": {
            "name": "Lydia Graves Chassaniol"
          }
        },
        {
          "node": {
            "name": "Lydia C. Blume"
          }
        }
      ]
    }
  }
}
```

It is also possible to request data from multiple root nodes at once, for example:

```
{
    people(first: 1) {
        edges {
            node { name }
        }
    }
    bills(first: 1) {
        edges {
            node { title }
        }
    }
}
```

Would give back something like:

```
{
  "data": {
    "people": {
      "edges": [
        {
          "node": {
            "name": "Lydia Brasch"
          }
        }
      ]
    },
    "bills": {
      "edges": [
        {
          "node": {
            "title": "Criminal Law - Animal Abuse Emergency Compensation Fund -␣
↪Establishment"
          }
```

(continues on next page)

(continued from previous page)

```
        }
      ]
    }
  }
}
```

You may notice something here, that you get back just the data you need. This is extremely powerful, and lets you do the equivalent of many traditional API calls in a single query.

1.2 Full-fledged Example

Let's take a look at a more useful example:

```
{
  bill(jurisdiction: "New York", session: "2017-2018", identifier: "S 5772") {
    title
    actions {
      description
      date
    }
    votes {
      edges {
        node {
          counts {
            value
            option
          }
          votes {
            voterName
            voter {
              id
              contactDetails {
                value
                note
                type
              }
            }
            option
          }
        }
      }
    }
    sources {
      url
    }
    createdAt
    updatedAt
  }
}
```

There's a lot going on there, let's break it down:

```
bill(jurisdiction: "New York", session: "2017-2018", identifier: "S 5772") {
```

We're hitting the `bill` root node, which takes 3 parameters. This should get us to a single bill from New York.

```
title
```

This is going to give us the title, just like we saw before.

```
actions {
   description
   date
}
```

Here we're going into a child node, in this case all of the actions taken on the bill. For each action we're requesting a the date & description.

```
votes {
  edges {
    node {
```

Here too we're going into a child node, but note that this time we use that "edges" and "node" pattern that we see on root level nodes. Certain child nodes in the API have the ability to be paginated or further limited, and votes happen to be one of them. In this case however we're not making use of that so we'll just ignore this.

(A full discussion of this pattern is out of scope but check out the Relay pagination specification for more detail for more.)

```
  counts {
    value
    option
  }
  votes {
    voterName
    voter {
      id
      contactDetails {
        value
        note
        type
      }
    }
    option
  }
}
```

Here we grab a few more fields, including child nodes of each vote on our Bill.

First, we get a list of counts (essentially pairs of outcomes + numbers e.g. (yes, 31), (no, 5))

We also get individual legislator votes by name, and we traverse into another object to get the Open States ID and contact details for the voter. (Don't sweat the exact data model here, there will be more on the structure once we get to the actual graph documentation.)

```
sources {
  url
}
createdAt
updatedAt
```

And back up at the top level, we grab a few more pieces of information about the Bill.

And now you've seen a glimpse of the power of this API. We were able to get back theexact fields we wanted on a bill, contact information on the legislators that have voted on the bill, and more.

Our result looks like this:


```
{
  "data": {
    "bill": {
      "title": "Relates to bureaus of administrative adjudication",
      "actions": [
        {
          "description": "REFERRED TO LOCAL GOVERNMENT",
          "date": "2017-04-28"
        },
        {
          "description": "COMMITTEE DISCHARGED AND COMMITTED TO RULES",
          "date": "2017-06-19"
        },
        {
          "description": "ORDERED TO THIRD READING CAL.1896",
          "date": "2017-06-19"
        },
        {
          "description": "RECOMMITTED TO RULES",
          "date": "2017-06-21"
        }
      ],
      "votes": {
        "edges": [
          {
            "node": {
              "counts": [
                {
                  "value": 25,
                  "option": "yes"
                },
                {
                  "value": 0,
                  "option": "no"
                },
                {
                  "value": 0,
                  "option": "other"
                }
              ],
              "votes": [
                {
                  "voterName": "John J. Bonacic",
                  "voter": {
                    "id": "ocd-person/da013cd5-dc67-4e65-a310-73aa32ad1f7c"
                    "contactDetails": [
                        {
                          "value": "bonacic@nysenate.gov",
                          "note": "Capitol Office",
                          "type": "email"
                        },
                        {
                          "value": "Room 503\nAlbany, NY 12247",
                          "note": "District Office",
                          "type": "address"
                        },
```


(continues on next page)

(continued from previous page)


```
                        {
                          "value": "518-455-3181",
                          "note": "District Office",
                          "type": "voice"
                        },
                        ...etc...
                    ]
                  },
                  "option": "yes"
                },
                {
                  "voterName": "Neil D. Breslin",
                  "voter": {
                    "id": "ocd-person/4b710aee-1b99-42e0-90e2-d41338e8c5df"
                    "contactDetails": [ ...etc... ],
                  },
                  "option": "yes"
                },
                {
                  "voterName": "David Carlucci",
                  "voter": {
                    "id": "ocd-person/1b0feab9-02a7-4bcc-b089-3ab23286da68"
                    "contactDetails": [ ...etc... ],
                  },
                  "option": "yes"
                },
              ]
            }
          },
          ...etc...
        ]
      },
      "sources": [
        {
          "url": "http://legislation.nysenate.gov/api/3/bills/2017-2018/S5772?
↪summary=&detail="
        },
        {
          "url": "http://www.nysenate.gov/legislation/bills/2017/S5772"
        },
        {
          "url": "http://assembly.state.ny.us/leg/?default_fld=&bn=S5772&Summary=Y&
↪Actions=Y&Text=Y"
        }
      ],
      "createdAt": "2017-07-15 05:08:15.848526+00:00",
      "updatedAt": "2017-07-15 05:08:15.848541+00:00"
    }
  }
}
```

1.3 Further Details

1.3.1 Root Nodes

As seen in the introduction, when constructing a query you will start your query at one (or more) root nodes. The following root nodes are available:

jurisdictions

Get a list of all jurisdictions.

This will return a list of *JurisdictionNode* objects, one for each state (plus Puerto Rico and DC).

Pagination: This endpoint accepts the usual *Pagination* parameters, but pagination is not required.

people

Get a list of all people matching certain criteria.

This will return a list of *PersonNode* objects, one for each person matching your query.

Pagination: This endpoint accepts the usual *Pagination* parameters, and you must limit your results to no more than 100 using either the "first" or "last" parameter.

Parameters:

`name` Limit response to people who's name contains the provided string.

Includes partial matches & case-insensitive matches.

`memberOf` Limit response to people that have a currently active membership record for an organization. The value passed to memberOf can be an ocd-organization ID or a name (e.g. 'Republican' or 'Nebraska Legislature').

`everMemberOf` Limit response to people that have any recorded membership record for an organization. Operates as a superset of memberOf.

Specifying `memberOf` and `everMemberOf` in the same query is invalid.

`district` When specifying either memberOf or everMemberOf, limits to people who's membership represented the district with a given label. (e.g. memberOf: "Nebraska Legislature", district: "7")

Specifying `district` without `memberOf` or `everMemberOf` is invalid.

`latitude` and `longitude` Limit to people that are currently representing the district(s) containing the point specified by the provided coordinates.

Must be specified together.

bills

Get a list of all bills matching certain criteria.

This will return a list of *BillNode* objects, one for each person matching your query.

Pagination: This endpoint accepts the usual *Pagination* parameters, and you must limit your results to no more than 100 using either the "first" or "last" parameter.

Parameters:

jurisdiction Limit to bills associated with given jurisdiction, parameter can either be a human-readable jurisdiction name or an ocd-jurisdiction ID.

chamber Limit to bills originating in a given chamber. (e.g. upper, lower, legislature)

session Limit to bills originating in a given legislative session. This parameter should be the desired session's `identifier`. (See *LegislativeSessionNode*).

classification Limit to bills with a given classification (e.g. "bill" or "resolution")

subject Limit to bills with a given subject (e.g. "Agriculture")

searchQuery Limit to bills that contain a given term. (Experimental until 2020!)

updatedSince Limit to bills that have had data updated since a given time.

Time should be in the format YYYY-MM-DD[THH:MM:SS].

actionsSince Limit to bills that have had actions since a given time.

Time should be in the format YYYY-MM-DD.

jurisdiction

Look up a single jurisdiction by name or ID.

This will return a single *JurisdictionNode* object with the provided name or ID parameter.

Parameters:

name The human-readable name of the jurisdiction, such as 'New Hampshire'.

id The ocd-jurisdiction ID of the desired jurisdiction, such as 'ocd-jurisdiction/country:us/state:nh'.

You are required to provide one of the two available parameters.

person

Look up a single person by ocd-person ID.

This will return a single *PersonNode* by ID.

Parameters:

id ocd-person ID for the desired individual.

organization

Look up a single organization by ocd-organization ID.

This will return a single *OrganizationNode* by ID.

Parameters:

id ocd-organization ID for the desired individual.

bill

Look up a single bill by ID, URL, or (jurisdiction, session, identifier) combo.

This will return a single *BillNode* object with the specified bill.

Parameters:

id The ocd-bill ID of the desired bill, such as 'ocd-jurisdiction/country:us/state:nh'.

openstatesUrl The URL of the desired bill, such as 'https://openstates.org/nc/bills/2019/HB760/'.

jurisdiction, session, identifier Must be specified together to fully identify a bill.

As is true elsewhere, jurisdiction may be specified by name (New Hampshire) or ocd-jurisdiction ID (ocd-jurisdiction/country:us/state:nh).

Session is specified by legislative session identifier (e.g. 2018 or 49).

Identifier is the exact identifier of the desired bill, such as "HB 327".

You are required to provide one either `id` or the other parameters to fully specify a bill. Use `bills` if you are looking for something more broad.

1.3.2 Data Types

Starting at the base nodes, data in the API is represented as interconnected nodes of various types. This page provides an overview of the nodes.

Another good way to get acquainted with the layout is to use the GraphiQL browser (click Docs in the upper right corner).

Jurisdictions & Sessions

JurisdictionNode

A Jurisdiction is the Open Civic Data term for the top level divisions of the US. Open States is comprised of 52 jurisdictions, one for each state, and two more for D.C. and Puerto Rico.

Each JurisdictionNode has the following attributes & nodes available:

- `id` - ocd-jurisdiction identifier, these are permanent identifiers assigned to each Jurisdiction
- `name` - human-readable name for the jurisdiction (e.g. Kansas)
- `url` - URL of official website for jurisdiction
- `featureFlags` - reserved for future use
- `legislativeSessions` - Paginated list (see *Pagination*) of *LegislativeSessionNode* belonging to this jurisdiction's legislature.
- **`organizations` - Paginated list of *OrganizationNode* belonging to this jurisdiction.**

  - it is also possible to filter the list of children using the `classification` parameter

See also: Open Civic Data Jurisdiction reference

LegislativeSessionNode

A legislative session is a convening of the legislature, either a primary or special session.

Each LegislativeSessionNode has the following attributes and nodes available:

- `jurisdiction` - *JurisdictionNode* which this session belongs to.
- `identifier` - short identifier by which this session is referred to (e.g. 2017s1 or 121)
- `name` - formal name of session (e.g. "2017 Special Session #1" or "121st Session"
- `classification` - "primary" or "special"
- `startDate` - start date of session if known
- `endDate` - end date of session if known

DivisionNode

Divisions represent particular geopolitical boundaries. Divisions exist for states as well as their component districts and are tied closely to political geographies.

- `id` - Open Civic Data Division ID
- `name` - human-readable name for the division
- `redirect` - link to another DivisionNode, only present if division has been replaced
- `country` - country code (will be "us") for all Open States divisions
- `createdAt` - date at which this object was created in our system
- `updatedAt` - date at which this object was last updated in our system
- `extras` - JSON string with optional additional information about the object

People & Organizations

PersonNode

People, typically legislators and their associated metadata.

Note that most fields are optional beyond name as often we don't have a reliable given/family name or birthDate for instance.

- `id` - Open Civic Data Person ID
- `name` - primary name for the person
- `sortName` - alternate name to sort by (if known)
- `familyName` - hereditary name, essentially a "last name" (if known)
- `givenName` - essentially a "first name" (if known)
- `image` - full URL to official image of legislator

- `birthDate` - see *Fuzzy Date Format*
- `deathDate` - see *Fuzzy Date Format*
- `identifiers` - list of other known identifiers, *IdentifierNode*
- `otherNames` - list of other known names, *NameNode*
- `links` - official URLs relating to this person, *LinkNode*
- `contactDetails` - ways to contact this person (via email, phone, etc.), *contactdetailnode*
- **`currentMemberships` - currently active memberships *MembershipNode***
  - can be filtered with the `classification` parameter to only get memberships to certain types of *OrganizationNode*
- **`oldMemberships` - inactive memberships *MembershipNode***
  - can be filtered with the `classification` parameter to only get memberships to certain types of *OrganizationNode*
- `sources` - URLs which were used in compiling Open States' information on this subject, *LinkNode*
- `createdAt` - date at which this object was created in our system
- `updatedAt` - date at which this object was last updated in our system
- `extras` - JSON string with optional additional information about the object

See also:

- Popolo's person
- Open Civic Data OCDEP 5

OrganizationNode

Organizations that comprise the state legislatures and their associated metdata.

A typical bicameral legislature is comprised of a top-level organization (classification=legislature), two chambers (classification=upper & lower), and any number of committees (classification=committee).

Each Organization is comprised of the following attributes and nodes:

- `id` - Open Civic Data Organization ID
- `name` - primary name for the person
- `image` - full URL to official image for organization
- `classification` - the type of organization as described above
- `foundingDate` - see *Fuzzy Date Format*
- `dissolutionDate` - see *Fuzzy Date Format*
- `parent` - parent OrganizationNode if one exists
- **`children` - paginated list of child OrganizationNode objects**
  - it is also possible to filter the list of children using the `classification` parameter
- `currentMemberships` - list of all current members of this Organization
- `identifiers` - list of other known identifiers for this organization, *IdentifierNode*
- `otherNames` - list of other known names for this organization, *NameNode*

- `links` - official URLs relating to this person, *LinkNode*
- `sources` - URLs which were used in compiling Open States' information on this subject, *LinkNode*
- `createdAt` - date at which this object was created in our system
- `updatedAt` - date at which this object was last updated in our system
- `extras` - JSON string with optional additional information about the object

See also:

- Popolo's organization
- Open Civic Data OCDEP 5

MembershipNode

A MembershipNode represents a connection between a *personnode* and a *organizationnode*. A membership may optionally also reference a particular *postnode*, such as a particular seat within a given chamber.

Each membership has the following attributes and nodes:

- `id` - Open Civic Data Membership ID
- `personName` the raw name of the person that the membership describes (see *Name Matching*)
- `person` - *personnode*
- `organization` - *organizationnode*
- `post` - *postnode*
- `label` - label assigned to this membership
- `role` - role fulfilled by this membership
- `startDate` - start date of membership if known
- `endDate` - end date of membership if known
- `createdAt` - date at which this object was created in our system
- `updatedAt` - date at which this object was last updated in our system
- `extras` - JSON string with optional additional information about the object

See also:

- Popolo's membership
- Open Civic Data OCDEP 5

PostNode

A PostNode represents a given position within an organization. The most common example would be a seat such as Maryland's 4th House Seat.

It is worth noting that some seats can have multiple active memberships at once, as noted in `maximumMemberships`.

Each post has the following attributes and nodes:

- `id` - Open Civic Data Post ID
- `label` - label assigned to this post (e.g. 3)

- `role` - role fulfilled by this membership (e.g. 'member')
- `division` - related *divisionnode* if this role has a relevant division
- `startDate` - start date of membership if known
- `endDate` - end date of membership if known
- `maximumMemberships` - typically 1, but set higher in the case of multi-member districts
- `createdAt` - date at which this object was created in our system
- `updatedAt` - date at which this object was last updated in our system
- `extras` - JSON string with optional additional information about the object

See also:

- Popolo's post
- Open Civic Data OCDEP 5

Bills & Votes

BillNode

A BillNode represents any legislative instrument such as a bill or resolution.

Each node has the following attributes and nodes available:

- `id` - Internal ocd-bill identifier for this bill.
- `legislativeSession` - link to *LegislativeSessionNode* this bill is from
- `identifier` - primary identifier for this bill (e.g. HB 264)
- `title` - primary title for this bill
- `fromOrganization` - organization (typically upper or lower chamber) primarily associated with this bill
- `classification` - list of one or more bill types such as "bill" or "resolution"
- `subject` - list of zero or more subjects assigned by the state
- `abstracts` - list of abstracts provided by the state, *BillAbstractNode*
- `otherTitles` - list of other titles provided by the state, *BillTitleNode*
- `otherIdentifiers` - list of other identifiers provided by the state, *BillIdentifierNode*
- `actions` - list of actions (such as introduction, amendment, passage, etc.) that have been taken on the bill, *BillActionNode*
- `sponsorships` - list of bill sponsors, *BillSponsorshipNode*
- `relatedBills` - list of related bills as provided by the state, *RelatedBillNode*
- `versions` - list of bill versions as provided by the state, *BillDocumentNode*
- `documents` - list of related documents (e.g. legal analysis, fiscal notes, etc.) as provided by the state, *BillDocumentNode*
- `votes` - paginated list of *VoteEventNode* related to the bill
- `sources` - URLs which were used in compiling Open States' information on this subject, *linknode*
- `openstatesUrl` - URL to bill page on OpenStates.org

- `createdAt` - date at which this object was created in our system
- `updatedAt` - date at which this object was last updated in our system
- `extras` - JSON string with optional additional information about the object

BillAbstractNode

Represents an official abstract for a bill, each BillAbstractNode has the following attributes:

- `abstract` - the abstract itself
- `note` - optional note about origin/purpose of abstract
- `date` - optional date associated with abstract

BillTitleNode

Represents an alternate title for a bill, each BillTitleNode has the following attributes:

- `title` - the alternate title
- `note` - optional note about origin/purpose of this title

BillIdentifierNode

Represents an alternate identifier for a bill, each BillIdentifierNode has the following attributes:

- `identifier` - the alternate identifier
- `scheme` - a name for the identifier scheme
- `note` - optional note about origin/purpose of this identifier

BillActionNode

Represents an action taken on a bill, each BillActionNode has the following attributes and nodes:

- `organization` - *OrganizationNode* where this action originated, will typically be either upper or lower chamber, or perhaps legislature as a whole.
- `description` - text describing the action as provided by the jurisdiction.
- `date` - date action took place (see *Fuzzy Date Format*)
- `classification` - list of zero or more normalized action types (see *Action Types*)
- `order` - integer by which actions can be sorted, not intended for display purposes
- `extras` - JSON string providing extra information about this action
- `vote` - if there is a known associated vote, pointer to the relevant *VoteEventNode*
- `relatedEntities` - a list of *RelatedEntityNode* with known entities referenced in this action

RelatedEntityNode

Represents an entity that is related to a *BillActionNode*.

- `name` - raw (source-provided) name of entity
- `entityType` - either organization or person
- `organization` - if `entityType` is 'organization', the resolved *OrganizationNode*
- `person` - if `entityType` is 'person', the resolved *PersonNode*

See *Name Matching* for details on how `name` relates to `organiation` and `person`.

BillSponsorshipNode

Represents a sponsor of a bill.

- `name` - raw (source-provided) name of sponsoring person or organization
- `entityType` - either organization or person
- `organization` - if `entityType` is 'organization', the resolved *OrganizationNode*
- `person` - if `entityType` is 'person', the resolved *PersonNode*
- `primary` - boolean, true if sponsorship is considered by the jurisdiction to be "primary" (note: in many states multiple primary sponsors may exist)
- `classification` - jurisdiction-provided type of sponsorship, such as "author" or "cosponsor". These meanings typically vary across states, which is why we provide `primary` as a sort of indicator of the degree of sponsorship indicated.

See *Name Matching* for details on how `name` relates to `organiation` and `person`.

RelatedBillNode

Represents relationships between bills.

- `identifier` - identifier of related bill (e.g. SB 401)
- `legislativeSession` - identifier of related session (in same jurisdiction)
- `relationType` - type of relationship such as "companion", "prior-session", "replaced-by", or "replaces"
- `relatedBill` - if the related bill is found to exist in our data, link to the *BillNode*

BillDocumentNode

Representation of `documents` and `versions` on bills. A given document can have multiple links representing different manifestations (e.g. HTML, PDF, DOC) of the same content.

- `note` - note describing the purpose of the document or version (e.g. Final Printing)
- `date` - optional date associated with the document
- `links` - list of one or more `MimetypeLinkNode` with actual URLs to bills.

MimetypeLinkNode

Represents a single manifestation of a particular document.

- `mediaType` - media type (aka MIME type) such as application/pdf or text/html
- `url` - URL to official copy of the bill
- `text` - text describing this particular manifestation (e.g. PDF)

VoteEventNode

Represents a vote taken on a bill.

- `id` - Internal ocd-vote identifier for this bill.
- `identifier` - Identifier used by jurisdiction to uniquely identify the vote.
- `motionText` - Text of the motion being voted upon, such as "motion to pass the bill as amended."
- `motionClassification` - List with zero or more classifications for this motion, such as "passage" or "veto-override"
- `startDate` - Date on which the vote took place. (see *Fuzzy Date Format*)
- `result` - Outcome of the vote, 'pass' or 'fail'.
- `organization` - Related *OrganizationNode* where vote took place.
- `billAction` - Optional linked *BillActionNode*.
- `votes` - List of *PersonVoteNode* for each individual's recorded vote. (May not be present depending on jurisdiction.)
- `counts` - List of *VoteCountNode* with sums of each outcome (e.g. yea/nay/abstain).
- `sources` - URLs which were used in compiling Open States' information on this subject, *LinkNode*
- `createdAt` - date at which this object was created in our system
- `updatedAt` - date at which this object was last updated in our system
- `extras` - JSON string with optional additional information about the object

See also: Open Civic Data vote format.

PersonVoteNode

Represents an individual person's vote (e.g. yea or nay) on a given bill.

- `option` - Option chosen by this individual. (yea, nay, abstain, other, etc.)
- `voterName` - Raw name of voter as provided by jurisdiction.
- `voter` - Resolved *PersonNode* representing voter. (See *Name Matching*)
- `note` - Note attached to this vote, sometimes used for explaining an "other" vote.

VoteCountNode

Represents the sum of votes for a given `option`.

- `option` - Option in question. (yea, nay, abstain, other, etc.)
- `value` - Number of individuals voting this way.

Other Nodes

IdentifierNode

Represents an alternate identifier, each with the following attributes:

- `identifier` - the alternate identifier
- `scheme` - a name for the identifier scheme

NameNode

Represents an alterante name, each with the following attributes:

- `name` - the alternate name
- `note` - note about usage/origin of this alternate name
- `startDate` - date at which this name began being valid (blank if unknown)
- `endDate` - date at which this name stopped being valid (blank if unknown or still active)

LinkNode

Represents a single link associated with a person or used as a source.

- `url` - URL
- `text` - text describing the use of this particular URL

ContactDetailNode

Used to represent a contact method for a given person.

- `type` - type of contact detail (e.g. voice, email, address, etc.)
- `value` - actual phone number, email address, etc.
- `note` - used to group contact data by location (e.g. Home Address, Office Address)
- `label` - human-readable label for this contact detail

1.3.3 Other Notes

There are a few other things to be aware of while using the API:

Explore the Graph

GraphQL is still quite new, so we figured it might be good to provide some helpful tips on how to think about the data and how you'll use the API.

First, it is probably well worth your time to play around in GraphiQL to explore the API and data. It was heavily used when developing the API and writing tests, and is a very powerful tool, particularly when you make use of the self-documenting nature of the graph.

When you're thinking about how to query don't necessarily try to replicate your old API calls exactly. For example, perhaps you were grabbing all bills that met a given criteria and then grabbing all sponsors contact details. This can now be done in one call by traversing from the *bills* root node into the *BillSponsorshipNode* and then up to the *PersonNode* and finally to the *ContactDetailNode*. This may sound complex at first, but once you get the hang of it, it really does unlock a ton of power and will make your apps more powerful and efficient.

Pagination

In several places (such as the *bills* and *BillNode*'s `votes`) we mention that nodes are paginated.

What this means in practice is that instead of getting back the underlying node type, say `BillNode`, directly, you'll get back `BillConnectionNode` or similar. (In practice there are connection node types for each paginated type, but all work the same way in our case.)

Arguments

Each paginated endpoint accepts any of four parameters:

- `first` - given an integer N, only return the first N nodes
- `last` - given an integer N, only return the last N nodes
- `after` - combined with `first`, will return first N nodes after a given "cursor"
- `before` - combined with `last`, will return last N nodes before a given "cursor"

So typically you'd paginate using `first`, obtaining a cursor, and then calling the API again with a combination of `first` and `after`.

The same process could be carried out with `last` and `before` to paginate in reverse.

Responses

Let's take a look at everything that pagination makes available:

```
{
  bills(first:20) {
    edges {
      node {
        title
      }
      cursor
    }
    pageInfo {
      hasNextPage
      hasPreviousPage
      endCursor
```

(continues on next page)

(continued from previous page)

```
      startCursor
    }
    totalCount
  }
}
```

You'll see that the connection node has three nodes: `edges`, `pageInfo`, and `totalCount`

- **`edges` - a list of objects that each have a `node` and `cursor` attribute:**
  - `node` - the underlying node type, in our case `BillNode`
  - `cursor` - an opaque cursor for this particular item, it can be used with the `before` and `after` parameters each paginated node accepts as arguments.
- **`pageInfo` - a list of helpful pieces of information about this page:**
  - `hasNextPage` - boolean that is true if there is another page after this
  - `hasPreviousPage` - boolean that is true if there is a page before this
  - `endCursor` - last cursor in the set of edges, can be used with `after` to paginate forward
  - `startCursor` - first cursor in the set of edges, can be used with `before` to paginate backwards
- `totalCount` - total number of objects available from this connection

In Practice

Let's say you want to get all of the people matching a given criteria:

You'd start with a query for all people matching your criteria, ensuring to set the page size to no greater than the maximum:

```
{
    people(memberOf: "Some Organization", first: 100) {
        edges {
            node {
                name
            }
        }
        pageInfo {
            hasNextPage
            endCursor
        }
    }
}
```

Let's say we got back a list of 100 edges and our `pageInfo` object looked like:

```
{
    "hasNextPage": true,
    "endCursor": "ZXJyYXlxb20uZWN0aW9uOjA="
}
```

So you'd make another call:

```
{
    people(memberOf: "Some Organization", first: 100, after:"ZXJyYXlxb20uZWN0aW9uOjA=
↪" ) {
        edges {
            node {
                name
            }
        }
        pageInfo {
            hasNextPage
            endCursor
        }
    }
}
```

And let's say in this case you got back only 75 edges, and our `pageInfo` object looks like:

```
{
    "hasNextPage": false,
    "endCursor": "AXjYylxX2bu1wxa9uunnb="
}
```

We'd stop iteration at this point, of course, if hasNextPage had been true, we'd continue on until it wasn't.

Renaming fields

A really useful trick that is often overlooked is that you can rename fields when retrieving them, for example:

```
{
  republicans: people(memberOf: "Republican", first: 5) {
    edges {
      node {
        full_name: name
      }
    }
  }
}
```

Would give back:

```
{
  "data": {
    "republicans": {
      "edges": [
        {
          "node": {
            "full_name": "Michelle Udall"
          }
        },
        {
          "node": {
            "full_name": "Kimberly Yee"
          }
        },
        {
          "node": {
```

(continues on next page)

(continued from previous page)

```
            "full_name": "Regina E. Cobb"
          }
        },
        {
          "node": {
            "full_name": "Michelle B. Ugenti-Rita"
          }
        },
        {
          "node": {
            "full_name": "David Livingston"
          }
        }
      ]
    }
  }
}
```

Note that we're both renaming a top-level node here as well as a piece of data within the query.

You can also use this to query the same root node twice (doing so without renaming isn't allowed since it results in a name conflict).

For example:

```
{
  republicans: people(memberOf: "Republican", first: 5) {
    edges {
      node {
        full_name: name
      }
    }
  }
  democrats: people(memberOf: "Democratic", first: 5) {
    edges {
      node {
        full_name: name
      }
    }
  }
}
```

Fuzzy Date Format

Unless otherwise noted (most notably `createdAt` and `updatedAt` all date objects are "fuzzy". Instead of being expressed as an exact date, it is possible a given date takes any of the following formats:

- YYYY
- YYYY-MM
- YYYY-MM-DD
- YYYY-MM-DD HH:MM:SS (if times are allowed)

Name Matching

In several places such as bill sponsorships and votes you'll notice that we have a raw string representing a person or organization as well as a place for a link to the appropriate *OrganizationNode* or *PersonNode*.

Because of the way we collect the data from states, we always collect the raw data and later make an attempt to (via a mix of automated matching and manual fixes) connect the reference with data we've already collected.

In many cases these linkages will not be provided, but with some upcoming new tools to help us improve this matching we'll be able to dramatically improve the number of matched entities in the near future.

1.3.4 Examples

Get basic information for all legislatures

See in GraphiQL

```
{
  jurisdictions {
    edges {
      node {
        name
        legislativeSessions {
          edges {
            node {
              name
            }
          }
        }
        legislature: organizations(classification: "legislature", first: 1) {
          edges {
            node {
              name
              classification
              children(first: 5) {
                edges {
                  node {
                    name
                    classification
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

Get overview of a legislature's structure

See in GraphiQL

```
{
  jurisdiction(name: "North Dakota") {
    name
    url
    legislativeSessions {
      edges {
        node {
          name
          identifier
        }
      }
    }
    organizations(classification: "legislature", first: 1) {
      edges {
        node {
          id
          name
          children(first: 20) {
            edges {
              node {
                name
              }
            }
          }
        }
      }
    }
  }
}
```

Search for bills that match a given condition

See in GraphiQL

```
{
  search_1: bills(first: 5, jurisdiction: "New York", session: "2017-2018", chamber:
↪"lower", classification: "resolution", updatedSince: "2017-01-15") {
    edges {
      node {
        id
        identifier
        title
        classification
        updatedAt
        createdAt
        legislativeSession {
          identifier
          jurisdiction {
            name
          }
        }
        actions {
          date
          description
          classification
        }
```

(continues on next page)

(continued from previous page)

```
        documents {
          date
          note
          links {
            url
          }
        }
        versions {
          date
          note
          links {
            url
          }
        }

        sources {
          url
          note

        }
      }
    }
  }
}
```

Get all information on a particular bill

See in GraphiQL

```
{
  b1: bill(jurisdiction: "Hawaii", session: "2017 Regular Session", identifier: "HB␣
↪475") {
    id
    identifier
    title
    classification
    updatedAt
    createdAt
    legislativeSession {
      identifier
      jurisdiction {
        name
      }
    }
    actions {
      date
      description
      classification
    }
    documents {
      date
      note
      links {
        url
      }
```

(continues on next page)

(continued from previous page)

```
    }
    versions {
      date
      note
      links {
        url
      }
    }
    sources {
      url
      note
    }
  }
  b2: bill(id: "ocd-bill/9c24aaa2-6acc-43ad-883b-ae9f677062e9") {
    id
    identifier
    title
    classification
    updatedAt
    createdAt
    legislativeSession {
      identifier
      jurisdiction {
        name
      }
    }
    actions {
      date
      description
      classification
    }
    documents {
      date
      note
      links {
        url
      }
    }
    versions {
      date
      note
      links {
        url
      }
    }
    sources {
      url
      note
    }
  }
}
```

Get information about a specific legislator

See in GraphiQL

```
{
  person(id:"ocd-person/dd05bd23-fe49-4e65-bfff-62db997e56e0"){
    name
    contactDetails {
      note
      type
      value
    }
    otherNames {
      name
    }
    sources {
      url
    }
    currentMemberships {
      organization {
        name
      }
    }
  }
}
```

Get legislators for a given state/chamber

See in GraphiQL

`ocd-organization/ddf820b5-5246-46b3-a807-99b5914ad39f` is the id of the Alabama Senate chamber.

```
{
  people(memberOf:"ocd-organization/ddf820b5-5246-46b3-a807-99b5914ad39f", first:␣
↪100) {
    edges {
      node {
        name
        party: currentMemberships(classification:"party") {
          organization {
            name

          }
        }
        links {
          url
        }
        sources {
          url
        }
        chamber: currentMemberships(classification:["upper", "lower"]) {
          post {
            label
          }
          organization {
            name
            classification
            parent {
              name
```

(continues on next page)

(continued from previous page)

```
            }
          }
        }
      }
    }
  }
}
```

Search for legislators that represent a given area

See in GraphQL

```
{
  people(latitude: 40.7460022, longitude: -73.9584642, first: 100) {
    edges {
      node {
        name
        chamber: currentMemberships(classification:["upper", "lower"]) {
          post {
            label
          }
          organization {
            name
            classification
            parent {
              name
            }
          }
        }
      }
    }
  }
}
```

1.3.5 Changelog

Changelog for Open States GraphQL API:

v2.3 (August 2019)

- add experimental full text search via searchQuery parameter to bills node

v2.2 (June 2019)

- add openstatesUrl to bills query
- speed improvments

v2.1 (Feb 2019)

- fix lat-lon behavior to limit to active memberships

- improve handling of retired legislators
- fix type of maximum_memberships
- bill version ordering is now consistent

v2.0 (January 2019)

- bugfix for maximum_memberships type
- bugfix for versions field
- improve tests

Beta Release (November 2018)

- **API Keys are now required**
- consider classification when using current_memberships
- fix geo filtering
- add openstatesUrl to Bill node for ease of linkage to OpenStates.org
- add Person.oldMemberships as analog to currentMemberships
- add actionSince filter to bills node
- fix 500 errors/optimization when using GraphQL fragments
- addition of basic protection for excessive queries
- add totalCount to assist in pagination
- add Organization.currentMemberships

Preview Release 1 (May 2018)

- fix for people pagination
- add updatedSince for people
- add sponsor argument for bills node
- allow votes to take pagination parameters
- allow traversing to votes from person

Preview Release 0 (Dec 2017)

Initial draft release of the API, no backwards-compatibility guarantee made.

CHAPTER 2

Open States API v1

Note: API v2 is now available, please check out API v2.

Open States provides a JSON API for accessing state legislative information.

2.1 Basics

- All API calls are URLs in the form `https://openstates.org/api/v1/METHOD/`
- Responses are JSON unless otherwise specified.
- If an error occurs the response will be a plain text error message with an appropriate HTTP error code (404 if object is not found, 401 if authentication fails, etc.).
- To use the API you must register for an API key.
- Once activated, pass your API key via the `apikey` query paramter or the `X-API-KEY` header.
- All changes to the API will be announced on the Open States Discourse. It is recommended you subscribe if you're using the API.
- For Python users, there's an official pyopenstates package available.

2.2 Deprecation

As of November 2018 we are beginning the process of removing lesser-used portions of this API. New applications should use API v2.

None of these were used by more than 1% of API users, and their removal will help us hopefully extend the life of the existing API.

The Event & Committee endpoints are no longer supported. Committees are available in API v2, and events are not currently collected part of Open States.

Additionally, **the boundary endpoint will no longer be something we provide**. This data changes rarely and is best served from a static resource instead of an HTTP query. We are exploring options to provide a suitable resource for this in the future.

Additionally, the following rarely-used parameters are no longer supported in v1, similar functionality is available in v2:

Bill Search:

- sponsor_id
- subject
- type
- bill_id__in
- sort=signed, sort=passed_upper, sort=passed_lower

Legislator Search:

- first_name
- last_name
- active=false
- party
- term

If you were using the fields= parameter to control which data was returned you'll find that it is no longer always respected. To ease caching we have altered the behavior, this was designed to be done in a backwards-compatible way: we now return more fields than we used to by default, and only in select cases is it necessary to add fields= to a request to obtain fields that would otherwise be omitted.

Additionally, the extra fields prefixed with + are scheduled to be removed. They were never guaranteed, and we presume this won't affect any users.

2.3 Deprecation of Identifiers

Old Open States IDs (resembling AKL000001 or NYB00012345) will continue to work for entities created before 2019, but going forward will not be available. In most cases this shouldn't be an issue, since the bill detail endpoint supports the <state>/<session>/<identifier> format. Right now we have no plans to bring v1 IDs back.

2.4 Data Types

Open States provides data about six core data types.

State Metadata Details on what data is available, including terms, sessions, and state-specific names for things.

Bills Details on bills & resolutions, including actions & votes.

Legislators Details on legislators, including contact details.

Districts Details on districts and their boundaries.

2.4.1 State Metadata

Note: API v2 is now available, please check out API v2.

Metadata Overview Get list of all states with data available and basic metadata about their status.

State Metadata Get detailed metadata for a particular state.

Metadata Fields

The following fields are available on metadata objects:

- `abbreviation` The two-letter abbreviation of the state.
- `capitol_timezone` Timezone of state capitol (e.g. 'America/New_York')
- `chambers` Dictionary mapping chamber type (upper/lower) to an object with the following fields:
  - `name` Short name of the chamber (e.g. 'House', 'Senate')
  - `title` Title of legislators in this chamber (e.g. 'Senator')
- `feature_flags` A list of which optional features are available, options include:
  - 'subjects' - bills have categorized subjects
  - 'influenceexplorer' - legislators have influence explorer ids
  - 'events' - event data is present
- `latest_csv_date` Date that the CSV file at `latest_csv_url` was generated.
- `latest_csv_url` URL from which a CSV dump of all data for this state can be obtained.
- `latest_json_date` Date that the JSON file at `latest_json_url` was generated.
- `latest_json_url` URL from which a JSON dump of all data for this state can be obtained.
- `latest_update` Last time a successful scrape was run.
- `legislature_name` Full name of legislature (e.g. 'North Carolina General Assembly')
- `legislature_url` URL to legislature's official website.
- `name` Name of state.
- `session_details` Dictionary of session names to detail dictionaries with the following keys:
  - `type` 'primary' or 'special'
  - `display_name` e.g. '2009-2010 Session'
  - `start_date` date session began
  - `end_date` date session began
- `terms` List of terms in order that they occurred. Each item in the list is comprised of the following keys:
  - `start_year` Year session started.
  - `end_year` Year session ended.
  - `name` Display name for term (e.g. '2009-2011').
  - `sessions` List of sessions (e.g. '2009'). Each session will be present in `session_details`.

