legislice

Release 0.6.0

Matt Carey

Sep 20, 2021

CONTENTS

Release v. 0.6.0.

Legislice is a utility for downloading the text of statutes and constitutional provisions, and then creating computable objects representing passages from those provisions.

CHAPTER

ONE

GUIDES

To access interactive versions of these guides in Jupyter Notebooks, you can download the Legislice repository on GitHub.

1.1 Downloading Legislation

Legislice is a utility for downloading the text of statutes and constitutional provisions, and then creating computable objects representing passages from those provisions. This guide will show how to get started.

Legislice depends on the AuthoritySpoke API as its source of legislative text. Currently the API serves the text of the United States Constitution, plus versions of the United States Code in effect since 2013. Provisions of the United States Code that were repealed prior to 2013 aren't yet available through the API, and neither are any regulations or any state law.

1.1.1 Using an API token

To get started, make an account on authorityspoke.com. Then go to the User Profile page, where you can find your API token. The token is a 40-character string of random letters and numbers. You'll be sending this token to AuthoritySpoke to validate each API request, and you should keep it secret as you would a password.

There are several ways for Python to access your API token. One way would be to simply define it as a Python string, like this:

```
>>> TOKEN = "YOU_COULD_PUT_YOUR_API_TOKEN_HERE"
```

However, a better option is to make your API token an **environment variable**, and then use Python to access that variable. Using a Python library called dotenv, you can define an environment variable in a file called `.env` in the root of your project directory. For instance, the contents of the file `.env` could look like this:

```
LEGISLICE_API_TOKEN=YOUR_API_TOKEN_GOES_HERE
```

By doing this, you can avoid having a copy of your API token in your Python working file or notebook. That makes it easier to avoid accidentally publishing the API token or sharing it with unauthorized people.

Here's an example of loading an API token from a `.env` file using `dotenv`.

```
>>> import os
>>> from dotenv import load_dotenv
>>> load_dotenv()
True
>>> TOKEN = os.getenv("LEGISLICE_API_TOKEN")
```

Now you can use the API token to create a Legislice *Client* object. This object holds your API token, so you can reuse the *Client* without re-entering your API token repeatedly.

```
>>> from legislice.download import Client
>>> client = Client(api_token=TOKEN)
```

1.1.2 Fetching a provision from the API

To download legislation using the *Client*, we must specify a `path` to the provision we want, and optionally we can specify the `date` of the version of the provision we want. If we don't specify a date, we'll be given the most recent version of the provision.

The `path` citation format is based on the section identifiers in the United States Legislative Markup standard, which is a United States government standard used by the Office of the Law Revision Counsel for publishing the United States Code. Similar to a URL path in a web address, the `path` format is a series of labels connected with forward slashes. The first part identifies the jurisdiction, the second part (if any) identifies the legislative code within that jurisdiction, the third part identifies the next-level division of the code such as a numbered title, and so on.

If we don't know the right citation for the provision we want, we can sign in to an AuthoritySpoke account and browse the directory of available provisions, where the links to each provision show the correct `path` for that provision. Or we can browse an HTML version of the API itself. If the error message "Authentication credentials were not provided" appears, that means we aren't signed in, and we might want to go back to the login page.

The *fetch()* method makes an API call to AuthoritySpoke, and returns JSON that is been converted to a Python `dict`. There are fields representing the `content` of the provision, the `start_date` when the provision went into effect, and more.

Here's an example of how to fetch the text of the Fourth Amendment using the *Client*.

```
>>> data = client.fetch(query="/us/const/amendment/IV")
>>> data
{'heading': 'AMENDMENT IV.',
'start_date': '1791-12-15',
'node': '/us/const/amendment/IV',
'text_version': {
    'id': 735706,
    'url': 'https://authorityspoke.com/api/v1/textversions/735706/',
    'content': 'The right of the people to be secure in their persons, houses, papers,␣
↪and effects, against unreasonable searches and seizures, shall not be violated, and no␣
↪Warrants shall issue, but upon probable cause, supported by Oath or affirmation, and␣
↪particularly describing the place to be searched, and the persons or things to be␣
↪seized.'},
'url': 'https://authorityspoke.com/api/v1/us/const/amendment/IV/',
'end_date': None,
'children': [],
'citations': [],
'parent': 'https://authorityspoke.com/api/v1/us/const/amendment/'}
```

1.1.3 Loading an Enactment object

If all we needed was to get a JSON response from the API, we could have used a more general Python library like `requests`. Legislice also lets us load the JSON response as a *legislice.enactments.Enactment* object, which has methods for selecting some but not all of the provision's text. One way to load an *Enactment* is with the *Client*'s *read_from_json()* method.

```
>>> fourth_a = client.read_from_json(data)
>>> fourth_a.node
'/us/const/amendment/IV'
```

Instead of always using *fetch()* followed by *read_from_json()*, we can combine the two functions together with *read()*. In this example, we'll use *read()* to load a constitutional amendment that contains subsections, to show that the structure of the amendment is preserved in the resulting *Enactment* object.

```
>>> thirteenth_a = client.read(query="/us/const/amendment/XIII")
```

The string representation of this provision includes both the selected text (which is the full text of the amendment) as well as a citation to the provision with its effective date.