Terms & Sessions

A common area for confusion, terms describe a period of time between legislative elections, for example '2009-2010'. A term can be comprised of one or more sessions:depending on how often the legislature met/adjourned within the term.

Terms are associated with legislators, while sessions are associated with bills.

Methods

Metadata Overview

This method returns just a subset (`abbreviation`, `name`, `chambers`, `feature_flags`) of metadata across all available entities.

Example: *openstates.org/api/v1/metadata/*

State Metadata

This method returns the full metadata for a state.

Example: *openstates.org/api/v1/metadata/nc/*

Examples

Metadata Overview

`openstates.org/api/v1/metadata/`


```
[
 { "name": "Alabama",
  "abbreviation": "al",
  "feature_flags": [ "subjects", "influenceexplorer" ],
  "chambers": {
   "upper": { "name": "Senate", "title": "Senator" },
   "lower": { "name": "House", "title": "Representative" }
  } },
 { "name": "Alaska",
  "abbreviation": "ak",
  "feature_flags": [ "subjects", "influenceexplorer" ],
  "chambers": {
   "upper": { "name": "Senate", "title": "Senator" },
   "lower": { "name": "House", "title": "Representative" }
  } },
 { "name": "Arizona",
  "abbreviation": "az",
  "feature_flags": [ "events", "influenceexplorer" ],
  "chambers": {
   "upper": { "name": "Senate", "title": "Senator" },
   "lower": { "name": "House", "title": "Representative" }
  } },
 { "name": "Arkansas",
  "abbreviation": "ar",
```


(continues on next page)

(continued from previous page)


```
  "feature_flags": [ "influenceexplorer" ],
  "chambers": {
   "upper": { "name": "Senate", "title": "Senator" },
   "lower": { "name": "House", "title": "Representative" }
  } },
 { "name": "California",
  "abbreviation": "ca",
  "feature_flags": [ "subjects", "influenceexplorer" ],
  "chambers": {
   "upper": { "name": "Senate", "title": "Senator" },
   "lower": { "name": "Assembly", "title": "Assemblymember" }
  } },
 { "name": "Colorado",
  "abbreviation": "co",
  "feature_flags": [ "influenceexplorer" ],
  "chambers": {
   "upper": { "name": "Senate", "title": "Senator" },
   "lower": { "name": "House", "title": "Representative" }
  } },
 { "name": "Connecticut",
  "abbreviation": "ct",
  "feature_flags": [ "subjects", "events", "influenceexplorer" ],
  "chambers": {
   "upper": { "name": "Senate", "title": "Senator" },
   "lower": { "name": "House", "title": "Representative" }
  } },
 { "name": "Delaware",
  "abbreviation": "de",
  "feature_flags": [ "events", "influenceexplorer" ],
  "chambers": {
   "upper": { "name": "Senate", "title": "Senator" },
   "lower": { "name": "House", "title": "Representative" }
  } },
 { "name": "District of Columbia",
  "abbreviation": "dc",
  "feature_flags": [],
  "chambers": {
   "upper": { "name": "Council", "title": "Councilmember" }
  } },
  ...truncated...
]
```


State Metadata

`openstates.org/api/v1/metadata/nc/`


```
{
 "abbreviation": "nc",
 "capitol_timezone": "America/New_York",
 "chambers": {
  "upper": { "name": "Senate", "title": "Senator" },
  "lower": { "name": "House", "title": "Representative" }
 },
 "feature_flags": [ "subjects", "influenceexplorer" ],
 "id": "nc",
```


(continues on next page)

(continued from previous page)


```
 "latest_csv_date": "2013-03-01 09:04:45",
 "latest_csv_url": "http://static.openstates.org/downloads/2013-03-01-nc-csv.zip",
 "latest_json_date": "2013-03-05 23:46:34",
 "latest_json_url": "http://static.openstates.org/downloads/2013-03-05-nc-json.zip",
 "latest_update": "2013-03-24 01:38:51",
 "legislature_name": "North Carolina General Assembly",
 "legislature_url": "http://www.ncleg.net/",
 "name": "North Carolina",
 "session_details": {
  "2009": { "type": "primary", "display_name": "2009-2010 Session", "start_date":
↪"2009-01-28 00:00:00" },
  "2011": { "type": "primary", "display_name": "2011-2012 Session", "start_date":
↪"2011-01-26 00:00:00" },
  "2013": { "type": "primary", "display_name": "2013-2014 Session", "start_date":
↪"2013-01-30 00:00:00" }
 },
 "terms": [
  { "end_year": 2010, "start_year": 2009, "name": "2009-2010", "sessions": [ "2009" ]␣
↪},
  { "end_year": 2012, "start_year": 2011, "name": "2011-2012", "sessions": [ "2011" ]␣
↪},
  { "end_year": 2014, "start_year": 2013, "name": "2013-2014", "sessions": [ "2013" ]␣
↪}
 ]
}
```


2.4.2 Bills

Note: API v2 is now available, please check out API v2.

Bill Search Search bills by (almost) any of their attributes, or full text.

Bill Detail Get full detail for bill, including any actions, votes, etc.

Bill Fields

The following fields are available on bill objects:

- `state` State abbreviation.
- `session` Session key (see *State Metadata* for details).
- `bill_id` The official id of the bill (e.g. 'SB 27', 'A 2111')
- `title` The official title of the bill. Bill titles vary widely in size and content by state. Some are over 10KB long, while others are a few non-specific words, e.g. "Regarding taxes".
- `alternate_titles` List of alternate titles that the bill has had. (Often empty.)
- `action_dates` Dictionary of notable action dates (useful for determining status). Contains the following fields:
  - `first` First action (only null if there are no actions).
  - `last` Last action (only null if there are no actions).

  - `passed_lower` Date that the bill seems to have passed the lower chamber (might be null).
  - `passed_upper` Date that the bill seems to have passed the upper chamber (might be null).
  - `signed` Date that the bill appears to have signed into law (might be null).
- `actions` List of objects representing every recorded action for the bill. Action objects have the following fields:
  - `date` Date of action.
  - `action` Name of action as state provides it.
  - `actor` The chamber, person, committee, etc. responsible for this action.
  - `type` Open States-provided action categories, see *Action Types*.
- `chamber` The chamber of origination ('upper' or 'lower')
- `created_at` The date that this object first appeared in our system. (Note: not the date of introduction, see `action_dates` for that information.)
- `updated_at` The date that this object was last updated in our system. (Note: not the last action date, see `action_dates` for that information.)
- `documents` List of associated documents, see `versions` for field details.
- `id` Open States-assigned permanent ID for this bill.
- `scraped_subjects` List of subject areas that the state categorized this bill under.
- `subjects` List of Open States standardized bill subjects, see *Subjects*.
- `sources` List of source URLs used to compile information on this object.
- `sponsors` List of bill sponsors.
  - `name` Name of sponsor as it appears on state website.
  - `leg_id` Open States assigned legislator ID (will be null if no match was found).
  - `type` Type of sponsor ('primary' or 'cosponsor')
- `type` List of *Bill Types*.
- `versions` Versions of the bill text. Both documents and `versions` have the following fields:
  - `url` Official URL for this document.
  - `name` An official name for this document.
  - `mimetype` The mimetype for the document (e.g. 'text/html')
  - `doc_id` An Open States-assigned id uniquely identifying this document.
- **`votes` List of vote objects. Votes may be in the past or future;** a future vote does not have vote-outcome fields like `passed`. A vote object consists of the following fields:
  - `motion` Name of motion being voted upon (e.g. 'Passage'). The nature of these varies widely by state. Some states have a concise vocabulary, some a sloppy vocabulary. Other states include a vote ID in the motion, rendering every motion unique.
  - `chamber` Chamber vote took place in ('upper', 'lower', 'joint')
  - `date` Date of vote.
  - `passed` Boolean; true if *vote* (not bill) succeeded.
  - `id` Open States-assigned unique identifier for vote.

  - `state` State abbreviation.
  - `session` Session key (see *State Metadata* for details).
  - `sources` List of source URLs used to compile information on this object. (Can be empty if vote shares sources with bill.)
  - `yes_count` Total number of yes votes.
  - `no_count` Total number of no votes.
  - `other_count` Total number of 'other' votes (abstain, not present, etc.).
  - `yes_votes`, `no_votes`, `other_votes` List of roll calls of each type. Each is an object consisting of two keys:
    * `name` Name of voter as it appears on state website.
    * `leg_id` Open States assigned legislator ID (will be null if no match was found).

Methods

Bill Search

This method returns just a subset (`state`, `chamber`, `session`, `subjects`, `type`, `id`, `bill_id`, `title`, `created_at`, `updated_at`) of the bill fields by default.

Filter Parameters

The following parameters filter the returned set of bills, at least one must be provided.

- `state` Only return bills from a given state (e.g. 'nc')
- `chamber` Only return bills matching the provided chamber ('upper' or 'lower')
- `bill_id` Only return bills with a given bill_id.
- `q` Only return bills matching the provided full text query.
- `search_window` By default all bills are searched, but if a time window is desired the following options can be passed to search_window:
  - `search_window=all` Default, include all sessions.
  - `search_window=term` Only bills from sessions within the current term.
  - `search_window=session` Only bills from the current session.
  - `search_window=session:2009` Only bills from the session named 2009.
- `updated_since` Only bills updated since a provided date (provided in YYYY-MM-DD format)

Additional Parameters

`sort` Sort-order of results, defaults to 'last', options are:

- first
- last
- updated_at

- created_at

See the above `action_dates`, `created_at`, and `updated_at` documentation for the meaning of these dates.

`page` and `per_page` The API will not return exceedingly large responses, so it may be necessary to use `page` and `per_page` to control the number of results returned:

- `page` Page of results, each of size `per_page` (defaults to 1)
- `per_page` Number of results per page, is unlimited unless page is set, in which case it defaults to 50.

Example: *openstates.org/api/v1/bills/?state=dc&q=taxi*

Bill Detail

This method returns the full detail object for a bill.

Example: *openstates.org/api/v1/bills/ca/20092010/AB%20667/*

Note: This method has an alternate URL form:

- `bills/openstates_bill_id` - e.g. `openstates.org/api/v1/bills/CAB00004148/` - allows lookup by bill_id

Examples

Bill Search

`openstates.org/api/v1/bills/?state=dc&q=taxi`


```
[
 {
  "title": "\"DOC INMATE PROCESSING AND RELEASE AMENDMENT ACT OF 2012\". ",
  "created_at": "2011-07-18 04:35:16",
  "updated_at": "2012-09-14 03:49:38",
  "chamber": "upper",
  "state": "dc",
  "session": "19",
  "subjects": [],
  "type": [ "bill" ],
  "id": "DCB00001021",
  "bill_id": "B 19-0428"
 },
 {
  "title": "\"TAXICAB SERVICE IMPROVEMENT AMENDMENT ACT OF 2012\".\r\n\r\n ",
  "created_at": "2012-01-06 20:53:35",
  "updated_at": "2012-12-07 20:31:54",
  "chamber": "upper",
  "state": "dc",
  "session": "19",
  "subjects": [],
  "type": [ "bill" ],
  "id": "DCB00001501",
  "bill_id": "B 19-0630"
 },
 {
  "title": "\"FISCAL YEAR 2013 BUDGET SUPPORT ACT OF 2012\". ",
  "created_at": "2012-03-27 02:19:29",
```


(continues on next page)

(continued from previous page)


```
  "updated_at": "2012-10-18 03:33:02",
  "chamber": "upper",
  "state": "dc",
  "session": "19",
  "subjects": [],
  "type": [ "bill" ],
  "id": "DCB00001892",
  "bill_id": "B 19-0743"
 },
 {
  "title": "\"FISCAL YEAR 2013 BUDGET SUPPORT EMERGENCY ACT OF 2012\". ",
  "created_at": "2012-06-08 02:51:47",
  "updated_at": "2012-09-07 03:51:01",
  "chamber": "upper",
  "state": "dc",
  "session": "19",
  "subjects": [],
  "type": [ "bill" ],
  "id": "DCB00002085",
  "bill_id": "B 19-0796"
 },
 {
  "title": "\"LEON SWAIN, JR. RECOGNITION RESOLUTION OF 2012\". ",
  "created_at": "2012-04-27 02:36:38",
  "updated_at": "2012-08-22 04:20:34",
  "chamber": "upper",
  "state": "dc",
  "session": "19",
  "subjects": [],
  "type": [ "resolution" ],
  "id": "DCB00001959",
  "bill_id": "CER 19-0218"
 },
 {
  "title": "\"WASHINGTON CONVENTION CENTER ADVISORY COMMITTEE RECOGNITION RESOLUTION␣
↪OF 2011\".",
  "created_at": "2012-03-20 02:17:18",
  "updated_at": "2012-08-22 04:20:34",
  "chamber": "upper",
  "state": "dc",
  "session": "19",
  "subjects": [],
  "type": [ "resolution" ],
  "id": "DCB00001795",
  "bill_id": "CER 19-0171"
 },
 {
  "title": "\"WHEELCHAIR ACCESSIBLE TAXICABS PARITY AMENDMENT ACT OF 2011\".",
  "created_at": "2012-01-06 20:53:35",
  "updated_at": "2012-08-22 04:20:26",
  "chamber": "upper",
  "state": "dc",
  "session": "19",
  "subjects": [],
  "type": [ "bill" ],
  "id": "DCB00001506",
  "bill_id": "B 19-0635"
```


(continues on next page)

(continued from previous page)


```
 },
 {
  "title": "\"FISCAL YEAR 2012 BUDGET SUPPORT ACT OF 2011\".",
  "created_at": "2011-04-06 01:53:14",
  "updated_at": "2012-10-18 03:32:58",
  "chamber": "upper",
  "state": "dc",
  "session": "19",
  "subjects": [],
  "type": [ "bill" ],
  "id": "DCB00000427",
  "bill_id": "B 19-0203"
 },
 {
  "title": "\"FISCAL YEAR 2012 BUDGET SUPPORT EMERGENCY ACT OF 2011\".\r\n ",
  "created_at": "2011-06-16 04:18:55",
  "updated_at": "2012-08-22 04:20:21",
  "chamber": "upper",
  "state": "dc",
  "session": "19",
  "subjects": [],
  "type": [ "bill" ],
  "id": "DCB00000794",
  "bill_id": "B 19-0338"
 },
 {
  "title": "\"PROFESSIONAL TAXICAB STANDARDS AND MEDALLION ESTABLISHMENT ACT OF 2011\
↪".",
  "created_at": "2011-03-21 18:55:32",
  "updated_at": "2012-08-22 04:20:17",
  "chamber": "upper",
  "state": "dc",
  "session": "19",
  "subjects": [],
  "type": [ "bill" ],
  "id": "DCB00000339",
  "bill_id": "B 19-0172"
 }
]
```


Bill Detail

`openstates.org/api/v1/bills/ca/20092010/AB%20667/`


```
{
 "action_dates": {
  "passed_upper": null,
  "passed_lower": null,
  "last": "2009-08-06 00:00:00",
  "signed": null,
  "first": "2009-02-25 00:00:00"
 },
 "actions": [
  { "date": "2009-02-25 00:00:00",
   "action": "Read first time. To print.",
```


(continues on next page)

(continued from previous page)


```
   "type": [ "bill:introduced", "bill:reading:1" ],
   "actor": "lower (Desk)" },
  { "date": "2009-02-26 00:00:00",
   "action": "From printer. May be heard in committee March 28.",
   "type": [ "other" ],
   "actor": "lower (Desk)" },
  { "date": "2009-03-23 00:00:00",
   "action": "Referred to Com. on HEALTH.",
   "type": [ "committee:referred" ],
   "actor": "lower (Committee CX08)" },
  { "date": "2009-04-02 00:00:00",
   "action": "From committee chair, with author's amendments: Amend, and re-refer to␣
↪Com. on HEALTH. Read second time and amended.",
   "type": [ "bill:reading:2" ],
   "actor": "lower (E&E Engrossing)" },
  { "date": "2009-04-13 00:00:00",
   "action": "Re-referred to Com. on HEALTH.",
   "type": [ "committee:referred" ],
   "actor": "lower (Committee CX08)" },
  { "date": "2009-04-15 00:00:00",
   "action": "From committee: Do pass, and re-refer to Com. on B. & P. with␣
↪recommendation: To Consent Calendar. Re-referred. (Ayes 19. Noes 0.) (April 14).",
   "type": [ "other" ],
   "actor": "lower (Committee)" },
  { "date": "2009-04-29 00:00:00",
   "action": "From committee: Do pass, and re-refer to Com. on APPR. with␣
↪recommendation: To Consent Calendar. Re-referred. (Ayes 10. Noes 0.) (April 28).",
   "type": [ "other" ],
   "actor": "lower (Committee)" },
  { "date": "2009-05-04 00:00:00",
   "action": "From committee chair, with author's amendments: Amend, and re-refer to␣
↪Com. on APPR. Read second time and amended.",
   "type": [ "bill:reading:2" ],
   "actor": "lower (E&E Engrossing)" },
  { "date": "2009-05-05 00:00:00",
   "action": "Re-referred to Com. on APPR.",
   "type": [ "committee:referred" ],
   "actor": "lower (Committee CX25)" },
  { "date": "2009-05-14 00:00:00",
   "action": "From committee: Do pass. To Consent Calendar. (May 13).",
   "type": [ "other" ],
   "actor": "lower" },
  { "date": "2009-05-18 00:00:00",
   "action": "Read second time. To Consent Calendar.",
   "type": [ "bill:reading:2" ],
   "actor": "lower" },
  { "date": "2009-05-21 00:00:00",
   "action": "Read third time, passed, and to Senate. (Ayes 77. Noes 0. Page 1628.)",
   "type": [ "other" ],
   "actor": "lower (E&E Engrossing)" },
  { "date": "2009-05-21 00:00:00",
   "action": "In Senate. Read first time. To Com. on RLS. for assignment.",
   "type": [ "bill:reading:1", "committee:referred" ],
   "actor": "upper (Rules)" },
  { "date": "2009-06-04 00:00:00",
   "action": "Referred to Com. on B., P. & E.D.",
   "type": [ "committee:referred" ],
```


(continues on next page)

(continued from previous page)


```
   "actor": "upper (Committee CS42)" },
  { "date": "2009-06-22 00:00:00",
   "action": "From committee: Do pass, and re-refer to Com. on APPR. Re-referred.␣
↪(Ayes 10. Noes 0.) (June 22).",
   "type": [ "other" ],
   "actor": "upper (Committee)" },
  { "date": "2009-06-29 00:00:00",
   "action": "From committee: Be placed on second reading file pursuant to Senate␣
↪Rule 28.8.",
   "type": [ "other" ],
   "actor": "upper" },
  { "date": "2009-06-30 00:00:00",
   "action": "Read second time. To third reading.",
   "type": [ "bill:reading:2" ],
   "actor": "upper" },
  { "date": "2009-07-02 00:00:00",
   "action": "Ordered to Special Consent Calendar.",
   "type": [ "other" ],
   "actor": "upper" },
  { "date": "2009-07-09 00:00:00",
   "action": "Read third time, passed, and to Assembly. (Ayes 34. Noes 0. Page 1667.)
↪",
   "type": [ "other" ],
   "actor": "upper (Desk)" },
  { "date": "2009-07-09 00:00:00",
   "action": "In Assembly. To enrollment.",
   "type": [ "other" ],
   "actor": "lower (E&E Enrollment)" },
  { "date": "2009-07-30 00:00:00",
   "action": "Enrolled and to the Governor at 2:30 p.m.",
   "type": [ "other" ],
   "actor": "executive" },
  { "date": "2009-08-05 00:00:00",
   "action": "Approved by the Governor.",
   "type": [ "other" ],
   "actor": "executive" },
  { "date": "2009-08-06 00:00:00",
   "action": "Chaptered by Secretary of State - Chapter 119, Statutes of 2009.",
   "type": [ "other" ],
   "actor": "Secretary of State" }
 ],
 "alternate_titles": [
  "An act to amend Section 104830 of, and to add Section 104762 to, the Health and␣
↪Safety Code, relating to oral health."
 ],
 "bill_id": "AB 667",
 "chamber": "lower",
 "created_at": "2010-07-09 17:28:10",
 "documents": [],
 "id": "CAB00004148",
 "level": "state",
 "scraped_subjects": [ "Topical fluoride application." ],
 "session": "20092010",
 "sources": [
  { "url": "http://leginfo.legislature.ca.gov/faces/billNavClient.xhtml?bill_
↪id=200920100AB667" }
 ],
```