Currently the only supported citation format is the path-style citation used in United States Legislative Markup. Future versions of Legislice may support the ability to convert to traditional statute citation styles.

```
>>> str(thirteenth_a)
'/us/const/amendment/XIII (1865-12-18)'
```

The text of the Thirteenth Amendment is all within Section 1 and Section 2 of the amendment. You can use the `Enactment.children` property to get a list of provisions contained within an `Enactment`.

```
>>> len(thirteenth_a.children)
2
```

Then we can access each child provision as its own `Enactment` object from the `children` list. Remember that lists in Python start at index 0, so if we want Section 2, we'll find it at index 1 of the `children` list.

```
>>> str(thirteenth_a.children[1].text)
'Congress shall have power to enforce this article by appropriate legislation.'
```

1.1.4 Downloading prior versions of an Enactment

The API can be used to access specific provisions deeply nested within the United States Code, and also to access multiple date versions of the same provision. Here's a subsection of an appropriations statute as of 2015. We can use the `end_date` attribute to find when this version of the statute was displaced by a new version.

```
>>> old_grant_objective = client.read(query="/us/usc/t42/s300hh-31/a/1", date="2015-01-01
↪")
>>> old_grant_objective.content
'strengthening epidemiologic capacity to identify and monitor the occurrence of␣
↪infectious diseases and other conditions of public health importance;'
>>> old_grant_objective.end_date
datetime.date(2019, 7, 5)
```

And here's the same provision as of 2020. Its content has changed.

```
>>> new_grant_objective = client.read(query="/us/usc/t42/s300hh-31/a/1", date="2020-01-01
↪")
>>> new_grant_objective.content
'strengthening epidemiologic capacity to identify and monitor the occurrence of␣
↪infectious diseases, including mosquito and other vector-borne diseases, and other␣
↪conditions of public health importance;'
```

The 2020 version of the statute has `None` in its `end_date` field because it's still in effect.

```
>>> str(new_grant_objective.end_date)
'None'
```

1.1.5 Exploring the structure of a legislative code

When we query the API for a provision at a path with less than four parts (e.g., when we query for an entire Title of the United States Code), the response doesn't include the full text of the provision's children. Instead, it only contains URLs that link to the child nodes. These URL links might help to automate the process of navigating the API and discovering the provisions we want. Here's an example that discovers the URLs for the articles of the US Constitution.

```
>>> articles = client.read(query="/us/const/article")
>>> articles.children
['https://authorityspoke.com/api/v1/us/const/article/I/', 'https://authorityspoke.com/
↪api/v1/us/const/article/II/', 'https://authorityspoke.com/api/v1/us/const/article/III/
↪', 'https://authorityspoke.com/api/v1/us/const/article/IV/', 'https://authorityspoke.
↪com/api/v1/us/const/article/V/', 'https://authorityspoke.com/api/v1/us/const/article/
↪VI/', 'https://authorityspoke.com/api/v1/us/const/article/VII/']
```

1.1.6 Downloading Enactments from cross-references

If an *Enactment* loaded from the API references other provisions, it may provide a list of *CrossReference* objects when we call its *cross_references()* method. You can pass one of these *CrossReference* objects to the *fetch()* or *read()* method of the download client to get the referenced *Enactment*.

```
>>> infringement_provision = client.read("/us/usc/t17/s109/b/4")
>>> str(infringement_provision.text)
'Any person who distributes a phonorecord or a copy of a computer program (including any␣
↪tape, disk, or other medium embodying such program) in violation of paragraph (1) is␣
↪an infringer of copyright under section 501 of this title and is subject to the␣
↪remedies set forth in sections 502, 503, 504, and 505. Such violation shall not be a␣
↪criminal offense under section 506 or cause such person to be subject to the criminal␣
↪penalties set forth in section 2319 of title 18.'
>>> len(infringement_provision.cross_references())
2
>>> str(infringement_provision.cross_references()[0])
'CrossReference(target_uri="/us/usc/t17/s501", reference_text="section 501 of this title
↪")'
>>> reference_to_title_18 = infringement_provision.cross_references()[1]
>>> referenced_enactment = client.read(reference_to_title_18)
>>> referenced_enactment.text[:239]
'Any person who violates section 506(a) (relating to criminal offenses) of title 17␣
↪shall be punished as provided in subsections (b), (c), and (d) and such penalties␣
↪shall be in addition to any other provisions of title 17 or any other law.'
```

(continues on next page)

(continued from previous page)

An important caveat for this feature is that the return value of the *cross_references()* method will only be populated with internal links that have been marked up in the United States Legislative Markup XML published by the legislature. Unfortunately, some parts of the United States Code don't include any link markup when making references to other legislation.

1.1.7 Downloading Enactments from inbound citations

The method in the previous section finds and downloads Enactments cited by a known *Enactment*. But sometimes we want to discover provisions that cite *to* a particular provision. These "inbound" citations are not stored on the Python Enactment object. Instead, we have to go back to the download client and make an API request to get them, using the *citations_to()* method.

In this example, we'll get all the citations to the provision of the United States Code cited `/us/usc/t17/s501` (in other words, Title 17, Section 501). This gives us all known provisions that cite to that node in the document tree, regardless of whether different text has been enacted at that node at different times.

```
>>> inbound_refs = client.citations_to("/us/usc/t17/s501")
>>> str(inbound_refs[0])
'InboundReference to /us/usc/t17/s501, from (/us/usc/t17/s109/b/4 2013-07-18)'
```

We can examine one of these *InboundReference* objects to see the text creating the citation.

```
>>> inbound_refs[0].content
'Any person who distributes a phonorecord or a copy of a computer program (including any␣
↪tape, disk, or other medium embodying such program) in violation of paragraph (1) is␣
↪an infringer of copyright under section 501 of this title and is subject to the␣
↪remedies set forth in sections 502, 503, 504, and 505. Such violation shall not be a␣
↪criminal offense under section 506 or cause such person to be subject to the criminal␣
↪penalties set forth in section 2319 of title 18.'
```

But an *InboundReference* doesn't have all the same information as an *Enactment* object. Importantly, it doesn't have the text of any subsections nested inside the cited provision. We can use the download *Client* again to convert the InboundReference into an Enactment.

```
>>> citing_enactment = client.read(inbound_refs[0])
>>> citing_enactment.node
'/us/usc/t17/s109/b/4'
>>> citing_enactment.text
'Any person who distributes a phonorecord or a copy of a computer program (including any␣
↪tape, disk, or other medium embodying such program) in violation of paragraph (1) is␣
↪an infringer of copyright under section 501 of this title and is subject to the␣
↪remedies set forth in sections 502, 503, 504, and 505. Such violation shall not be a␣
↪criminal offense under section 506 or cause such person to be subject to the criminal␣
↪penalties set forth in section 2319 of title 18.'
```

This Enactment happens not to have any child nodes nested within it, so its full text is the same as what we saw when we looked at the InboundReference's content attribute.

```
>>> citing_enactment.children
[]
```

Sometimes, an *InboundReference* has more than one citation and start date. That means that the citing text has been enacted in different places at different times. This can happen because the provisions of a legislative code have been reorganized and renumbered. Here's an example. We'll look for citations to Section 1301 of USC Title 2, which is a section containing definitions of terms that will be used throughout the rest of Title 2.

```
>>> refs_to_definitions = client.citations_to("/us/usc/t2/s1301")
>>> str(refs_to_definitions[0])
'InboundReference to /us/usc/t2/s1301, from (/us/usc/t2/s4579/a/4/A 2018-05-09) and 2␣
↪other locations'
```

The *citations_to()* method returns a list, and two of the InboundReferences in this list have been enacted in three different locations.

```
>>> str(refs_to_definitions[0].locations[0])
'(/us/usc/t2/s60c-5/a/2/A 2013-07-18)'
```

When we pass an InboundReference to *read()*, the download client makes an *Enactment* from the most recent location where the citing provision has been enacted.

```
>>> str(client.read(refs_to_definitions[0]))
'/us/usc/t2/s4579/a/4/A (2018-05-09)'
```

If we need the *Enactment* representing the statutory text before it was moved and renumbered, we can pass one of the *CitingProvisionLocation* objects to the *Client* instead. Note that the Enactment we get this way has the same content text, but a different citation node, an earlier start date, and an earlier end date.

```
>>> citing_enactment_before_renumbering = client.read(refs_to_definitions[0].
↪locations[0])
>>> str(citing_enactment_before_renumbering)
'/us/usc/t2/s60c-5/a/2/A (2013-07-18)'
```

```
>>> citing_enactment_before_renumbering.end_date
datetime.date(2014, 1, 16)
```

1.2 Comparing Enactments

This notebook will show how `legislice` provides convenient functions for comparing passages of legislative text.

As explained in the *Downloading Legislation* guide, begin by creating a *Client* to download and create *Enactment* objects.

```
>>> import os
>>> from dotenv import load_dotenv
>>> from legislice.download import Client
>>> load_dotenv()
True
>>> TOKEN = os.getenv("LEGISLICE_API_TOKEN")
>>> client = Client(api_token=TOKEN)
```

1.2.1 Features of an Enactment

Now let's load the Fourteenth Amendment. We can get its effective date as a Python `datetime.date` object.

```
>>> fourteenth_amendment = client.read(query="/us/const/amendment/XIV")
>>> fourteenth_amendment.start_date
datetime.date(1868, 7, 28)
```

We can also view its `level`, which is "constitution", as opposed to "statute" or "regulation".

```
>>> fourteenth_amendment.level
<CodeLevel.CONSTITUTION: 1>
```

And we can find out the `node`, or place in the document tree, for the Fourteenth Amendment itself as well at its subsections.

```
>>> fourteenth_amendment.node
'/us/const/amendment/XIV'
```

```
>>> fourteenth_amendment.children[0].node
'/us/const/amendment/XIV/1'
```

We can also isolate some parts of the `node` path, such as the `jurisdiction` and `code` the amendment comes from.

```
>>> fourteenth_amendment.jurisdiction
'us'
```

```
>>> fourteenth_amendment.code
'const'
```