(continues on next page)

(continued from previous page)


```
 "sponsors": [
  { "leg_id": "CAL000044", "type": "primary", "name": "Block" }
 ],
 "state": "ca",
 "subjects": [],
 "title": "An act to amend Section 1750.1 of the Business and Professions Code, and␣
↪to amend Section 104830 of, and to add Section 104762 to, the Health and Safety␣
↪Code, relating to oral health.",
 "type": [ "bill", "fiscal committee" ],
 "updated_at": "2012-04-06 17:17:37",
 "versions": [
  {
   "url": "http://leginfo.legislature.ca.gov/faces/billNavClient.xhtml?bill_
↪id=200920100AB667",
   "mimetype": "text/html", "doc_id": "CAD00040031", "name": "AB667"
  }
 ],
 "votes": [
  {
   "other_count": 6, "+threshold": "1/2",
   "other_votes": [
    { "leg_id": "CAL000014", "name": "Ashburn" },
    { "leg_id": "CAL000036", "name": "Calderon" },
    { "leg_id": "CAL000010", "name": "Corbett" },
    { "leg_id": "CAL000026", "name": "Harman" },
    { "leg_id": "CAL000021", "name": "Oropeza" },
    { "leg_id": "CAL000005", "name": "Wolk" }
   ],
   "yes_count": 34,
   "yes_votes": [
    { "leg_id": "CAL000004", "name": "Aanestad" },
    { "leg_id": "CAL000039", "name": "Alquist" },
    { "leg_id": "CAL000029", "name": "Benoit" },
    { "leg_id": "CAL000017", "name": "Cedillo" },
    { "leg_id": "CAL000011", "name": "Cogdill" },
    { "leg_id": "CAL000037", "name": "Correa" },
    { "leg_id": "CAL000001", "name": "Cox" },
    { "leg_id": "CAL000007", "name": "DeSaulnier" },
    { "leg_id": "CAL000032", "name": "Denham" },
    { "leg_id": "CAL000038", "name": "Ducheny" },
    { "leg_id": "CAL000023", "name": "Dutton" },
    { "leg_id": "CAL000033", "name": "Florez" },
    { "leg_id": "CAL000009", "name": "Hancock" },
    { "leg_id": "CAL000027", "name": "Hollingsworth" },
    { "leg_id": "CAL000022", "name": "Huff" },
    { "leg_id": "CAL000030", "name": "Kehoe" },
    { "leg_id": "CAL000003", "name": "Leno" },
    { "leg_id": "CAL000016", "name": "Liu" },
    { "leg_id": "CAL000080", "name": "Lowenthal" },
    { "leg_id": "CAL000012", "name": "Maldonado" },
    { "leg_id": null, "name": "Negrete McLeod" },
    { "leg_id": "CAL000034", "name": "Padilla" },
    { "leg_id": "CAL000018", "name": "Pavley" },
    { "leg_id": "CAL000040", "name": "Price" },
    { "leg_id": "CAL000019", "name": "Romero" },
    { "leg_id": "CAL000013", "name": "Runner" },
    { "leg_id": "CAL000031", "name": "Simitian" },
```


(continues on next page)

(continued from previous page)


```
    { "leg_id": "CAL000006", "name": "Steinberg" },
    { "leg_id": "CAL000015", "name": "Strickland" },
    { "leg_id": "CAL000025", "name": "Walters" },
    { "leg_id": "CAL000002", "name": "Wiggins" },
    { "leg_id": "CAL000035", "name": "Wright" },
    { "leg_id": "CAL000028", "name": "Wyland" },
    { "leg_id": "CAL000008", "name": "Yee" }
   ],
   "no_count": 0,
   "motion": "Special Consent #12 AB667 Block By Alquist",
   "chamber": "upper",
   "state": "ca",
   "session": "20092010",
   "sources": [],
   "passed": true,
   "date": "2009-07-09 16:50:00",
   "vote_id": "CAV00009230",
   "type": "other",
   "id": "CAV00009230",
   "bill_id": "CAB00004148",
   "no_votes": []
  }
 ]
}
```


2.4.3 Legislators

Note: API v2 is now available, please check out API v2.

Legislator Search Search legislators by their attributes.

Legislator Detail Get full detail for a legislator, including all roles.

Geo Lookup Lookup all legislators that serve districts containing a given point.

Legislator Fields

The following fields are available on legislator objects:

- `leg_id` Legislator's permanent Open States ID. (e.g. 'ILL000555', 'NCL000123')
- `state` Legislator's state.
- `active` Boolean value indicating whether or not the legislator is currently in office.
- `chamber` Chamber the legislator is currently serving in if active ('upper' or 'lower')
- `district` District the legislator is currently serving in if active (e.g. '7', '6A')
- `party` Party the legislator is currently representing if active.
- `email` Legislator's primary email address.
- `full_name` Full display name for legislator.
- `first_name` First name of legislator.

- middle_name Middle name of legislator.
- last_name Last name of legislator.
- suffixes Name suffixes (e.g. 'Jr.', 'III') of legislator.
- photo_url URL of an official photo of this legislator.
- url URL of an official webpage for this legislator.
- created_at The date that this object first appeared in our system.
- updated_at The date that this object was last updated in our system.
- created_at Date at which this legislator was added to our system.
- updated_at Date at which this legislator was last updated.
- offices List of office objects representing contact details for the legislator. Comprised of the following fields:
  - type 'capitol' or 'district'
  - name Name of the address (e.g. 'Council Office', 'District Office')
  - address Street address.
  - phone Phone number.
  - fax Fax number.
  - email Email address. *Any of these fields may be ''null'' if not found.*
- roles List of currently active *role objects* if legislator is in office.
- old_roles Dictionary mapping term keys to lists of roles that were valid for that term.

Roles

roles and old_roles are comprised of role objects.

Role objects can have the following fields:

- term Term key for this role. (See metadata *notes on terms and sessions* for details.)
- chamber
- state
- start_date (optional)
- end_date (optional)
- type 'member' or 'committee member'

If the role type is 'member':

- party
- district

And if the type is 'committee member':

- committee name of parent committee
- subcommittee name of subcommittee (if null, membership is just for a committee)
- committee_id Open States id for committee that legislator is a member of
- position position on committee

- `old_roles`
- `sources` List of URLs used in gathering information for this legislator.

Methods

Legislator Search

This method allows looking up a legislator by a number of parameters, the results do not include the `roles` or `old_roles` items by default.

Parameters

- `state` Filter by state.
- `chamber` Only legislators with a role in the specified chamber.
- `district` Only legislators that have represented the specified district.

Example: *openstates.org/api/v1/legislators/?state=dc&chamber=upper*

Legislator Detail

This method returns the full detail for a legislator.

Example: *openstates.org/api/v1/legislators/DCL000012/*

Geo Lookup

Lookup all legislators serving districts containing a given location.

Example: *openstates.org/api/v1/legislators/geo/?lat=35.79&long=-78.78*

Examples

Legislator Search

`openstates.org/api/v1/legislators/?state=dc&chamber=upper`


```
[
 {
  "first_name": "Anita",
  "last_name": "Bonds",
  "middle_name": "",
  "district": "At-Large",
  "chamber": "upper",
  "url": "http://dccouncil.us/council/anita-bonds",
  "created_at": "2013-01-07 21:05:06",
  "updated_at": "2013-03-26 03:22:24",
  "email": "abonds@dccouncil.us",
  "active": true,
  "state": "dc",
```


(continues on next page)

(continued from previous page)


```
  "offices": [
   {
    "fax": "(202) 724-8099",
    "name": "Council Office",
    "phone": "(202) 724-8064",
    "address": "1350 Pennsylvania Avenue NW, Suite 408, Washington, DC 20004",
    "type": "capitol",
    "email": null
   }
  ],
  "full_name": "Anita Bonds",
  "leg_id": "DCL000021",
  "party": "Democratic",
  "suffixes": "",
  "id": "DCL000021",
  "photo_url": "http://dccouncil.us/files/user_uploads/member_photos/AAA_small.jpg"
 },
 {
  "+fax": "(202) 724-8099",
  "last_name": "Mendelson",
  "updated_at": "2013-03-26 03:20:14",
  "full_name": "Phil Mendelson",
  "id": "DCL000005",
  "first_name": "Phil",
  "middle_name": "",
  "district": "Chairman",
  "office_address": "1350 Pennsylvania Avenue NW, Suite 402, Washington, DC 20004",
  "state": "dc",
  "votesmart_id": "72089",
  "party": "Democratic",
  "email": "pmendelson@dccouncil.us",
  "leg_id": "DCL000005",
  "active": true,
  "photo_url": "http://dccouncil.us/files/user_uploads/member_photos/mendelson.jpg",
  "level": "state",
  "url": "http://dccouncil.us/council/phil-mendelson",
  "created_at": "2011-02-17 22:43:55",
  "chamber": "upper",
  "offices": [
   {
    "fax": "(202) 724-8099",
    "name": "Council Office",
    "phone": "(202) 724-8032     ",
    "address": "1350 Pennsylvania Avenue NW, Suite 504, Washington, DC 20004",
    "type": "capitol",
    "email": null
   }
  ],
  "suffixes": "",
  "+phone": "(202) 724-8064      "
 },
 {
  "first_name": "David",
  "last_name": "Grosso",
  "middle_name": "",
  "district": "At-Large",
  "chamber": "upper",
```


(continues on next page)

(continued from previous page)


```
  "url": "http://dccouncil.us/council/david-grosso",
  "created_at": "2013-01-07 21:05:06",
  "updated_at": "2013-03-26 03:22:24",
  "email": "dgrosso@dccouncil.us",
  "active": true,
  "state": "dc",
  "offices": [
   {
    "fax": "(202) 724-8071",
    "name": "Council Office",
    "phone": "(202) 724-8105",
    "address": "1350 Pennsylvania Avenue NW, Suite 406, Washington, DC 20004",
    "type": "capitol",
    "email": null
   }
  ],
  "full_name": "David Grosso",
  "leg_id": "DCL000020",
  "party": "Independent",
  "suffixes": "",
  "id": "DCL000020",
  "photo_url": "http://dccouncil.us/files/user_uploads/member_photos/david_grosso_
↪color__small.jpg"
 },
 {
  "+fax": "(202) 741-0911",
  "last_name": "Alexander",
  "updated_at": "2013-03-26 03:22:24",
  "full_name": "Yvette Alexander",
  "id": "DCL000010",
  "first_name": "Yvette",
  "middle_name": "",
  "district": "Ward 7",
  "office_address": "1350 Pennsylvania Avenue, Suite 400, NW Washington, DC 20004",
  "state": "dc",
  "votesmart_id": "72072",
  "party": "Democratic",
  "email": "yalexander@dccouncil.us",
  "leg_id": "DCL000010",
  "active": true,
  "photo_url": "http://dccouncil.us/files/user_uploads/member_photos/alexander_
↪dec2011.jpg",
  "level": "state",
  "url": "http://dccouncil.us/council/yvette-alexander",
  "created_at": "2011-02-17 22:43:55",
  "chamber": "upper",
  "offices": [
   {
    "fax": "(202) 741-0911",
    "name": "Council Office",
    "phone": "(202) 724-8068",
    "address": "1350 Pennsylvania Avenue, Suite 400, NW Washington, DC 20004",
    "type": "capitol",
    "email": null
   }
  ],
  "+phone": "(202) 724-8068",
```


(continues on next page)

(continued from previous page)


```
  "suffixes": ""
 },
 {
  "+fax": "(202) 724-8054",
  "last_name": "Wells",
  "updated_at": "2013-03-26 03:22:24",
  "full_name": "Tommy Wells",
  "id": "DCL000008",
  "first_name": "Tommy",
  "middle_name": "",
  "district": "Ward 6",
  "office_address": "1350 Pennsylvania Avenue, Suite 408, NW Washington, DC 20004",
  "state": "dc",
  "votesmart_id": "72071",
  "party": "Democratic",
  "email": "twells@dccouncil.us",
  "leg_id": "DCL000008",
  "active": true,
  "photo_url": "http://dccouncil.us/files/user_uploads/member_photos/wells2.jpg",
  "level": "state",
  "url": "http://dccouncil.us/council/tommy-wells",
  "created_at": "2011-02-17 22:43:55",
  "chamber": "upper",
  "offices": [
   {
    "fax": "(202) 724-8054",
    "name": "Council Office",
    "phone": "(202) 724-8072",
    "address": "1350 Pennsylvania Avenue, Suite 402, NW Washington, DC 20004",
    "type": "capitol",
    "email": null
   }
  ],
  "+phone": "(202) 724-8072",
  "suffixes": ""
 },
 {
  "+fax": "(202) 727-8210",
  "last_name": "Orange",
  "updated_at": "2013-03-26 03:22:24",
  "full_name": "Vincent Orange",
  "id": "DCL000014",
  "first_name": "Vincent",
  "middle_name": "",
  "district": "At-Large",
  "office_address": "1350 Pennsylvania Avenue NW, Suite 107, Washington, DC 20004",
  "state": "dc",
  "party": "Democratic",
  "email": "vorange@dccouncil.us",
  "leg_id": "DCL000014",
  "active": true,
  "photo_url": "http://dccouncil.us/files/user_uploads/member_photos/orange.jpg",
  "level": "state",
  "url": "http://dccouncil.us/council/vincent-orange",
  "created_at": "2011-05-12 02:08:19",
  "chamber": "upper",
  "offices": [
```


(continues on next page)

(continued from previous page)


```
    {
     "fax": "(202) 727-8210",
     "name": "Council Office",
     "phone": "(202) 724-8174      ",
     "address": "1350 Pennsylvania Avenue NW, Suite 107, Washington, DC 20004",
     "type": "capitol",
     "email": null
    }
   ],
   "+phone": "(202) 724-8174      ",
   "suffixes": ""
  },
  {
   "+fax": "(202) 741-0908",
   "last_name": "Bowser",
   "updated_at": "2013-03-26 03:22:24",
   "full_name": "Muriel Bowser",
   "id": "DCL000011",
   "first_name": "Muriel",
   "middle_name": "",
   "district": "Ward 4",
   "office_address": "1350 Pennsylvania Avenue, Suite 110, NW Washington, DC 20004",
   "state": "dc",
   "votesmart_id": "72064",
   "party": "Democratic",
   "email": "mbowser@dccouncil.us",
   "leg_id": "DCL000011",
   "active": true,
   "photo_url": "http://dccouncil.us/files/user_uploads/member_photos/Bowser_Official_
↪Photo_2012_small.jpg",
   "level": "state",
   "url": "http://dccouncil.us/council/muriel-bowser",
   "created_at": "2011-02-17 22:43:55",
   "chamber": "upper",
   "offices": [
    {
     "fax": "(202) 741-0908",
     "name": "Council Office",
     "phone": "(202) 724-8052",
     "address": "1350 Pennsylvania Avenue, Suite 110, NW Washington, DC 20004",
     "type": "capitol",
     "email": null
    }
   ],
   "suffixes": "",
   "+phone": "(202) 724-8052"
  },
  {
   "+fax": "(202) 724-8087",
   "last_name": "Catania",
   "updated_at": "2013-03-26 03:22:24",
   "full_name": "David Catania",
   "id": "DCL000003",
   "first_name": "David",
   "middle_name": "",
   "district": "At-Large",
   "office_address": "1350 Pennsylvania Avenue NW, Suite 404, Washington, DC 20004",
```


(continues on next page)

(continued from previous page)


```
  "state": "dc",
  "votesmart_id": "72081",
  "party": "Independent",
  "email": "dcatania@dccouncil.us",
  "leg_id": "DCL000003",
  "active": true,
  "photo_url": "http://dccouncil.us/files/user_uploads/member_photos/catania.jpg",
  "level": "state",
  "url": "http://dccouncil.us/council/david-catania",
  "created_at": "2011-02-17 22:43:55",
  "chamber": "upper",
  "offices": [
   {
    "fax": "(202) 724-8087",
    "name": "Council Office",
    "phone": "(202) 724-7772      ",
    "address": "1350 Pennsylvania Avenue NW, Suite 404, Washington, DC 20004",
    "type": "capitol",
    "email": null
   }
  ],
  "+phone": "(202) 724-7772      ",
  "suffixes": ""
 },
 {
  "+fax": "(202) 724-8076",
  "last_name": "McDuffie",
  "updated_at": "2013-03-26 03:22:24",
  "full_name": "Kenyan McDuffie",
  "id": "DCL000017",
  "first_name": "Kenyan",
  "middle_name": "",
  "district": "Ward 5",
  "office_address": "1350 Pennsylvania Avenue NW, Suite 410, Washington, DC 20004",
  "state": "dc",
  "party": "Democratic",
  "email": "kmcduffie@dccouncil.us",
  "leg_id": "DCL000017",
  "active": true,
  "photo_url": "http://dccouncil.us/files/user_uploads/member_photos/Councilmember_
↪Kenyan_R._McDuffie_Official_Photograph_small.jpg",
  "level": "state",
  "url": "http://dccouncil.us/council/kenyan-mcduffie",
  "created_at": "2012-05-31 02:28:23",
  "chamber": "upper",
  "offices": [
   {
    "fax": "(202) 724-8076",
    "name": "Council Office",
    "phone": "(202) 724-8028 ",
    "address": "1350 Pennsylvania Avenue NW, Suite 506, Washington, DC 20004",
    "type": "capitol",
    "email": null
   }
  ],
  "suffixes": "",
  "+phone": "(202) 724-8028 "
```


(continues on next page)