1.2.2 Selecting text

When we use the *`text()`* method, we get all the enacted text in the Fourteenth Amendment, including all its subsections.

```
>>> fourteenth_amendment.text
'All persons born or naturalized in the United States, and subject to the jurisdiction␣
↪thereof, are citizens of the United States and of the State wherein they reside. No␣
↪State shall make or enforce any law which shall abridge the privileges or immunities␣
↪of citizens of the United States; nor shall any State deprive any person of life,␣
↪liberty, or property, without due process of law; nor deny to any person within its␣
↪jurisdiction the equal protection of the laws. Representatives shall be apportioned␣
↪among the several States according to their respective numbers, counting the whole␣
↪number of persons in each State, excluding Indians not taxed. But when the right to␣
↪vote at any election for the choice of electors for President and Vice President of␣
↪the United States, Representatives in Congress, the Executive and Judicial officers of␣
↪a State, or the members of the Legislature thereof, is denied to any of the male␣
↪inhabitants of such State, being twenty-one years of age, and citizens of the United␣
↪States, or in any way abridged, except for participation in rebellion, or other crime,␣
↪the basis of representation therein shall be reduced in the proportion which the␣
↪number of such male citizens shall bear to the whole number of male citizens twenty-
↪one years of age in such State. No person shall be a Senator or Representative in␣
↪Congress, or elector of President and Vice President, or hold any office, civil or␣
↪military, under the United States, or under any State, who, having previously taken an␣
↪oath, as a member of Congress, or as an officer of the United States, or as a member␣
↪of any State legislature, or as an executive or judicial officer of any State, to␣
↪support the Constitution of the United States, shall have engaged in insurrection or␣
↪rebellion against the same, or given aid or comfort to the enemies thereof. But␣
↪Congress may by a vote of two-thirds of each House, remove such disability. The␣
↪validity of the public debt of the United States, authorized by law, including debts␣
↪incurred for payment of pensions and bounties for services in suppressing insurrection
```

(continues on next page)

(continued from previous page)

However, we might want an *Enactment* object that only represents a part of the Fourteenth Amendment that's relevant to a particular case. We can use the *select()* method to limit the text of the provision that's considered "selected". One way to do this is with a series of strings that exactly match the text we want to select. Because we're selecting only some of the text, the output of the *selected_text()* method will be different.

```
>>> passages = fourteenth_amendment.select(["No State shall", "deprive any person of",
↪"liberty", "without due process of law"])
>>> passages.selected_text()
'...No State shall...deprive any person of...liberty...without due process of law...'
```

The *select()* method returns a new *EnactmentPassage* object, which holds both the *Enactment* and a TextPositionSet indicating which text is selected. But if we want to select additional text without clearing the existing selection, we can use the EnactmentPassage's *select_more()* method. It's okay if the selection we pass in to *select_more()* overlaps with text we've already selected.

```
>>> passages.select_more("life, liberty, or property,")
>>> passages.selected_text()
'...No State shall...deprive any person of life, liberty, or property, without due␣
↪process of law...'
```

If we need to select a passage that occurs more than once in the Enactment, we can import the TextQuoteSelector class instead of using strings. With a TextQuoteSelector, we specify not just the `exact` phrase we want to select, but also a `prefix` or `suffix` that makes the phrase uniquely identifiable. In this example, the text being selected is the second instance of the phrase "twenty-one years of age" in the Fourteenth Amendment.

```
>>> from legislice import TextQuoteSelector
>>> age_passage = fourteenth_amendment.select(TextQuoteSelector(prefix="male citizens ",␣
↪exact="twenty-one years of age"))
>>> age_passage.selected_text()
'...twenty-one years of age...'
```

If we happen to know the start and end indices of the passage we want, then we can use a TextPositionSelector or TextPositionSet to select it, instead of specifying the exact text.

```
>>> from legislice.enactments import TextPositionSelector, TextPositionSet
>>> amendment_passage = fourteenth_amendment.
↪select(TextPositionSet(positions=[TextPositionSelector(start=1921, end=1973),␣
↪TextPositionSelector(start=2111, end=2135)]))
>>> amendment_passage.selected_text()
'...The validity of the public debt of the United States...shall not be questioned....'
```

We can also use the method *child_passages()* to get a new *EnactmentPassage* with only the subsection of the Fourteenth Amendment that interests us. The citation stored in the `node` attribute is now different, but the text selector still remains in place, so we can still get the same selected text.

```
>>> public_debt_provision = amendment_passage.child_passages[3]
>>> public_debt_provision.node
'/us/const/amendment/XIV/4'
>>> public_debt_provision.selected_text()
'The validity of the public debt of the United States...shall not be questioned....'
```

1.2.3 Comparing Selected Text

Next, we'll create a new *EnactmentPassage* to compare by changing the selected text of the original *Enactment* to include all the text that was selected before, plus more.

```
>>> debt_passage = fourteenth_amendment.select(TextPositionSelector(start=1921,␣
↪end=2135))
>>> debt_passage.selected_text()
'...The validity of the public debt of the United States, authorized by law, including␣
↪debts incurred for payment of pensions and bounties for services in suppressing␣
↪insurrection or rebellion, shall not be questioned....'
```

Now we can compare the text selections in these two *EnactmentPassage*s. The *implies()* method checks whether the Enactment on the left has all the text of the Enactment on the right. The *means()* method checks whether they both have the same text.

```
>>> fourteenth_amendment.implies(public_debt_provision)
True
```

We can also use Python's built-in "greater than or equal" operator as an alias for the *implies()* method.

```
>>> fourteenth_amendment >= public_debt_provision
True
```

Notice that Legislice is able to compare these two passages even though `amendment` is a text selection from the entire Fourteenth Amendment, while `public_debt_provision` is a text selection from only section 4 of the Fourteenth Amendment. We can verify this by checking the "node" attribute on each Enactment.

```
>>> fourteenth_amendment.node
'/us/const/amendment/XIV'
```

```
>>> public_debt_provision.node
'/us/const/amendment/XIV/4'
```

To determine whether two Enactments have the same text (and neither has any more than the other), use the *means()* method. Here's how we can check that the Fifth Amendment doesn't have identical text to the first section of the Fourteenth Amendment.

```
>>> fifth_amendment = client.read(query="/us/const/amendment/V")
>>> fifth_amendment.text
'No person shall be held to answer for a capital, or otherwise infamous crime, unless on␣
↪a presentment or indictment of a Grand Jury, except in cases arising in the land or␣
↪naval forces, or in the Militia, when in actual service in time of War or public␣
↪danger; nor shall any person be subject for the same offence to be twice put in␣
↪jeopardy of life or limb; nor shall be compelled in any Criminal Case to be a witness␣
↪against himself; nor be deprived of life, liberty, or property, without due process of␣
↪law; nor shall private property be taken for public use, without just compensation.'
```

```
>>> fourteenth_amendment_section_1 = client.read(query="/us/const/amendment/XIV/1")
>>> fifth_amendment.means(fourteenth_amendment_section_1)
False
```

However, the Fifth Amendment and the first section of the Fourteenth Amendment both happen to contain the phrase "life, liberty, or property, without due process of law". If we select that same passage from both provisions, then we can use the *means()* method to verify that both text selections are identical.

```
>>> phrase = "life, liberty, or property, without due process of law"
>>> due_process_14 = fourteenth_amendment_section_1.select(phrase)
>>> due_process_5 = fifth_amendment.select(phrase)
>>> due_process_14.means(due_process_5)
True
```

There are many situations in real legal analysis where it's helpful to know if identical text has been enacted at different citations. It could mean that the identical section has been renumbered, or it could mean that a judicial interpretation of one Enactment is also relevant to the other Enactment. Legislice's *implies()* and *means()* methods can help to automate that analysis.

Since >= is an alias for *implies()*, we might expect to be able to use == as an alias for *means()*. Currently we can't do that, because overriding the equals function could interfere with Python's ability to determine what objects are identical, and could cause bugs that would be difficult to diagnose. However, we can use > as an alias that returns `True` only if *implies()* would return `True` while *means()* would return `False`.

1.2.4 Combining Enactments

When we have two Enactments and either they are at the same node or one is a descendant of the other, we can combine them into a new Enactment using the addition sign. Here's an example from a copyright statute in the United States Code. The example shows that we can load section `/us/usc/t17/s103`, select some text from subsection `b` of that provision, and then add it to a separate Enactment representing the entirety of subsection `/us/usc/t17/s103/a`. Legislice combines the text from subsection `a` and subsection `b` in the correct order.

```
>>> s103 = client.read(query="/us/usc/t17/s103", date="2020-01-01")
>>> selections = ["The copyright in such work is independent of", "any copyright␣
↪protection in the preexisting material."]
>>> s103_passage = s103.select(selections)
>>> s103_passage.selected_text()
'...The copyright in such work is independent of...any copyright protection in the␣
↪preexisting material.'
```

```
>>> s103a = client.read(query="/us/usc/t17/s103/a", date="2020-01-01")
>>> s103a.text
'The subject matter of copyright as specified by section 102 includes compilations and␣
↪derivative works, but protection for a work employing preexisting material in which␣
↪copyright subsists does not extend to any part of the work in which such material has␣
↪been used unlawfully.'
```

```
>>> combined_passage = s103_passage + s103a
>>> combined_passage.selected_text()
'The subject matter of copyright as specified by section 102 includes compilations and␣
↪derivative works, but protection for a work employing preexisting material in which␣
↪copyright subsists does not extend to any part of the work in which such material has␣
↪been used unlawfully....The copyright in such work is independent of...any copyright␣
↪protection in the preexisting material.'
```

1.2.5 EnactmentGroups

When we want to work with groups of Enactments that may or may not be nested inside one another, we can put them together in an *EnactmentGroup*. When we create a new EnactmentGroup or *__add__()* two EnactmentGroups together, any overlapping *Enactment*s inside will be combined into a single Enactment.