(continued from previous page)


```
},
{
 "+fax": "(202) 724-8023",
 "last_name": "Evans",
 "updated_at": "2013-03-26 03:22:24",
 "full_name": "Jack Evans",
 "id": "DCL000009",
 "first_name": "Jack",
 "middle_name": "",
 "district": "Ward 2",
 "office_address": "1350 Pennsylvania Avenue, Suite 106, NW Washington, DC 20004",
 "state": "dc",
 "votesmart_id": "72044",
 "party": "Democratic",
 "email": "jevans@dccouncil.us",
 "leg_id": "DCL000009",
 "active": true,
 "photo_url": "http://dccouncil.us/files/user_uploads/member_photos/evans.jpg",
 "level": "state",
 "url": "http://dccouncil.us/council/jack-evans",
 "created_at": "2011-02-17 22:43:55",
 "chamber": "upper",
 "offices": [
  {
   "fax": "(202) 724-8023",
   "name": "Council Office",
   "phone": "(202) 724-8058",
   "address": "1350 Pennsylvania Avenue, Suite 106, NW Washington, DC 20004",
   "type": "capitol",
   "email": null
  }
 ],
 "+phone": "(202) 724-8058",
 "suffixes": ""
},
{
 "+fax": "(202) 724-8109",
 "last_name": "Graham",
 "updated_at": "2013-03-26 03:22:24",
 "full_name": "Jim Graham",
 "id": "DCL000007",
 "first_name": "Jim",
 "middle_name": "",
 "district": "Ward 1",
 "office_address": "1350 Pennsylvania Avenue, Suite 105, NW Washington, DC 20004",
 "state": "dc",
 "votesmart_id": "72038",
 "party": "Democratic",
 "email": "jgraham@dccouncil.us",
 "leg_id": "DCL000007",
 "active": true,
 "photo_url": "http://dccouncil.us/files/user_uploads/member_photos/graham.jpg",
 "level": "state",
 "url": "http://dccouncil.us/council/jim-graham",
 "created_at": "2011-02-17 22:43:55",
 "chamber": "upper",
 "offices": [
```


(continues on next page)

(continued from previous page)


```
   {
    "fax": "(202) 724-8109",
    "name": "Council Office",
    "phone": "(202) 724-8181",
    "address": "1350 Pennsylvania Avenue, Suite 105, NW Washington, DC 20004",
    "type": "capitol",
    "email": null
   }
  ],
  "+phone": "(202) 724-8181",
  "suffixes": ""
 },
 {
  "+fax": "(202) 724-8118",
  "last_name": "Cheh",
  "updated_at": "2013-03-26 03:22:24",
  "full_name": "Mary M Cheh",
  "id": "DCL000002",
  "first_name": "Mary",
  "middle_name": "M",
  "district": "Ward 3",
  "office_address": "1350 Pennsylvania Avenue, Suite 108, NW  Washington, DC 20004",
  "state": "dc",
  "votesmart_id": "72047",
  "party": "Democratic",
  "email": "mcheh@dccouncil.us",
  "leg_id": "DCL000002",
  "active": true,
  "photo_url": "http://dccouncil.us/files/user_uploads/member_photos/cheh.jpg",
  "level": "state",
  "url": "http://dccouncil.us/council/mary-m.-cheh",
  "created_at": "2011-02-17 22:43:55",
  "chamber": "upper",
  "offices": [
   {
    "fax": "(202) 724-8118",
    "name": "Council Office",
    "phone": "(202) 724-8062",
    "address": "1350 Pennsylvania Avenue, Suite 108, NW  Washington, DC 20004",
    "type": "capitol",
    "email": null
   }
  ],
  "+phone": "(202) 724-8062",
  "suffixes": ""
 },
 {
  "+fax": "(202) 724-8055",
  "last_name": "Barry",
  "updated_at": "2013-03-26 03:22:24",
  "full_name": "Marion Barry",
  "id": "DCL000012",
  "first_name": "Marion",
  "middle_name": "",
  "district": "Ward 8",
  "office_address": "1350 Pennsylvania Avenue NW, Suite 102, Washington, DC 20004",
  "state": "dc",
```


(continues on next page)

(continued from previous page)


```
  "votesmart_id": "72074",
  "party": "Democratic",
  "email": "mbarry@dccouncil.us",
  "leg_id": "DCL000012",
  "active": true,
  "photo_url": "http://dccouncil.us/files/user_uploads/member_photos/barry.jpg",
  "level": "state",
  "url": "http://dccouncil.us/council/marion-barry",
  "created_at": "2011-02-17 22:43:55",
  "chamber": "upper",
  "offices": [
   {
    "fax": "(202) 724-8055",
    "name": "Council Office",
    "phone": "(202) 724-8045",
    "address": "1350 Pennsylvania Avenue NW, Suite 102, Washington, DC 20004",
    "type": "capitol",
    "email": null
   }
  ],
  "+phone": "(202) 724-8045",
  "suffixes": ""
 }
]
```


Legislator Detail

`openstates.org/api/v1/legislators/DCL000012/`


```
{
 "active": true,
 "chamber": "upper",
 "created_at": "2011-02-17 22:43:55",
 "district": "Ward 8",
 "email": "mbarry@dccouncil.us",
 "first_name": "Marion",
 "full_name": "Marion Barry",
 "id": "DCL000012",
 "last_name": "Barry",
 "leg_id": "DCL000012",
 "level": "state",
 "middle_name": "",
 "office_address": "1350 Pennsylvania Avenue NW, Suite 102, Washington, DC 20004",
 "offices": [
  {
   "fax": "(202) 724-8055",
   "name": "Council Office",
   "phone": "(202) 724-8045",
   "address": "1350 Pennsylvania Avenue NW, Suite 102, Washington, DC 20004",
   "type": "capitol",
   "email": null
  }
 ],
 "old_roles": {
  "2011-2012": [
```


(continues on next page)

(continued from previous page)


```
  {
   "term": "2011-2012",
   "end_date": null,
   "district": "Ward 8",
   "chamber": "upper",
   "state": "dc",
   "party": "Democratic",
   "type": "member",
   "start_date": null
  },
  {
   "term": "2011-2012",
   "committee_id": "DCC000017",
   "chamber": "upper",
   "state": "dc",
   "subcommittee": null,
   "committee": "Finance and Revenue",
   "position": "member",
   "type": "committee member"
  },
  {
   "term": "2011-2012",
   "committee_id": "DCC000027",
   "chamber": "upper",
   "state": "dc",
   "subcommittee": null,
   "committee": "Jobs and Workforce Development",
   "position": "member",
   "type": "committee member"
  },
  {
   "term": "2011-2012",
   "committee_id": "DCC000021",
   "chamber": "upper",
   "state": "dc",
   "subcommittee": null,
   "committee": "the Judiciary",
   "position": "member",
   "type": "committee member"
  },
  {
   "term": "2011-2012",
   "committee_id": "DCC000019",
   "chamber": "upper",
   "state": "dc",
   "subcommittee": null,
   "committee": "Aging and Community Affairs",
   "position": "member",
   "type": "committee member"
  },
  {
   "term": "2011-2012",
   "committee_id": "DCC000026",
   "chamber": "upper",
   "state": "dc",
   "subcommittee": null,
   "committee": "Economic Development and Housing",
```


(continues on next page)

(continued from previous page)


```
    "position": "member",
    "type": "committee member"
   },
   {
    "term": "2011-2012",
    "committee_id": "DCC000014",
    "chamber": "upper",
    "state": "dc",
    "subcommittee": null,
    "committee": "Human Services",
    "position": "member",
    "type": "committee member"
   },
   {
    "term": "2011-2012",
    "committee_id": "DCC000023",
    "chamber": "upper",
    "state": "dc",
    "subcommittee": null,
    "committee": "Health",
    "position": "member",
    "type": "committee member"
   }
  ]
 },
 "party": "Democratic",
 "photo_url": "http://dccouncil.us/files/user_uploads/member_photos/barry.jpg",
 "roles": [
  {
   "term": "2013-2014",
   "end_date": null,
   "district": "Ward 8",
   "chamber": "upper",
   "state": "dc",
   "party": "Democratic",
   "type": "member",
   "start_date": null
  },
  {
   "term": "2013-2014",
   "committee_id": "DCC000014",
   "chamber": "upper",
   "state": "dc",
   "subcommittee": null,
   "committee": "Human Services",
   "position": "member",
   "type": "committee member"
  },
  {
   "term": "2013-2014",
   "committee_id": "DCC000017",
   "chamber": "upper",
   "state": "dc",
   "subcommittee": null,
   "committee": "Finance and Revenue",
   "position": "member",
   "type": "committee member"
```


(continues on next page)

(continued from previous page)


```
  },
  {
   "term": "2013-2014",
   "committee_id": "DCC000032",
   "chamber": "upper",
   "state": "dc",
   "subcommittee": null,
   "committee": "Education",
   "position": "member",
   "type": "committee member"
  },
  {
   "term": "2013-2014",
   "committee_id": "DCC000031",
   "chamber": "upper",
   "state": "dc",
   "subcommittee": null,
   "committee": "Workforce and Community Affairs",
   "position": "member",
   "type": "committee member"
  }
 ],
 "sources": [ { "url": "http://dccouncil.us/council/marion-barry" } ],
 "state": "dc",
 "suffixes": "",
 "updated_at": "2013-03-26 03:22:24",
 "url": "http://dccouncil.us/council/marion-barry",
 "votesmart_id": "72074"
}
```


Geo Lookup

`openstates.org/api/v1/legislators/geo/?lat=35.79&long=-78.78`


```
[
 {
  "last_name": "Stein",
  "suffix": "",
  "updated_at": "2013-03-27 02:35:39",
  "sources": [ { "url": "http://www.ncga.state.nc.us/gascripts/members/viewMember.pl?
↪sChamber=Senate&nUserID=267" } ],
  "full_name": "Josh Stein",
  "old_roles": {
   "2009-2010": [
    {
     "term": "2009-2010",
     "end_date": null,
     "district": "16",
     "level": "state",
     "chamber": "upper",
     "state": "nc",
     "party": "Democratic",
     "type": "member",
     "start_date": null
    },
```


(continues on next page)

(continued from previous page)


```
    {
     "term": "2009-2010",
     "committee_id": "NCC000002",
     "level": "state",
     "chamber": "upper",
     "state": "nc",
     "subcommittee": null,
     "committee": "Appropriations on Department of Transportation",
     "type": "committee member"
    },
    {
     "term": "2009-2010",
     "committee_id": "NCC000008",
     "level": "state",
     "chamber": "upper",
     "state": "nc",
     "subcommittee": null,
     "committee": "Appropriations/Base Budget",
     "type": "committee member"
    },
    {
     "term": "2009-2010",
     "committee_id": "NCC000009",
     "level": "state",
     "chamber": "upper",
     "state": "nc",
     "subcommittee": null,
     "committee": "Commerce",
     "type": "committee member"
    },
    {
     "term": "2009-2010",
     "committee_id": "NCC000010",
     "level": "state",
     "chamber": "upper",
     "state": "nc",
     "subcommittee": null,
     "committee": "Education/Higher Education",
     "type": "committee member"
    },
    {
     "term": "2009-2010",
     "committee_id": "NCC000073",
     "level": "state",
     "chamber": "upper",
     "state": "nc",
     "subcommittee": null,
     "committee": "Finance",
     "type": "committee member"
    },
    {
     "term": "2009-2010",
     "committee_id": "NCC000012",
     "level": "state",
     "chamber": "upper",
     "state": "nc",
     "subcommittee": null,
```


(continues on next page)

(continued from previous page)


```
    "committee": "Health Care",
    "type": "committee member"
   },
   {
    "term": "2009-2010",
    "committee_id": "NCC000074",
    "level": "state",
    "chamber": "upper",
    "state": "nc",
    "subcommittee": null,
    "committee": "Judiciary I",
    "type": "committee member"
   },
   {
    "term": "2009-2010",
    "committee_id": "NCC000022",
    "level": "state",
    "chamber": "upper",
    "state": "nc",
    "subcommittee": null,
    "committee": "Select Committee on Economic Recovery",
    "type": "committee member"
   },
   {
    "term": "2009-2010",
    "committee_id": "NCC000024",
    "level": "state",
    "chamber": "upper",
    "state": "nc",
    "subcommittee": null,
    "committee": "Select Committee on Energy, Science and Technology",
    "type": "committee member"
   }
  ],
  "2011-2012": [
   {
    "term": "2011-2012",
    "end_date": null,
    "district": "16",
    "chamber": "upper",
    "state": "nc",
    "party": "Democratic",
    "type": "member",
    "start_date": null
   },
   {
    "term": "2011-2012",
    "committee_id": "NCC000009",
    "chamber": "upper",
    "state": "nc",
    "subcommittee": null,
    "committee": "Commerce",
    "position": "member",
    "type": "committee member"
   },
   {
    "term": "2011-2012",
```


(continues on next page)

(continued from previous page)


```
    "committee_id": "NCC000100",
    "chamber": "upper",
    "state": "nc",
    "subcommittee": null,
    "committee": "Education / Higher Education",
    "position": "member",
    "type": "committee member"
   },
   {
    "term": "2011-2012",
    "committee_id": "NCC000073",
    "chamber": "upper",
    "state": "nc",
    "subcommittee": null,
    "committee": "Finance",
    "position": "member",
    "type": "committee member"
   },
   {
    "term": "2011-2012",
    "committee_id": "NCC000074",
    "chamber": "upper",
    "state": "nc",
    "subcommittee": null,
    "committee": "Judiciary I",
    "position": "member",
    "type": "committee member"
   },
   {
    "term": "2011-2012",
    "committee_id": "NCC000018",
    "chamber": "upper",
    "state": "nc",
    "subcommittee": null,
    "committee": "Rules and Operations of the Senate",
    "position": "member",
    "type": "committee member"
   }
  ]
 },
 "id": "NCL000047",
 "first_name": "Josh",
 "middle_name": "",
 "district": "16",
 "state": "nc",
 "votesmart_id": "102971",
 "party": "Democratic",
 "email": "Josh.Stein@ncleg.net",
 "leg_id": "NCL000047",
 "boundary_id": "sldu/nc-16",
 "active": true,
 "transparencydata_id": "d3917a35b626477a9a7afaf7dbf206be",
 "photo_url": "http://www.ncga.state.nc.us/Senate/pictures/hiRes/267.jpg",
 "roles": [
  {
   "term": "2013-2014",
   "end_date": null,
```


(continues on next page)

(continued from previous page)


```
    "district": "16",
    "chamber": "upper",
    "state": "nc",
    "party": "Democratic",
    "type": "member",
    "start_date": null
  },
  {
    "term": "2013-2014",
    "committee_id": "NCC000009",
    "chamber": "upper",
    "state": "nc",
    "subcommittee": null,
    "committee": "Commerce",
    "position": "member",
    "type": "committee member"
  },
  {
    "term": "2013-2014",
    "committee_id": "NCC000100",
    "chamber": "upper",
    "state": "nc",
    "subcommittee": null,
    "committee": "Education / Higher Education",
    "position": "member",
    "type": "committee member"
  },
  {
    "term": "2013-2014",
    "committee_id": "NCC000073",
    "chamber": "upper",
    "state": "nc",
    "subcommittee": null,
    "committee": "Finance",
    "position": "member",
    "type": "committee member"
  },
  {
    "term": "2013-2014",
    "committee_id": "NCC000012",
    "chamber": "upper",
    "state": "nc",
    "subcommittee": null,
    "committee": "Health Care",
    "position": "member",
    "type": "committee member"
  },
  {
    "term": "2013-2014",
    "committee_id": "NCC000074",
    "chamber": "upper",
    "state": "nc",
    "subcommittee": null,
    "committee": "Judiciary I",
    "position": "member",
    "type": "committee member"
  },
```


(continues on next page)

(continued from previous page)


```
    {
     "term": "2013-2014",
     "committee_id": "NCC000018",
     "chamber": "upper",
     "state": "nc",
     "subcommittee": null,
     "committee": "Rules and Operations of the Senate",
     "position": "member",
     "type": "committee member"
    }
   ],
   "level": "state",
   "url": "http://www.ncga.state.nc.us/gascripts/members/viewMember.pl?sChamber=Senate&
↪nUserID=267",
   "created_at": "2010-08-03 17:14:46",
   "nimsp_id": "9383",
   "chamber": "upper",
   "offices": [
    {
     "fax": null,
     "name": "Capitol Office",
     "phone": "(919) 715-6400",
     "address": "NC Senate\n16 W. Jones Street, Room 1113\n\nRaleigh, NC 27601-2808",
     "type": "capitol",
     "email": null
    }
   ],
   "suffixes": ""
  },
  {
   "last_name": "Hall",
   "updated_at": "2013-03-27 02:35:42",
   "sources": [
    {
     "url": "http://www.ncga.state.nc.us/gascripts/members/viewMember.pl?
↪sChamber=House&nUserID=679"
    }
   ],
   "full_name": "Duane Hall",
   "id": "NCL000282",
   "first_name": "Duane",
   "middle_name": "",
   "district": "11",
   "state": "nc",
   "party": "Democratic",
   "email": "Duane.Hall@ncleg.net",
   "leg_id": "NCL000282",
   "boundary_id": "sldl/nc-11",
   "+notice": null,
   "transparencydata_id": "07eff70ee51441d093b33667a2a6f877",
   "active": true,
   "photo_url": "http://www.ncga.state.nc.us/House/pictures/hiRes/679.jpg",
   "roles": [
    {
     "term": "2013-2014",
     "end_date": null,
     "district": "11",
```


(continues on next page)

(continued from previous page)


```
    "chamber": "lower",
    "state": "nc",
    "party": "Democratic",
    "type": "member",
    "start_date": null
   },
   {
    "term": "2013-2014",
    "committee_id": "NCC000028",
    "chamber": "lower",
    "state": "nc",
    "subcommittee": null,
    "committee": "Appropriations",
    "position": "member",
    "type": "committee member"
   },
   {
    "term": "2013-2014",
    "committee_id": "NCC000035",
    "chamber": "lower",
    "state": "nc",
    "subcommittee": null,
    "committee": "Appropriations Subcommittee on Transportation",
    "position": "member",
    "type": "committee member"
   },
   {
    "term": "2013-2014",
    "committee_id": "NCC000082",
    "chamber": "lower",
    "state": "nc",
    "subcommittee": null,
    "committee": "Commerce and Job Development",
    "position": "member",
    "type": "committee member"
   },
   {
    "term": "2013-2014",
    "committee_id": "NCC000178",
    "chamber": "lower",
    "state": "nc",
    "subcommittee": null,
    "committee": "Commerce and Job Development Subcommittee on Alcoholic Beverage␣
↪Control",
    "position": "member",
    "type": "committee member"
   },
   {
    "term": "2013-2014",
    "committee_id": "NCC000168",
    "chamber": "lower",
    "state": "nc",
    "subcommittee": null,
    "committee": "Elections",
    "position": "member",
    "type": "committee member"
   },
```


(continues on next page)

(continued from previous page)


```
    {
     "term": "2013-2014",
     "committee_id": "NCC000088",
     "chamber": "lower",
     "state": "nc",
     "subcommittee": null,
     "committee": "Government",
     "position": "member",
     "type": "committee member"
    },
    {
     "term": "2013-2014",
     "committee_id": "NCC000107",
     "chamber": "lower",
     "state": "nc",
     "subcommittee": null,
     "committee": "Homeland Security, Military, and Veterans Affairs",
     "position": "member",
     "type": "committee member"
    },
    {
     "term": "2013-2014",
     "committee_id": "NCC000172",
     "chamber": "lower",
     "state": "nc",
     "subcommittee": null,
     "committee": "Public Utilities and Energy",
     "position": "member",
     "type": "committee member"
    }
   ],
   "url": "http://www.ncga.state.nc.us/gascripts/members/viewMember.pl?sChamber=House&
↪nUserID=679",
   "created_at": "2013-01-03 19:15:14",
   "chamber": "lower",
   "offices": [
    {
     "fax": null,
     "name": "Capitol Office",
     "phone": "919-733-5755",
     "address": "NC House of Representatives\n16 W. Jones Street, Room 1019\n\nRaleigh,
↪ NC 27601-1096",
     "type": "capitol",
     "email": null
    }
   ],
   "suffixes": ""
  }
]
```


2.4.4 Districts

Note: API v2 is now available, please check out API v2.