In this example, we create two EnactmentGroups called `left` and `right`, each containing two Enactments, and add them together. Because one of the Enactments in `left` overlaps with one of the Enactments in `right`, when we add `left` and `right` together those two Enactments will be combined into one. Thus the resulting EnactmentGroup will contain three Enactments instead of four.

```
>>> from legislice import EnactmentGroup
>>> first = client.read(query="/us/const/amendment/I")
>>> establishment_clause=first.select("Congress shall make no law respecting an␣
↪establishment of religion")
>>> speech_clause = first.select(["Congress shall make no law", "abridging the freedom␣
↪of speech"])
>>> second = client.read(query="/us/const/amendment/II")
>>> arms_clause = second.select("the right of the people to keep and bear arms, shall␣
↪not be infringed.")
>>> third = client.read(query="/us/const/amendment/III")
>>> left = EnactmentGroup(passages=[establishment_clause, arms_clause])
>>> right = EnactmentGroup(passages=[third, speech_clause])
>>> combined = left + right
>>> print(combined)
the group of Enactments:
  "Congress shall make no law respecting an establishment of religion...abridging the␣
↪freedom of speech..." (/us/const/amendment/I 1791-12-15)
  "...the right of the people to keep and bear arms, shall not be infringed." (/us/const/
↪amendment/II 1791-12-15)
  "No soldier shall, in time of peace be quartered in any house, without the consent of␣
↪the Owner, nor in time of war, but in a manner to be prescribed by law." (/us/const/
↪amendment/III 1791-12-15)
>>> len(combined)
3
```

1.2.6 Converting Enactments to JSON

When we want a representation of a legislative passage that's precise, machine-readable, and easy to share over the internet, we can use Legislice's JSON schema. Here's how to convert the Enactment object called `combined_enactment`, which was created in the example above, to JSON.

As explained in the section above, this JSON represents a selection of three nonconsecutive passages from the most recent version of section 103 of Title 17 of the United States Code. The schema's `json()` method returns a JSON string, while the `dict()` method returns a Python dictionary.

```
>>> combined.passages[0].json()
'{"enactment": {"node": "/us/const/amendment/I", "start_date": "1791-12-15", "heading":
↪"AMENDMENT I.", "text_version": {"content": "Congress shall make no law respecting an␣
↪establishment of religion, or prohibiting the free exercise thereof; or abridging the␣
↪freedom of speech, or of the press; or the right of the people peaceably to assemble,␣
↪and to petition the Government for a redress of grievances.", "url": "https://
↪authorityspoke.com/api/v1/textversions/735703/", "id": 735703}, "end_date": null,
↪"first_published": "1788-06-21", "earliest_in_db": "1788-06-21", "anchors": [],
↪"citations": [], "name": "", "children": []}, "selection": {"positions": [{"start": 0,
↪"end": 66}, {"start": 113, "end": 144}], "quotes": []}}'
```

(continues on next page)

(continued from previous page)

1.2.7 Formatting Citations (Experimental)

Legislice has preliminary support for serializing citations for Enactment objects based on Citation Style Language JSON. The goal of this feature is to support compatibility with Jurism. Please open an issue in the Legislice repo if you have suggestions for how this feature should develop to support your use case.

```
>>> cares_act_benefits = client.read("/us/usc/t15/s9021/")
>>> cares_act_benefits.heading
'Pandemic unemployment assistance'
>>> citation = cares_act_benefits.as_citation()
>>> str(citation)
'15 U.S. Code § 9021 (2020)'
>>> cares_act_benefits.csl_json()
'{"jurisdiction": "us", "code_level_name": "CodeLevel.STATUTE", "volume": "15", "section
↪": "sec. 9021", "type": "legislation", "container-title": "U.S. Code", "event-date": {
↪"date-parts": [["2020", 4, 10]]}}'
```

This CSL-JSON format currently only identifies the cited provision down to the section level. Calling the *as_citation()* method on a subsection or deeper nested provision will produce the same citation data as its parent section.

CHAPTER

TWO

DEVELOPMENT UPDATES

2.1 Reporting Bugs and Issues

Your feedback is very valuable. If you discover a bug, if Legislice isn't behaving as expected, or if you want to suggest a new feature, please comment or open an issue on Legislice's GitHub issue tracker. For other comments please use the Twitter contact information below.

If you want to submit a pull request for either Legislice or AuthoritySpoke, please also submit the AuthoritySpoke contributor license agreement.

2.2 Current Updates

2.2.1 GitHub

You can find open issues and current changes to Legislice through its GitHub repo.

2.2.2 Twitter

On Twitter, you can follow @authorityspoke or @mcareyaus for project updates.

2.2.3 Changelog

0.6.0 (2021-09-20)

- add EnactmentPassage class
- select_from_text_positions_without_nesting doesn't accept RangeSet
- Enactment.limit_selection.start must be an int
- no separate LinkedEnactment class for Enactments with URL links as children
- remove BaseEnactment parent class
- replace Marshmallow schemas with Pydantic models

0.5.2 (2021-05-20)

- sort EnactmentGroups by level
- add California to KNOWN_CODES

0.5.1 (2021-05-08)

- separate schemas for YAML and JSON input
- flag determines if read_from_json uses text expansion
- change InboundReference to dataclass

0.5.0 (2021-03-26)

- add EnactmentGroup class
- drop Python 3.7 support
- import Citation and Client at top level of library
- Client.fetch_cross_reference no longer will ignore "date" param
- EnactmentGroup init method can accept None as "enactments" param
- remove "text expansion" module and functions
- remove ExpandableSchema class

0.4.1 (2020-12-31)

- fix bug: Client made API request requiring 301 redirect

0.4.0 (2020-12-29)