District Search List districts for state (and optionally filtered by chamber).

Methods

District Search

The district search method requires a `state` and can optionally also take a `chamber` as part of the URL.

The method returns a list of district objects with the following fields:

- `abbr` State abbreviation.
- `boundary_id` boundary_id used in *Examples*
- `chamber` Whether this district belongs to the upper or lower chamber.
- `id` A unique ID for this district (separate from boundary_id).
- `legislators` List of legislators that serve in this district. (may be more than one if `num_seats` > 1)
- `name` Name of the district (e.g. '14', '33A', 'Fifth Suffolk')
- `num_seats` Number of legislators that are elected to this seat. Generally one, but will be 2 or more if the seat is a multi-member district.

Example: *openstates.org/api/v1/districts/nc/lower/*

Examples

District Search

`openstates.org/api/v1/districts/nc/lower/`


```
[
 { "abbr": "nc",
  "boundary_id": "sldl/nc-1",
  "chamber": "lower",
  "id": "nc-lower-1",
  "legislators": [
   { "full_name": "Bob Steinburg", "leg_id": "NCL000302" }
  ],
  "name": "1",
  "num_seats": 1 },
 { "abbr": "nc",
  "boundary_id": "sldl/nc-12",
  "chamber": "lower",
  "id": "nc-lower-12",
  "legislators": [
   { "full_name": "George Graham", "leg_id": "NCL000281" }
  ],
  "name": "12",
  "num_seats": 1 },
 { "abbr": "nc",
  "boundary_id": "sldl/nc-13",
  "chamber": "lower",
  "id": "nc-lower-13",
  "legislators": [
   { "full_name": "Pat McElraft", "leg_id": "NCL000137" }
```


(continues on next page)

(continued from previous page)


```
  ],
  "name": "13",
  "num_seats": 1 },
 { "abbr": "nc",
  "boundary_id": "sldl/nc-14",
  "chamber": "lower",
  "id": "nc-lower-14",
  "legislators": [
   { "full_name": "George G Cleveland", "leg_id": "NCL000076" }
  ],
  "name": "14",
  "num_seats": 1 },
 { "abbr": "nc",
  "boundary_id": "sldl/nc-15",
  "chamber": "lower",
  "id": "nc-lower-15",
  "legislators": [
   { "full_name": "Phil R Shepard", "leg_id": "NCL000221" }
  ],
  "name": "15",
  "num_seats": 1 },
 ... truncated ...
]
```


2.4.5 Categorization

One of the ways that we add value to the data we provide is by attempting to categorize bills, actions, and votes across states.

Bill Types

Most state legislatures deal with more than bills. Despite the name of the bill objects in our data we take in all types of legislation that a state might produce. Generally looking at the bill_id will help you determine the type of legislation, but to make things easier across states we provide a "type" field on bills.

- bill
- resolution
- joint resolution
- concurrent resolution
- constitutional amendment

States with additional types might make use of additional types such as:

- contract
- nomination
- memorial
- ...

Action Types

Although most states follow very similar parlimentary procedure the names that their bill status systems use for various actions almost never match up. To make analysis and the building of certain types of tools easier we attempt to categorize about 30 types of common actions. In using our data you'll find these values in the type field of actions.

Possible values:

bill:introduced Bill is introduced or prefiled

bill:passed Bill has passed a chamber

bill:failed Bill has failed to pass a chamber

bill:withdrawn Bill has been withdrawn from consideration

bill:veto_override:passed The chamber attempted a veto override and succeeded

bill:veto_override:failed The chamber attempted a veto override and failed

bill:reading:1 A bill has undergone its first reading

bill:reading:2 A bill has undergone its second reading

bill:reading:3 A bill has undergone its third (or final) reading

bill:filed A bill has been filed (for states where this is a separate event from bill:introduced)

bill:substituted A bill has been replaced with a substituted wholesale (called hoghousing in some states)

governor:received The bill has been transmitted to the governor for consideration

governor:signed The bill has signed into law by the governor

governor:vetoed The bill has been vetoed by the governor

governor:vetoed:line-item The governor has issued a line-item (partial) veto

amendment:introduced An amendment has been offered on the bill

amendment:passed The bill has been amended

amendment:failed An offered amendment has failed

amendment:amended An offered amendment has been amended (seen in Texas)

amendment:withdrawn An offered amendment has been withdrawn

amendment:tabled An amendment has been 'laid on the table' (generally preventing further consideration)

committee:referred The bill has been referred to a committee

committee:passed The bill has been passed out of a committee

committee:passed:favorable The bill has been passed out of a committee with a favorable report

committee:passed:unfavorable The bill has been passed out of a committee with an unfavorable report

committee:failed The bill has failed to make it out of committee

other All other actions will have a type of "other"

Vote Types

Similarly to actions, we make an effort to categorize the motion being voted upon. You'll find these values in the type field of vote objects.

Possible values:

passage This is a vote to pass (either out of committee or a chamber)

amendment Vote on amending a bill

veto_override Vote to override an executive veto

reading:1 Vote on a first reading

reading:2 Vote on a second reading

other All other votes

Subjects

Many states provide a list of subject areas for individual pieces of legislation. We've made an attempt to map these to a comprehensive set of subjects.

If you're using the API data you'll find these in the subjects field if we've been able to categorize a state's bills. If you're interested in the state's native categories those can be found in "scraped_subjects"

Possible values:

- Agriculture and Food
- Animal Rights and Wildlife Issues
- Arts and Humanities
- Budget, Spending, and Taxes
- Business and Consumers
- Campaign Finance and Election Issues
- Civil Liberties and Civil Rights
- Commerce
- Crime
- Drugs
- Education
- Energy
- Environmental
- Executive Branch
- Family and Children Issues
- Federal, State, and Local Relations
- Gambling and Gaming
- Government Reform
- Guns
- Health
- Housing and Property
- Immigration
- Indigenous Peoples
- Insurance

- Judiciary
- Labor and Employment
- Legal Issues
- Legislative Affairs
- Military
- Municipal and County Issues
- Nominations
- Other
- Public Services
- Recreation
- Reproductive Issues
- Resolutions
- Science and Medical Research
- Senior Issues
- Sexual Orientation and Gender Issues
- Social Issues
- State Agencies
- Technology and Communication
- Trade
- Transportation
- Welfare and Poverty

2.5 Methods

Method	URL pattern	Description
Metadata Overview	/metadata/	Get list of all states with data available and basic metadata about their status.
State Metadata	/metadata/*state*/	Get detailed metadata for a particular state.
Bill Search	/bills/	Search bills by (almost) any of their attributes, or full text.
Bill Detail	/bills/*state*/*session*/*bill_id*/	Get full detail for bill, including any actions, votes, etc.
Legislator Search	/legislators/	Search legislators by their attributes.
Legislator Detail	/legislators/*leg_id*/	Get full detail for a legislator, including all roles.
Geo Lookup	/legislators/geo/?lat=latitude&long=longitude	Lookup all legislators that serve districts containing a given point.
District Search	/districts/*state*/[*chamber*/]	List districts for state (and optionally filtered by chamber).

CHAPTER 3

Static Data

We have several types of static data files available:

3.1 Boundary JSON

A static API for obtaining simplified GeoJSON describing the boundaries of legislative districts.

If you obtain an Open Civic Data Division ID (e.g. `ocd-division/country:us/state:ks/sldl:1` you can convert it to a URL like `https://data.openstates.org/boundaries/2018/ocd-division/country:us/state:ks/sldl:1.json`.

The URL consists of a few parts:

- `https://data.openstates.org/boundaries/` - base URL
- `2018` - division issue year, currently only 2018 supported (divisions do not change most years)
- `ocd-division/country:us/state:ks/sldl:1` - Open Civic Data Division ID
- `.json` - end URL with *.json*

3.1.1 File Format

- `division_id` - Open Civic Data Division ID, same as URL.
- `year` - Issue year for Boundary, same as URL.
- `centroid` - GeoJSON Point representing the centroid of this boundary.
- `extent` - Coordinates for upper-left & lower-right extent of boundary.
- `shape` - Simplified GeoJSON MultiPolygon for display purposes. This data has been simplified (losing some detail to gain speed when drawing/processing) and should be used for display but not intersection/point-in-polygon testing.

- `metadata` - Metadata that was provided by issuer, typically a bit of Census data. No particular guarantee is made about what is here.

3.1.2 Obtaining ocd-division IDs

If you're using Open States data you'll see ocd-division IDs in a few places:

- The division_id & boundary_id keys in the *Districts* endpoint of API v1.
- the id attribute on *DivisionNode* which can be accessed via Post.

3.2 People CSVs

As of April 2019 we are making experimental CSV files of **current** legislators available.

The latest files are available via: https://github.com/openstates/people/tree/master/csv

Note: Due to the limitations of the CSV format we can't include every field we collect in these files. The goal here is to make a file that has the fields people are most likely to want. The same Git repository also has YAML files available if you'd like historical roles, etc.

If you're using these files, you can give feedback by opening a GitHub issue: https://github.com/openstates/people/issues

3.3 Legacy JSON Data

The data contained in these downloads is in roughly the same format as that obtained via the legacy *Open States API v1*. (Refer to those docs for specifics on file formats.)

These downloads are not intended as a replacement for using the API. behind the current data. This makes them useful for general research purposes or initializing a database that will then be kept up to date via regular calls. If you'd like more current data please use the API.

Latest Update: **November 3, 2018**

The latest legacy JSON files can be obtained from a URL in the format: `https://data.openstates.org/legacy/json/ak.zip`

Just substitute the postal code of your state for the *ak* portion of the URL.

- Alabama
- Alaska
- Arizona
- Arkansas
- California
- Colorado
- Connecticut
- Delaware
- District of Columbia
- Florida

- Georgia
- Hawaii
- Idaho
- Illinois
- Indiana
- Iowa
- Kansas
- Kentucky
- Louisiana
- Maine
- Maryland
- Massachusetts
- Michigan
- Minnesota
- Mississippi
- Missouri
- Montana
- Nebraska
- Nevada
- New Hampshire
- New Jersey
- New Mexico
- New York
- North Carolina
- North Dakota
- Ohio
- Oklahoma
- Oregon
- Pennsylvania
- Puerto Rico
- Rhode Island
- South Carolina
- South Dakota
- Tennessee
- Texas
- Utah

- Vermont
- Virginia
- Washington
- West Virginia
- Wisconsin
- Wyoming

3.4 Legacy CSV Data

These files are a CSV transformation of the data available in the Legacy JSON archives. The CSV files do not reflect extra fields that the JSON files are able to include. This is due to an inherent limitation of the CSV format. If extra fields are required you should use the JSON downloads or API.

3.4.1 Format

For each state we provide a .zip file containing several CSV files.

The following CSV files exist:

legislators.csv

Basic information on legislators.

- *leg_id*: Open States ID for legislator
- *full_name*
- *first_name*
- *middle_name*
- *last_name*
- *suffixes*
- *nickname*
- *active*: True or False
- *state*: if active, state for current role
- *chamber*: if active, chamber for current role
- *district*: if active, district for current role
- *party*: if active, party for current role
- *votesmart_id*: ID on votesmart.org
- *transparencydata_id*: ID on Influence Explorer
- *photo_url*
- *created_at*: timestamp of when object was created in our system
- *updated_at*: timestamp of when object was last updated in our system

legislator_roles.csv

Links legislators (by id) to any roles that they've had.

- *leg_id*: Open States ID for legislator
- *type*: type of role (likely to be 'member' or 'committee member')
- *term*: term for which legislator held role
- *district*: if role is member, will be district served
- *chamber*: chamber of legislature (upper/lower)
- *state*: if role is member, will be state legislator was active in
- *party*: if role is member, will be legislator's party
- *committee_id*: if role is committee member, will be Open States ID for committee
- *committee*: if role if committee member, will be name of committee (parent if subcommittee)
- *subcommittee*: if role if committee member, will be name of subcommittee
- *start_date*: optional, start date for role
- *end_date*: optional, end date for role

committees.csv

List of committees (see legislator_roles.csv) for membership info.

- *id*: Open States ID for committee
- *state*
- *chamber*: may be upper, lower, or joint
- *committee*: name of committee (parent if subcommitee)
- *subcommittee*: name of subcommittee (blank if committee)
- *parent_id*: Open States ID for parent committee if this is a subcommitee

bills.csv

Basic details on all bills for a given state.

- *bill_id*: state-assigned bill id (eg. HB 271 or SR 10)
- *state*
- *session*
- *chamber*: upper or lower
- *title*: state-given title of bill
- *created_at*: timestamp of when object was created in our system
- *updated_at*: timestamp of when object was last updated in our system
- *type*: bill types, pipe delimited if multi-valued
- *subjects*: bill subjects, pipe delimited if multi-valued

bill_actions.csv

Listing of actions taken on all bills.

- *state*: uniquely identifies bill together with session, chamber, and bill_id
- *session*: uniquely identifies bill together with state, chamber, and bill_id
- *chamber*: uniquely identifies bill together with state, session, and bill_id
- *bill_id*: uniquely identifies bill together with session, chamber, and chamber
- *date*: date that action took place
- *action*: state-given name of action
- *actor*: often upper/lower, can be other values
- *type*: type of action

bill_sponsors.csv

Listing of sponsors across all bills.

- *state*: uniquely identifies bill together with session, chamber, and bill_id
- *session*: uniquely identifies bill together with state, chamber, and bill_id
- *chamber*: uniquely identifies bill together with state, session, and bill_id
- *bill_id*: uniquely identifies bill together with session, chamber, and chamber
- *type*: type of sponsor (primary/cosponsor/etc)
- *name*: given name of sponsor
- *leg_id*: Open States Legislator ID for sponsor

bill_votes.csv

A listing of all votes on all bills.

- *state*: uniquely identifies bill together with session, chamber, and bill_id
- *session*: uniquely identifies bill together with state, chamber, and bill_id
- *chamber*: uniquely identifies bill together with state, session, and bill_id
- *bill_id*: uniquely identifies bill together with session, chamber, and chamber
- *vote_id*: Open States Vote ID for this vote
- *vote_chamber*: chamber vote took place in (upper/lower)
- *motion*
- *date*
- *type*: type of vote
- *yes_count*: number of 'yes' votes
- *no_count*: number of 'no' votes
- *other_count*: number of 'other' votes

bill_legislator_votes.csv

Pairing of vote ids with how a specific legislator voted.

- *vote_id*: Open States Vote ID, matches *vote_id* from bill_votes.csv
- *leg_id*: Open States Legislator ID
- *name*: name of legislator
- *vote*: yes/no/other

3.4.2 Downloads

Latest Update: **November 3, 2018**

The latest legacy CSV files can be obtained from a URL in the format: `https://data.openstates.org/legacy/csv/ak.zip`

Just substitute the postal code of your state for the *ak* portion of the URL.

- Alabama
- Alaska
- Arizona
- Arkansas
- California
- Colorado
- Connecticut
- Delaware
- District of Columbia
- Florida
- Georgia
- Hawaii
- Idaho
- Illinois
- Indiana
- Iowa
- Kansas
- Kentucky
- Louisiana
- Maine
- Maryland
- Massachusetts
- Michigan
- Minnesota

- Mississippi
- Missouri
- Montana
- Nebraska
- Nevada
- New Hampshire
- New Jersey
- New Mexico
- New York
- North Carolina
- North Dakota
- Ohio
- Oklahoma
- Oregon
- Pennsylvania
- Puerto Rico
- Rhode Island
- South Carolina
- South Dakota
- Tennessee
- Texas
- Utah
- Vermont
- Virginia
- Washington
- West Virginia
- Wisconsin
- Wyoming

CHAPTER 4

Contributing to Open States

Open States' origin is as a community-driven project, and is only possible because of the sustained effort of dozens of volunteers.

The guides below will walk you through how we work, and various ways you can contribute:

4.1 Getting Started

No matter how experienced you are, it is a good idea to read this section before diving into Open States' code.

This guide assumes a basic familiarity with using the command line, git, and Python.

No worries if you aren't an expert though, we'll walk you through the steps. And as for Python, if you've written other languages like Javascript or Ruby you'll probably be just fine. Don't be afraid to *ask for help* either!

4.1.1 Installing docker

The first thing you will need to do is get a working docker environment on your local machine. We'll do this using Docker. No worries if you aren't familiar with Docker, you'll barely have to touch it beyond what this guide explains.

Install Docker and docker-compose (if not already installed on your local system):

(a) Installing Docker:

- On OSX: Docker for Mac
- On Windows: Docker for Windows
- On Linux: Use your package manager of choice or follow Docker's instructions.

(*Docker Compose is probably already installed by step 1(a) if not, proceed to step 1(b)*)

(b) Installing docker-compose:

- For easy installation on macOS, Windows, and 64-bit Linux.

Ensure that Docker and docker-compose are installed locally:

```
$ docker --version
Docker version 19.03.4, build 9013bf5
$ docker-compose --version
docker-compose version 1.24.1, build 4667896b
```

Of course, your versions will differ, but ensure they are relatively recent to avoid strange issues.

4.1.2 Installing pre-commit

To help keep the code as managable as possible we **strongly recommend** you use pre-commit to make sure all commits adhere to our preferred style.

- See pre-commit's installation instructions
- Within each repo you check out, run `pre-commit install` after checking out. It should look something like:

```
$ pre-commit install
pre-commit installed at .git/hooks/pre-commit
```

Note: If you're running `flake8` and `black` yourself via your editor or similar this isn't strictly necessary, but we find it helps ensure commits don't fail linting. **We require all PRs to pass linting!**

4.1.3 Get Familiar With Our Processes

We're glad to have you joining us, taking a few minutes to read the following pages will help you be a better member of our community:

- Our *Code of Conduct* is important to us, and helps us maintain a healthy community.
- We also have a *guide to help you learn where to get help* that you should look over.
- There's also a *repository overview and roadmap* that can be helpful context for the work we're doing here.

4.2 Contributing to Scrapers

Scrapers are at the core of what Open States does, each state requires several custom scrapers designed to extract bills, legislators, committees, and votes from the state website. All together there are around 200 scrapers, each one essentially independent, which means that there is always more work to do, but fortunately plenty of prior work to learn from.

4.2.1 Checking out

Fork and clone the main scraper repository:

- Visit https://github.com/openstates/openstates and click the 'Fork' button.
- Clone your fork using your tool of choice or the command line:

```
$ git clone git@github.com:yourname/openstates.git
Cloning into 'openstates'...
```

- And remember to *install pre-commit*:

```
$ pre-commit install
pre-commit installed at .git/hooks/pre-commit
```

Warning: Before cloning on a Windows computer, you will need to disable line-ending conversion. `git config --global core.autocrlf false` After cloning and entering the repo, you'll likely want to set global line-ending conversion back to *true*, and set local conversion to *false*.

4.2.2 Repository overview

At this point you'll have a local `openstates` directory. Within it, you'll find a directory called `openstates` lets take a look at it:

```
$ ls openstates
__init__.py dc          in          mn          nj          pr          va
ak          de          ks          mo          nm          ri          vi
al          fl          ky          ms          nv          sc          vt
ar          ga          la          mt          ny          sd          wa
az          hi          ma          nc          oh          tn          wi
ca          ia          md          nd          ok          tx          wv
co          id          me          ne          or          ut          wy
ct          il          mi          nh          pa          utils
```

This directory is a python module with 50+ subpackages, one for each state.

Let's look inside one:

```
$ ls openstates/nc
__init__.py    bills.py       committees.py  people.py      votes.py
```

Some states' directories will differ a bit, but all will have `__init__.py` and `bills.py`.

The `__init__.py` file for each state has basic metadata on the state including a list of sessions.

Other files contain the scrapers, typically named `bills`, `votes`, etc.

4.2.3 Running Your First Scraper

Let's run your state's bills scraper (substitute your state for 'nc' below)

```
$ docker-compose run --rm scrape nc bills --fastmode --scrape
```

The parameters you pass after `docker-compose run --rm scrape` are passed to `pupa update`. Here we're saying that we're running NC's scrapers, and that we want to do it in "fast mode". By default, `pupa update` imports results into a postgres database; the `--scrape` flag skips that step.

You'll see the *run plan*, which is what `pupa` aims to capture; in this case we're scraping the state website's data into JSON files:

```
loaded Open States pupa settings...
nc (scrape)
  bills: {}
```

Then legislative posts and organizations get created, which is mostly boilerplate:

```
08:46:35 INFO pupa: save jurisdiction North Carolina as jurisdiction_ocd-jurisdiction-
↪country:us-state:nc-government.json
08:46:35 INFO pupa: save organization North Carolina General Assembly as organization_
↪01d6327c-72d2-11e7-8df8-0242ac130003.json
08:46:35 INFO pupa: save organization Executive Office of the Governor as␣
↪organization_01d63560-72d2-11e7-8df8-0242ac130003.json
08:46:35 INFO pupa: save organization Senate as organization_01d636e6-72d2-11e7-8df8-
↪0242ac130003.json
08:46:35 INFO pupa: save post 1 as post_01d63a06-72d2-11e7-8df8-0242ac130003.json
08:46:35 INFO pupa: save post 2 as post_01d63b96-72d2-11e7-8df8-0242ac130003.json
08:46:35 INFO pupa: save post 3 as post_01d63cea-72d2-11e7-8df8-0242ac130003.json
08:46:35 INFO pupa: save post 4 as post_01d63e34-72d2-11e7-8df8-0242ac130003.json
08:46:35 INFO pupa: save post 5 as post_01d63f74-72d2-11e7-8df8-0242ac130003.json
```

And then the actual data scraping begins, defaulting to the most recent legislative session:

```
08:46:36 INFO pupa: no session specified, using 2017
08:46:36 INFO scrapelib: GET - http://www.ncga.state.nc.us/gascripts/
↪SimpleBillInquiry/displaybills.pl?Session=2017&tab=Chamber&Chamber=Senate
08:46:38 INFO scrapelib: GET - http://www.ncga.state.nc.us/gascripts/BillLookUp/
↪BillLookUp.pl?Session=2017&BillID=S1
08:46:39 INFO pupa: save bill SR 1 in 2017 as bill_03c7edb4-72d2-11e7-8df8-
↪0242ac130003.json
08:46:39 INFO scrapelib: GET - http://www.ncga.state.nc.us/gascripts/BillLookUp/
↪BillLookUp.pl?Session=2017&BillID=S2
08:46:39 INFO pupa: save bill SJR 2 in 2017 as bill_044a5fc4-72d2-11e7-8df8-
↪0242ac130003.json
08:46:39 INFO scrapelib: GET - http://www.ncga.state.nc.us/gascripts/BillLookUp/
↪BillLookUp.pl?Session=2017&BillID=S3
08:46:40 INFO pupa: save bill SB 3 in 2017 as bill_04e8c66e-72d2-11e7-8df8-
↪0242ac130003.json
08:46:40 INFO scrapelib: GET - http://www.ncga.state.nc.us/gascripts/BillLookUp/
↪BillLookUp.pl?Session=2017&BillID=S4
08:46:41 INFO pupa: save bill SB 4 in 2017 as bill_05781f08-72d2-11e7-8df8-
↪0242ac130003.json
08:46:41 INFO scrapelib: GET - http://www.ncga.state.nc.us/gascripts/BillLookUp/
↪BillLookUp.pl?Session=2017&BillID=S5
```

Depending on the scraper you run, this part takes a while. Some scrapers can take hours to run depending on the number of bills and speed of the state's website.

Note: It is often desirable to bail out of running the whole scrape (Ctrl-C) after it has gotten a bit of data, instead of letting it run the entire scrape.

To review the data you just fetched, you can browse the _data/nc/ directory and inspect the JSON files. If you're trying to make a small fix this is often sufficient, you can confirm that the scraped data looks correct and move on.

Note: It is of course possible that the scrape fails. If so, there's a good chance that isn't your fault, especially if it starts to run and then errors out. Scrapers do break, and there's no guarantee North Carolina didn't change their legislator page yesterday, breaking our tutorial here.

If that's the case and you think the issue is with the scraper, feel free to get in touch with us or file an issue.

At this point you're ready to run scrapers and contribute fixes. Hop onto our GitHub ticket queue, pick an issue to solve, and then submit a Pull Request!

4.2.4 Importing Data

Optionally, if you'd like to see how your scraped data imports into the database, perhaps to diagnose an issue that is happening after the scrape, pop over to *getting a working database* to see how to get a local database that you can import data into.

Once that's done, make sure that the db image from openstates.org is running:

```
$ docker ps
CONTAINER ID        IMAGE                          COMMAND                  CREATED
↪       STATUS              PORTS                     NAMES
27fe691ad7c5        mdillon/postgis:11-alpine   "docker-entrypoint.s..."   3 hours
↪ago          Up 3 hours          0.0.0.0:5405->5432/tcp   openstatesorg_db_1
```

Your output will vary, but if you don't see something named openstatesorg_db running you should run this command (from the openstates.org directory, not your scraper directory):

```
$ docker-compose up -d db
```

Now, when you want to run imports, you can drop the `--scrape` portion of the command you've been running. Or if you just want to test the import of already scraped data you can replace it with `--import`.

An import looks something like this:

```
$ docker-compose run --rm scrape fl bills --fast
... (truncated) ...
23:03:34 ERROR pupa: cannot resolve pseudo id to Person: ~{"name": "Grant, M."}
23:03:36 ERROR pupa: cannot resolve pseudo id to Person: ~{"name": "Rodrigues, R."}
fl (import)
  bills: {}
import:
  bill: 0 new 0 updated 2620 noop
  jurisdiction: 0 new 0 updated 1 noop
  vote_event: 21 new 12 updated 533 noop
```

The errors about unresolved psuedo-ids can safely be ignored, as long as you see the final run report the data you scraped is available in your database.

The number of objects of each type that were created & updated are available for spot checking, as well as the total number of items that were seen that already exactly matched what was in the database. These can be useful stats as you try to see if your local changes to a scraper have the impact you expect.

4.3 Working On openstates.org

openstates.org is the public-facing result of all the scraping we do. The site is built in Django and includes the web frontend and API.

4.3.1 Checking out

Fork and clone the openstates.org repository:

- Visit https://github.com/openstates/openstates.org and click the 'Fork' button.
- Clone your fork using your tool of choice or the command line:

```
$ git clone git@github.com:yourname/openstates.org.git
Cloning into 'openstates.org'...
```

- And remember to *install pre-commit*:

```
$ pre-commit install
pre-commit installed at .git/hooks/pre-commit
```

4.3.2 Getting a working database

Whether you're aiming to work on openstates.org or just want to import scraped data, you'll need postgres server running in your docker environment.

If you haven't set up docker yet, see *Installing docker*.

There's a one line command that will download a recent copy of the Open States database and restore it:

```
docker-compose run --rm --entrypoint ./docker/init-db.sh django
```

Warning: This command takes several GB of disk space. The pgdump file that is downloaded is ~2GB as of December 2019 and the restored database is around 7GB. If you don't have room for this but want to contribute let us know so we can prioritize more compact options.

You'll see this command download a file from S3, which can take a while depending upon your internet connection. It will then go silent for a while as it works to restore the database. This takes 5-10 minutes on a late-2018 Macbook Pro, but your experience may vary. So long as it isn't spitting out errors, things should be fine.

If you're working on scrapers you'll now find that this database is available to your scrape processes!

4.3.3 Repository overview

TODO

4.3.4 Running openstates.org

Simply running `docker-compose up` should start django & the database, then browse to http://localhost:8000 and you'll be looking at your own local copy of openstates.org

More coming soon!

4.4 Contributing People Data

Person data is maintained in the openstates/people repository. This repository contains YAML files with all the information on given individuals, as well as scripts to work with & maintain the data.

Also, please note that this portion of the project is in the public domain in the United States with all copyright waived via a CC0 dedication. By contributing you agree to waive all copyright claims.

4.4.1 Checking out

Fork and clone the people repository:

- Visit https://github.com/openstates/people and click the 'Fork' button.
- Clone your fork using your tool of choice or the command line:

```
$ git clone git@github.com:yourname/people.git
Cloning into 'people'..
```

- And remember to *install pre-commit*:

```
$ pre-commit install
pre-commit installed at .git/hooks/pre-commit
```

4.4.2 Repository overview

The repository consists of a few key components:

- `settings.yml` Settings for state legislatures, including the number of seats, and current vacancies.
- `data/` Data files in YAML format on legislators, organized by state & status.
- `scripts` Various scripts used to maintain the data.
- `scrape/` Experimental new people scrapers, work-in-progress.

To run a script using `docker-compose` you can run a command like:

```
docker-compose run --rm people ./scripts/lint_yaml.py
```

4.4.3 Common tasks

Updating legislator data by hand

Let's say you call a legislator and find out that they have a new phone number, contribute back!

See schema.md for details on the acceptable fields. If you're looking to add a lot of data but unsure where it fits feel free to ask via an issue and we can either amend the schema or make a recommendation.

0. Start a new branch for this work
1. Make the edits you need in the appropriate YAML file. Please keep edits to a minimum (e.g. don't re-order fields)
2. Submit a PR, please describe how you came across this information to expedite review.

Retiring a legislator

0. Start a new branch for this work
1. Run `./scripts/retire.py` on the appropriate legislator file(s)
2. Review the automatically edited files & submit a PR.

Updating an entire state via a scrape

Let's say a North Carolina has had an election & it makes sense to re-scrape everything for that state.

0. Start a new branch for this work
1. Scrape data using Open States' Scrapers
2. Run `./scripts/to_yaml.py` against the generated JSON data, this will populate the incoming/ directory
3. Check for merge candidates using `./scripts/merge.py --incoming nc`
4. Manually reconcile remaining changes, will almost certainly require some retirements as well.
5. Check that data looks clean with `./scripts/lint_yaml.py nc --summary` and prepare a PR.

Updating a single field for many people

Let's say you want to add foobar_id to a ton of legislators from your own data set or similar.

TBD - We need to create a tool that will aid in this as it will prove a common use case & we can lower the barrier here.

4.4.4 Scripts

Several scripts are provided to help maintain/check the data.

to_yaml.py

```
to_yaml.py [OPTIONS] INPUT_DIR

  Convert pupa scraped JSON in INPUT_DIR to YAML files for this repo.

Convert a pupa scrape directory to YAML.  Will put data into incoming/
directory for usage with merge.py's --incoming option.
```

lint_yaml.py

```
lint_yaml.py [OPTIONS] [ABBREVIATIONS]

  Lint YAML files, optionally also providing a summary of state's data.

  <ABBREVIATIONS> can be provided to restrict linting to select states.

Options:
  -v, --verbose
  --summary / --no-summary  Print summary after validation errors.
```

merge.py

```
merge.py [OPTIONS]

  Script to assist with merging legislator files.

```

(continues on next page)

(continued from previous page)

```
  Can be used in two modes: incoming or file merge.

  Incoming mode analyzes incoming/ directory files (generated with
  to_yaml.py) and discovers identical & similar files to assist with
  merging.

  File merge mode merges two legislator files.

Options:
  --incoming TEXT    Operate in incoming mode, argument should be state abbr
                     to scan.
  --retirement TEXT  Set retirement date for all people marked retired (in
                     incoming mode).
  --old TEXT         Operate in merge mode, this is the older of two files &
                     will be kept.
  --new TEXT         In merge mode, this is the newer file that will be
                     removed after merge.
  --help             Show this message and exit.
```

new_person.py

```
new_person.py [OPTIONS]

Create a new person record.

  Arguments can be passed via command line flags, omitted arguments will be
  prompted.

  Be sure to review the file and add any additional data before committing.

Options:
  --fname TEXT       First Name
  --lname TEXT       Last Name
  --name TEXT        Optional Name, if not provided First + Last will be used
  --state TEXT       State abbreviation
  --district TEXT    District
  --party TEXT       Party
  --rtype TEXT       Role Type
  --url TEXT         Source URL
  --image TEXT       Image URL
  --start-date TEXT  Start Date YYYY-MM-DD
```

retire.py

```
retire.py [OPTIONS] END_DATE FILENAME

  Retire a legislator, given END_DATE and FILENAME.

  Will set end_date on active roles & committee memberships.
```

to_database.py

```
to_database.py [OPTIONS] [ABBREVIATIONS]

  Sync YAML files to DB.

Options:
  --purge / --no-purge  Purge all legislators from DB that aren't in YAML.
  --safe / --no-safe    Operate in safe mode, no changes will be written to
                        database.
```

sync_images.py

```
sync_images.py [OPTIONS] [ABBREVIATIONS]...

  Download images and sync them to S3.

  <ABBR> can be provided to restrict to single state.

Options:
  --skip-existing / --no-skip-existing  Skip processing for files that already exist
                                        on S3. (default: true)
```

to_csv.py

```
to_csv.py [ABBREVIATIONS]...

  Update CSVs of current legislators.

  <ABBR> can be provided to restrict to single state.
```

4.5 Text Extraction

The *bill scrapers* scrape the web and pull down metadata, including links to various versions of the bills. As a later step, we extract the actual text of the bill so that it can be indexed for search and other uses.

4.5.1 Checking out

Fork and clone the text-extraction repository:

- Visit https://github.com/openstates/text-extraction and click the 'Fork' button.
- Clone your fork using your tool of choice or the command line:

```
$ git clone git@github.com:yourname/text-extraction.git
Cloning into 'text-extraction'...
```

- And remember to *install pre-commit*:

```
$ pre-commit install
pre-commit installed at .git/hooks/pre-commit
```

4.5.2 Repository overview

The text extraction code itself is written as a standalone Python script `text_extract.py` that uses configuration and utility functions from within `extract/`.

You'll also notice a directory called `raw/` – this contains a sampling of bills for each state that we can use to test text-extraction.

Typically if you're making changes in the repository you'll be editing files within `extract/`, we'll come back to that later.

4.5.3 Running text_extract

Just like in other repositories, we'll use docker-compose to run the code. In this case docker-compose is running `text_extract.py`, an all-in-one tool that has a few useful subcommands:

```
Usage: text_extract.py [OPTIONS] COMMAND [ARGS]...

Options:
  --help  Show this message and exit.

Commands:
  reindex-state  rebuild the search index objects for a given state
  sample         obtain a sample of bills to extract text from
  status         print a status table showing the current condition of...
  test           run sample on all states, used for CI
  update         update the saved bill text in the database
```

For the purposes of development, `sample` and `update` are the only two commands that you'll need to look at.

Let's go ahead and run sample against NC:

```
$ docker-compose run --rm text-extract sample nc
raw/nc/2017-HR 924-Edition 1.pdf => text/nc/2017-HR 924-Edition 1.pdf.txt (1507 bytes)
raw/nc/2017-HB 1034-Edition 1.pdf => text/nc/2017-HB 1034-Edition 1.pdf.txt (3096␣
↪bytes)
raw/nc/2019-SB 421-Edition 1.pdf => text/nc/2019-SB 421-Edition 1.pdf.txt (961 bytes)
raw/nc/2019-HB 430-Edition 1.pdf => text/nc/2019-HB 430-Edition 1.pdf.txt (4831 bytes)
raw/nc/2017-SB 753-Edition 1.pdf => text/nc/2017-SB 753-Edition 1.pdf.txt (719 bytes)
raw/nc/2019-HB 788-Edition 1.pdf => text/nc/2019-HB 788-Edition 1.pdf.txt (2674 bytes)
raw/nc/2017-SB 373-Filed.pdf => text/nc/2017-SB 373-Filed.pdf.txt (18538 bytes)
raw/nc/2019-SB 574-Filed.pdf => text/nc/2019-SB 574-Filed.pdf.txt (1712 bytes)
raw/nc/2017-SJR 686-Resolution 2017-12.pdf => text/nc/2017-SJR 686-Resolution 2017-12.
↪pdf.txt (15928 bytes)
raw/nc/2017-HB 1007-Filed.pdf => text/nc/2017-HB 1007-Filed.pdf.txt (6248 bytes)
nc: processed 10, 0 skipped, 0 missing, 0 empty
```

The exact output and number of bills will vary across states, but should be pretty similar.

This command just did a lot:

- Read in the file `raw/nc.csv` to get a list of bills to sample.
- Downloaded those files (assuming this was the first run) to `raw/nc/` so future runs will be faster.
- Used the extraction function(s) defined in `extract/__init__.py` for NC to extract text from the given documents.
- Wrote that output to `text/nc/` so you can compare.

You'll also notice that it helpfully prints the number of bytes of text extracted, this is useful as a first check. Let's go ahead and look at the shortest one, `text/nc/2017-SB 753-Edition 1.pdf.txt`. (Your run may differ, pick whichever you prefer.)

```
$ cat "text/nc/2017-SB 753 Edition 1.pdf.txt"
A BILL TO BE ENTITLED
AN ACT PROVIDING THAT THE DEPOSIT OF CURRENCY AND COINS INTO A CASH
VAULT THAT PHYSICALLY SECURES THE CASH AND ELECTRONICALLY
RECORDS THE DEPOSIT DAILY IN AN OFFICIAL DEPOSITORY BANK QUALIFIES
AS A DAILY DEPOSIT UNDER THE LOCAL GOVERNMENT BUDGET AND FISCAL
CONTROL ACT FOR FRANKLIN AND WAKE COUNTIES AND THE
MUNICIPALITIES IN THOSE COUNTIES.
The General Assembly of North Carolina enacts:
SECTION 1. Section 2 of S.L. 2011-89 reads as rewritten:
"SECTION 2. This act applies only to the City of Winston-Salem only.Winston-Salem,
Franklin County and the municipalities in Franklin County, and Wake County and the
municipalities in Wake County."
SECTION 2. This act is effective when it becomes law.
```

This looks complete, but to check, go ahead and open the equivalent source file, in this case `raw/nc/2017-SB 753-Edition 1.pdf` and confirm visually that all the text was extracted. Don't worry about formatting, or the preamble, as we'll often exclude that and just aim for the interesting bits of the text.

4.5.4 Making changes

Let's say that we discover that a state has started publishing their bills in a new format. Perhaps Alabama switches from PDF to HTML. It'd first be good to add some of these new bills to the sample csv, which you can do manually or by invoking sample with the `--resample` flag.:

```
docker-compose run --rm text-extract sample --resample al
```

Running would result in some warnings being printed and some zero byte files.