- add Citation class
- add Citation Style Language JSON serializer methods
- remove mock Clients by migrating tests to pytest-vcr

0.3.1 (2020-12-12)

- order fields in serialized Enactment JSON output format for readability
- remove include_start and include_end from serialized Enactment JSON output
- fix bug: Enactment.select_all created zero length selectors

0.3.0 (2020-11-17)

- add CrossReference class as memo of cited Enactment to download
- add CitingProvisionLocation as memo of citing Enactment to download
- add cross_references attr to Enactment model
- add citations_to method to Client class
- EnactmentSchema's content field is moved to a new nested model called TextVersionSchema
- add ability to pass CitingProvisionLocation to Client.read
- add ability to pass InboundReference to Client.read

0.2.0 (2020-08-30)

- don't add ellipsis to selected_text for node with no text
- accept list of strings to generate anchorpoint TextPositionSet
- combine selected text passages within 3 characters of each other

0.1.1 (2020-08-23)

- initial release

CHAPTER

THREE

API DOCUMENTATION

3.1 Download Client

class legislice.download.**Client**(*api_token=''*, *api_root='https://authorityspoke.com/api/v1'*, *update_coverage_from_api=True*)

Downloader for legislative text.

citations_to(*target*)

Load an InboundReference object for each provision citing the target USLM provision URI.

Parameters **target** (Union[str, *Enactment*, *CrossReference*]) – a string URI for the cited node, an Enactment at the cited node, or a CrossReference to the cited node

Return type List[*InboundReference*]

Returns a list of InboundReferences to the cited node

fetch(*query*, *date=''*)

Download legislative provision from string identifier or cross-reference.

Parameters

- **query** (Union[str, *CitingProvisionLocation*, *CrossReference*, *InboundReference*]) – A cross-reference to the desired legislative provision, or a path to the desired legislation section using the United States Legislation Markup tree-like citation format.
- **date** (Union[date, str]) – The date of the desired version of the provision to be downloaded. This is not needed if a *CrossReference* passed to the `query` param specifies a date. If no date is provided, the API will use the most recent date.

Return type Dict[str, Union[Any, str, List[Union[str, Dict[str, str]]]]]

fetch_citations_to(*target*)

Query API for citations to a given target node, withoout loading them as InboundCitations.

Parameters **target** (Union[str, *Enactment*, *CrossReference*]) – a string URI for the cited node, an Enactment at the cited node, or a CrossReference to the cited node

Return type List[Dict]

Returns a list of dicts representing citations to the cited node

fetch_citing_provision(*query*)

Download legislative provision as Enactment from CitingProvisionLocation.

CitingProvisionLocations are found in the *locations* attribute of the *InboundReference* objects obtained when using the *citations_to* method.

Return type Dict[str, Union[Any, str, List[Union[str, Dict[str, str]]]]]

fetch_cross_reference(*query*, *date=''*)
Download legislative provision from cross-reference.

Parameters

- **query** (*CrossReference*) – A cross-reference to the desired legislative provision. Found by calling the *cross_references()* method on an *Enactment* that contains one or more citations to other provisions.
- **date** (Union[date, str]) – The date of the desired version of the provision to be downloaded. This is not needed if the *CrossReference* passed to the query param specifies a date. If no date is provided, the API will use the most recent date.

Return type Dict[str, Union[Any, str, List[Union[str, Dict[str, str]]]]]

fetch_db_coverage(*code_uri*)
Document date range of provisions of a code of laws available in API database.

Return type Dict[str, Union[str, date]]

fetch_inbound_reference(*query*)
Download legislative provision from InboundReference.

Parameters **query** (*InboundReference*) – An InboundReference identifying a provision containing a citation.

Return type Dict[str, Union[Any, str, List[Union[str, Dict[str, str]]]]]

Returns An Enactment representing the provision containing the citation (not the cited provision).

If the InboundReference has been enacted more than once, the latest one will be chosen.

fetch_uri(*query*, *date=''*)
Fetch data about legislation at specified path and date from Client's assigned API root.

Parameters

- **query** (str) – A path to the desired legislation section using the United States Legislation Markup tree-like citation format. Examples: "/us/const/amendment/IV", "/us/usc/t17/s103"
- **date** (Union[date, str]) – A date when the desired version of the legislation was in effect. This does not need to be the "effective date" or the first date when the version was in effect. However, if you select a date when two versions of the provision were in effect at the same time, you will be given the version that became effective later.

Return type Dict[str, Union[Any, str, List[Union[str, Dict[str, str]]]]]

get_db_coverage(*uri*)
Add data about the API's coverage date range to the Enactment to be loaded.

As a side effect, changes the Client's "coverage" attribute.

Parameters **uri** (str) – identifier for the Enactment to be created

Return type str

Returns identifier for the Code of the Enactment to be created

read(*query*, *date=''*)
Fetch data from Client's assigned API root and builds Enactment or LinkedEnactment.

All text is selected by default.

Parameters

- **path** – A path to the desired legislation section using the United States Legislation Markup tree-like citation format. Examples: "/us/const/amendment/IV", "/us/usc/t17/s103"
- **date** (Union[date, str]) – A date when the desired version of the legislation was in effect. This does not need to be the "effective date" or the first date when the version was in effect. However, if you select a date when two versions of the provision were in effect at the same time, you will be given the version that became effective later.