To actually handle the HTML documents we'd open up `extract/__init__.py` and find the `CONVERSION_FUNCTIONS` dictionary, you'll see a line like:

```
CONVERSION_FUNCTIONS = {
    "al": {"application/pdf": extract_line_numbered_pdf},
  ...
```

The way extraction works is by matching a document found in a scrape to an appropriate function, in this case PDFs will be sent through the `extract_line_numbered_pdf` function.

If the new HTML was wrapped in a given element, perhaps with `<div id="billtext">` we could just update that line to look like:

```
CONVERSION_FUNCTIONS = {
    "al": {
        "application/pdf": extract_line_numbered_pdf,
        "text/html": extractor_for_element_by_id("billtext"),
    },
  ...
```

And we'd be good to go.

4.5.5 Tips & Tricks

- Functions already exist for common configurations of PDF, HTML, Word Doc, and even OCR. Rarely will you need to write a custom function, always look at the options first.
- When dealing with PDFs, most are either handled by `extract_line_numbered_pdf` or `extract_sometimes_numbered_pdf`, the difference is that "sometimes numbered PDF" accounts for cases where 90% or so of bills are numbered, but a few (often resolutions) are not numbered.

4.5.6 Formatting Guidelines

How far do we go? Should we strip punctuation? Newlines? Whitespace? Section headings?

- Try not to be too aggressive with punctuation stripping, search indices/etc. can easily strip it later, but it can be handy if someone decides they want to search for things like "§ 143C-4-8.b"
- Ideally leave newlines as-is since it makes looking at changes a lot nicer for humans and stripping newlines out for final products (search/text comparison/etc.) is trivial.
- Collapsing spaces/etc. is recommended, but not required.
- Removal of section headers/etc. is fine, but if the only reason you're writing a new function instead of using a common one is to do this, reconsider.

When in doubt, **ask**, you may have encountered something we haven't considered yet and we can discuss the best practice and add it here.

Should we include bill digests?

There isn't a need to, but it doesn't hurt if separating the two is difficult.

What about additions & deletions?

See text-extraction issue #6.

4.6 Code of Conduct

4.6.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

4.6.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience

- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

4.6.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

4.6.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

4.6.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at contact@openstates.org. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

4.6.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the project community.

4.6.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

4.7 Communication

When joining a new community, it can be tough to figure out *where* to ask questions, provide feedback, or help out. Don't worry! As long as you're respectful and follow our *Code of Conduct*, we're happy to have you!

Here are some guidelines regarding the best way to get in touch or contribute. Do note that Open States is a volunteer-powered project, and all of the core developers have day jobs; we're excited to talk to you, but it may sometimes take a bit of time to get back to you.

4.7.1 Recommendations

Want to ask a general question, have a conversation, or keep up with the community?

> The best place is our Discourse Forum. This is the main place for Open States discussion. The core team and many other contributors are present there, and we're usually able to answer questions in a timely fashion.

Have a private or financial question, or a security concern?

Email contact@openstates.org; only the administrative team can see these.

Have you found an error or issue in the Open States data?

File an issue on our bug tracker. And before you do, quickly check whether anyone else there has already reported the same bug.

Have a technical issue not related to the data itself?

Try to find the appropriate repository in our GitHub organization, and file an issue there. For example:

- openstates.org: https://github.com/openstates/openstates.org/issues/
- our documentation: https://github.com/openstates/documentation/issues/

Want to contribute to the project more regularly?

We also have a private Slack that we can invite you to. If you're interested in an invite, introduce yourself in an email to contact@openstates.org.

4.7.2 Discouraged Methods of Communication

Please *avoid* using these channels to get in touch with us:

Personal email addresses of Open States developers

Please respect our boundaries & refrain from contacting any of the developers directly, unless we ask you to do so.

Twitter (or any other social media)

We mainly use the @openstates twitter account to make announcements, and don't have the resources to provide technical support or other feedback on Twitter.

4.8 Overview & Roadmap

Open States is a fairly large & somewhat complex project comprised of many moving parts with a long history.

As you look to contribute, it may be beneficial to understand a little bit about the various components.

4.8.1 Repositories

These repositories make up the core of the project, if you're looking to contribute there's a 90% chance one of these is what you're looking for.

- openstates - Open States bill & vote scrapers.
- people - Open States People data, maintained as editable YAML files
- text-extraction - Text extraction powering full text search.
- openstates.org - Powers https://openstates.org/ website & API.
- documentation - https://docs.openstates.org/ (you're reading it now)

These repositories are other pieces of our infrastructure, but are generally not interesting to the average person.

- task-definitions - Definitions for bobsled.
- maintenance-scripts - Internal scripts used for various maintenance tasks.
- indiana-docs - A proxy for fetching Indiana's docs without API key.

- blog - https://blog.openstates.org/
- openstates-district-maps - Source for generating Open States' map tiles.

4.8.2 2020 Roadmap

Our current priorities:

Power User Features

- Add user logins & profiles. (Q1)
- Introduce bill & issue tracking. (Q1)

Data Quality

- Improve data quality and timeliness. (Ongoing)
- Provide publicly accessible data quality dashboard. (Q2)

API

- Improve speed of most popular graph queries. (TBD)
- Provide simplified endpoints for common queries. (TBD)
- Introduce a pub/sub type mechanism for staying in sync with bill & vote updates. (TBD)

Bulk Data

- Add new per-state CSV data exports. (Q1)
- Add custom data-export creation page. (TBD)
- Provide bulk geographic data ahead of 2021 redistricting. (TBD)

Community

- Documentation updates (Q1)
- New Contributor Support (Q1-Q2)
- API User Dashboard (Q2-Q3)

4.8.3 Recent Major Work

To give a sense of recent priorities, here are major milestones from the past few years:

- **Restoration of Historical Legislator Data** - Q4 2019
- Full Text Search - Q4 2019
- **2019 Legislative Session Updates** - Q1 2019
- OpenStates.org 2019 rewrite - Q1 2019

- OpenStates GraphQL API - Q4 2018
- **Scraper Overhaul** - Throughout much of 2017 we reworked our scrapers to be more resilient and to use an updated tech stack, replacing the one that powered the site from 2011-2016.

CHAPTER 5

Usage

Open States is used by major media organizations, political parties, nonprofits, and for-profit entities. Our Privacy Policy of course prohibits us from sharing information on specific users without their permission, but quite a few users have allowed us to share information on their projects.

If you'd like your project to be included here, please either open a pull request or issue on the documentation repo.

5.1 Notable Uses

- NPR StateImpact
- The Chicago Tribune
- MinnPost Legislative Bill Tracker

5.2 Open Source Projects

- Jim Grenadier built a C# Library & Azure Bot that uses the GraphQL API, source is available here: https://github.com/jgrenadier/OpenStatesGraphCS

CHAPTER 6

Open States Infrastructure

Warning: This section is out of date and will be rewritten soon.

Open States has grown to be a rather complex project over the years. The purpose of this documentation is to help explain how Open States works end-to-end to better help contributors and project members grasp the entirety of the system.

6.1 Overview

A great deal of Open States' infrastructure falls within the billy project. Here's roughly how it works:

- A scraper is written that utilizes `billy.scrape`'s Python helpers.
- When invoked via `billy-update`, this scraper writes JSON files to disk.
- These JSON files are then imported by `billy.importers` into MongoDB.
- A denormalization step takes place allowing us to have aggregate info, `billy.reports`.

At this point the data has been scraped, validated, post-processed, imported, and we've generated some statistics like how many bills were updated, etc.

From here, the data is served out via a Django project. `billy` also helps with this, in the form of three applications:

`billy.web.public` Essentially the site you see when you browse OpenStates.org.

`billy.web.api` Open States API v1

`billy.web.admin` This is a custom-built admin, truthfully just a series of views that project maintainers have found useful over time. There are some tools for manual data entry/reconciliation as well as error reporting.

Note: Now that the project is more community-driven and doesn't have a full-time staff, this approach is somewhat outdated and we'll be looking to improve access so that non-staff can help with some of the tasks included in the admin.

CHAPTER 7

Policies

Standardizing legislative information across states is a hairy task and sometimes we have to make compromises. We'll do our best to both articulate the policy and (where necessary) the rationale.

7.1 Categorization

One of the ways that we add value to the data we provide is by attempting to categorize bills, actions, and votes across states.

7.1.1 Bill Types

State legislatures deal with more than bills. Despite the name of the bill objects in our data we take in all types of legislation that a state might produce. Generally looking at the bill_id will help you determine the type of legislation, but to make things easier across states we provide a *type* field on bills. This field is a list with one (or more) of the following values:

Common Values: * bill * resolution * joint resolution * concurrent resolution * constitutional amendment

Some states also make use of additional types such as 'contract', 'nomination', 'memorial' and more.

7.1.2 Action Types

Although most states follow very similar parlimentary procedure the names that their bill status systems use for various actions almost never match up. To make analysis and the building of certain types of tools easier we attempt to categorize about 30 types of common actions. In using our data you'll find these values in the *type* field of actions.

- **bill:introduced** - Bill is introduced or prefiled
- **bill:passed** - Bill has passed a chamber
- **bill:failed** - Bill has failed to pass a chamber
- **bill:withdrawn** - Bill has been withdrawn from consideration

- **bill:veto_override:passed** - The chamber attempted a veto override and succeeded
- **bill:veto_override:failed** - The chamber attempted a veto override and failed
- **bill:reading:1** - A bill has undergone its first reading
- **bill:reading:2** - A bill has undergone its second reading
- **bill:reading:3** - A bill has undergone its third (or final) reading
- **bill:filed** - A bill has been filed (for states where this is a separate event from bill:introduced)
- **bill:substituted** - A bill has been replaced with a substituted wholesale (called hoghousing in some states)
- **governor:received** - The bill has been transmitted to the governor for consideration
- **governor:signed** - The bill has signed into law by the governor
- **governor:vetoed** - The bill has been vetoed by the governor
- **governor:vetoed:line-item** - The governor has issued a line-item (partial) veto
- **amendment:introduced** - An amendment has been offered on the bill
- **amendment:passed** - The bill has been amended
- **amendment:failed** - An offered amendment has failed
- **amendment:amended** - An offered amendment has been amended (seen in Texas)
- **amendment:withdrawn** - An offered amendment has been withdrawn
- **amendment:tabled** - An amendment has been 'laid on the table' (generally preventing further consideration)
- **committee:referred** - The bill has been referred to a committee
- **committee:passed** - The bill has been passed out of a committee
- **committee:passed:favorable** - The bill has been passed out of a committee with a favorable report
- **committee:passed:unfavorable** - The bill has been passed out of a committee with an unfavorable report
- **committee:failed** - The bill has failed to make it out of committee
- **other** - All other actions will have a type of "other"

7.1.3 Vote Types

Similarly to actions, we make an effort to categorize the motion being voted upon. You'll find these values in the *type* field of vote objects in billy or the *categorization* field in pupa.

Possible values:

- **passage** - This is a vote to pass (either out of committee or a chamber)
- **amendment** - Vote on amending a bill
- **veto_override** - Vote to override an executive veto
- **reading:1** - Vote on a first reading
- **reading:2** - Vote on a second reading
- **other** - All other votes

7.1.4 Subjects

Many states provide a list of subject areas for individual pieces of legislation. We've made an attempt to map these to a comprehensive set of subjects.

If you're using the API data you'll find these in the *subjects* field if we've been able to categorize a state's bills. If you're interested in the state's native categories those can be found in *scraped_subjects* field.

- Agriculture and Food
- Animal Rights and Wildlife Issues
- Arts and Humanities
- Budget, Spending, and Taxes
- Business and Consumers
- Campaign Finance and Election Issues
- Civil Liberties and Civil Rights
- Commerce
- Crime
- Drugs
- Education
- Energy
- Environmental
- Executive Branch
- Family and Children Issues
- Federal, State, and Local Relations
- Gambling and Gaming
- Government Reform
- Guns
- Health
- Housing and Property
- Immigration
- Indigenous Peoples
- Insurance
- Judiciary
- Labor and Employment
- Legal Issues
- Legislative Affairs
- Military
- Municipal and County Issues
- Nominations

- Other
- Public Services
- Recreation
- Reproductive Issues
- Resolutions
- Science and Medical Research
- Senior Issues
- Sexual Orientation and Gender Issues
- Social Issues
- State Agencies
- Technology and Communication
- Trade
- Transportation
- Welfare and Poverty

7.2 Session Naming

States name their sessions drastically differently, and sometimes inconsistently even within their own site. (49th vs 2008 Regular Session). As our goal is to help smooth these inconsistencies we put forward this guide to naming sessions within state metadata. (See https://github.com/sunlightlabs/openstates/issues/81 for discussion on the topic)

7.2.1 Default Session Names

The `sessions` list within `terms` is dangerous to change as all bill data is keyed off it. As a rule these should be short and generally useful for the scraper to make the appropriate decisions on what data to scrape.

If a state calls its 1st special session in 2010 '2010E1' this is a perfectly acceptable name for the session in the metadata. Similarly 49th-regular, 2009-Special-B, etc. are fine names. Generally names with spaces should be avoided simply for ease of construction of URLs, etc. In states where spaces are already in use it is fine to continue to use them.

The one caveat is that if a state uses a unique ID that has no bearing on the session itself such as '7323' for the 2011 session, this *should not* be used. Instead add some mapping that maps a session name that is descriptive to their internal ids.

7.2.2 Session Display Names

Because the most convenient name to refer to a session is often far from what a user might expect to see upon opening a mobile application, the `session_details` dict supports a `display_name` key.

Suitable display names are descriptive but also short and obey a given style.

General Rules

- All sessions should be in title case.
- Fewer than 20 characters is highly preferable.
- Months should be abbreviated to 3 letters (Jan., Feb., Jun., Dec.)

Ordinals

If no special sessions are present:

- [Ordinal] Legislature

If special sessions are present:

- [Ordinal] Regular Session
- [Ordinal], [Ordinal] Special Session

Examples:

- 82nd Legislature
- 82nd Regular Session
- 82nd, 3rd Special Session

Years

- [Year/Year-Range] Regular Session
- [Year/Year-Range], [Ordinal] Special Session
- [Mon. Year] Special Session

Examples:

- 2010 Regular Session
- 2011-2012, 4th Special Session
- Dec. 2011 Special Session

7.3 State API Keys

Unfortunately, some states find it necessary to require API Keys (or other credentials) to access their best data.

Despite the difficulties this creates for contributors, in the interest of ensuring we have the best possible data we've made the decision that we will use this data where possible.

Our policy:

- We will maintain (when possible) two copies of credentials, one for development and one for production. (Thus minimizing the chance that a mistake made w/ a development key will jeopardize our ability to scrape.)
- We encourage developers to get an API key of their own, but if necessary we can share our testing key in limited circumstances.

Currently only a few states require API keys:

- New York - http://legislation.nysenate.gov/static/docs/html/index.html

  - Request Form: http://legislation.nysenate.gov/

- Indiana - http://docs.api.iga.in.gov/api.html
  - API Key Request Process: Email Bob Amos (bob.amos@iga.in.gov or bamos@iga.in.gov), and include your name, address, phone, email address and company. Also indicate that you have read the terms of service at the link above.
- Oregon - https://www.oregonlegislature.gov/citizen_engagement/Pages/data.aspx
  - API Credentials Request Process: Email help.leg@oregonlegislature.gov and include your name, e-mail address, company or organization name, and contact phone number. Also please read over the API agreement and let them know you agree to the terms in the email.

If you're in need of an API key and unable to via these channels please contact a member of the core team to discuss getting access to the development key.

7.4 Testing Scrapers

One of the first things people new to the project tend to notice is that there aren't a lot of tests in the scrapers.

Over the years we've evolved a de facto policy of somewhat discouraging tests, which is definitely an unusual stance to take and warrants explanation.

7.4.1 Intentionally Fragile Scrapers

When it comes to scrapers, there are two major types of breakage:

1) the scraper collects bad information and inserts it into the database
2) the scraper encounters an error and quits without importing data

Given a choice, the second is greatly preferable. Once bad data makes it into the database, it can be difficult to detect and remove. On the other hand, the second can be triggered to alert us immediately and someone can evaluate the proper fix.

The best way to favor the second over first is to write "intentionally fragile" scrapers. That is, scrapers that raise an exception when they see unexpected input.

While it is possible to try to write a resilient scraper that recovers, by nature these scrapers are more likely to produce the first kind of error, and so we encourage scraper writers to be conservative in what errors are suppressed.

Here's an example of an overly permissive scraper:

```
party_abbr = doc.xpath('//span[@class="partyabbr"])
if party_abbr == 'D':
    party = 'Democratic'
elif party_abbr == 'R':
    party = 'Republican'
else:
    # haven't seen this yet, but let's just keep things moving
    party = party_abbr
```

The following would be preferred:

```
party_abbr = doc.xpath('//span[@class="partyabbr"])
party = {'D': 'Democratic', 'R': 'Republican'}[party_abbr]
```

This code would raise a `KeyError` the first time a new party is found. This forces someone to take a look, fix the scraper with an entry for the new party, and then the scraper will be able to run again with correct data.

7.4.2 Testing Scrapers Is Hard

On most software projects a failing test means that something is broken, and passing tests should mean that things are working just fine.

In our experience however, the majority of the "breaks" that occur in scrapers are due to upstream site changes.

In the past the fragile nature of scrapers has led to people writing a lot of bad tests, which is where our stance of somewhat discouraging tests has come from. An example of a bad test:

```
def extract_name(doc):
    return doc.xpath('//h2[@class="legislatorName"]').text_content().strip()


def test_extract_name():
    # probably a snapshot of the page at some point in time
    EXAMPLE_LEGISLATOR_HTML = '...'

    doc = lxml.html.fromstring(EXAMPLE_LEGISLATOR_HTML)
    assert extract_name(doc) == 'Erica Example'
```

With a test like this:

- As soon as the HTML changes, the scraper will start failing, but the tests will still pass.
- The scraper will then be updated, breaking the test.
- The test HTML will be updated, fixing the test.

But since the initial scraper breakage isn't predicted by a failing test, this type of test really doesn't serve us any purpose and just results in extra code to maintain every time the scraper needs a slight change.

7.4.3 Other Strategies

Of course this isn't to say that we just abandon the idea of testing, altogether.

If you're more comfortable writing tests, say you're parsing a particularly nasty PDF and want to run it against some test data: a test might make sense there as a way to be confident in your own code, by all means, write a test.

We also have some other strategies to help ensure data quality:

Validate Scraper Output

Scraper output is verified against JSON schemas that protect against common regressions (missing sources, invalid formatted districts, etc.) - most of these tests can be written effectively against scraper output across the board, and in doing so also applies universally across all 50 states.

We also aim for our underlying libraries like billy to be as well tested as possible. (To be 100% clear, our lax testing philosophy only applies to site-specific scraper code, not these support libraries.)

Run Scrapers Regularly

In a sense, the scrapers are tested every night by being run. This is why the intentionally fragile approach is so important; those failures are in essence the same as integration test failures. Of course, this doesn't tell us if the scraper is picking up bad data, etc., but combined with validation we can be fairly confident in our data.

Test Utilities

One area we can definitely improve upon is our use of (and then thorough testing of) common functions. Right now (largely because of the great variety of authors, etc.) many scrapers do similar things like conversion of party abbreviations and whitespace normalization in slightly different ways. We should be making a push to use common utility functions and thoroughly test those.

CHAPTER 8

Related Projects

billy Python backend for Open States' scrapers and website.

pyopenstates Python client library for Open States API.