Return type *Enactment*

read_from_json(*data*, *use_text_expansion=True*)
Create a new Enactment object using imported JSON data.

If fields are missing from the JSON, they will be fetched using the API key.

Return type *Enactment*

read_passage_from_json(*data*)
Create a new EnactmentPassage object using imported JSON data.

If fields are missing from the JSON, they will be fetched using the API key.

Return type *EnactmentPassage*

update_data_from_api_if_needed(*data*)
Update data from API with data from API coverage.

Parameters **data** (Dict[str, Union[Any, str, List[Union[str, Dict[str, str]]]]]) – a dict representing the data from the API

Return type Dict[str, Union[Any, str, List[Union[str, Dict[str, str]]]]]

Returns a dict representing the data from the API with updated data from API coverage

update_enactment_from_api(*data*)
Use API to fill in missing fields in a dict representing an *Enactment*.

Useful when the dict has missing data because it was created by a user.

Return type Dict[str, Union[Any, str, List[Union[str, Dict[str, str]]]]]

update_entries_in_enactment_index(*enactment_index*)
Fill in missing fields in every entry in an EnactmentIndex.

Return type Mapping[str, Dict[str, Union[Any, str, List[Union[str, Dict[str, str]]]]]]

uri_from_query(*target*)
Get a URI for the target object.

Return type str

url_from_enactment_path(*path*, *date=''*)
Generate URL for API call for specified USLM path and date.

Return type str

legislice.download.**normalize_path**(*path*)
Make sure path starts but does not end with a slash.

Return type str

3.2 Enactments

class legislice.enactments.**Enactment**(***data*)

Base class for Enactments.

Whether connected to subnodes by linking, or nesting.

Parameters

- **node** – identifier for the site where the provision is codified
- **heading** – full heading of the provision
- **text_version** – full text content at this node, even if not all of it is cited
- **start_date** – date when the text was enacted at the cited location
- **known_revision_date** – whether the "start_date" field is known to be a date when the provision was revised in the code where it was publised. If False, then the Enactment refers to a time range when the text was codified, without asserting that the Enactment was not codified at earlier dates. This field is useful when working from incomplete legislative records.
- **end_date** – date when the text was removed from the cited location
- **first_published** – date when this Enactment's code was first published.
- **earliest_in_db** – date of the earliest version of this Enactment in the database. Used to determine whether the start_date of the Enactment is a date when the Enactment was amended or first published.
- **anchors** – a list of selectors representing the part of some other document (e.g. an Opinion) that references the Enactment. Unlike the selection field, it doesn't reference part of the Enactment itself. For use as a temporary place to store the anchors until they can be moved over to the citing document.
- **name** – a user-assigned label for the object

as_citation()

Create Citation Style Language markup for the Enactment.

Return type *Citation*

property code

Get "code" part of node identifier.

property content: str

Get text for this version of the Enactment.

Return type str

convert_quotes_to_position(*quotes*)

Convert quote selector to the corresponding position selector for this Enactment.

Return type TextPositionSet

convert_selection_to_set(*selection*)

Create a TextPositionSet from a different selection method.

Return type TextPositionSet

cross_references()

Return all cross-references from this node and subnodes.

Return type List[*CrossReference*]

csl_json()
Serialize a citation to this provision in Citation Style Language JSON.

Experimental feature. This CSL-JSON format currently only identifies the cited provision down to the section level. A citation to a subsection or deeper nested provision will be the same as a citation to its parent section.

See https://citeproc-js.readthedocs.io/en/latest/csl-json/markup.html for a guide to this CSL-JSON format.

Return type str

get_identifier_part(*index*)
Get a part of the split node identifier, by number.

Return type Optional[str]

get_string(*selection*)
Use text selector to get corresponding string from Enactment.

Return type str

implies(*other*)
Test whether `self` has all the text passages of `other`.

Return type bool

property is_federal
Check if self is from a federal jurisdiction.

property jurisdiction
Get "sovereign" part of node identifier.

property known_revision_date: bool
Determine if Enactment's start_date reflects its last revision date.

If not, then the *start_date* merely reflects the earliest date that versions of the *Enactment*'s code exist in the database.

Return type bool

property level: str
Get level of code for this Enactment, e.g. "statute" or "regulation".

Return type str

limit_selection(*selection*, *start=0*, *end=None*)
Limit selection to the range defined by start and end points.

Return type TextPositionSet

limit_selection_to_current_node(*selection*)
Limit selection to the current node.

Return type TextPositionSet

make_selection(*selection=True*, *start=0*, *end=None*)
Make a TextPositionSet for specified text in this Enactment.

Return type TextPositionSet

make_selection_of_all_text()
Return a TextPositionSet of all text in this Enactment.

Return type TextPositionSet

make_selection_of_this_node()
Return a TextPositionSet of the text at this node, not child nodes.

Return type TextPositionSet

classmethod make_text_version_from_str(*value*)
Allow content to be used to populate text_version.

Return type Optional[TextVersion]

means(*other*)
Determine if self and other have identical text.

Return type bool

property nested_children
Get nested children attribute.

property padded_length
Get length of self's content plus one character for space before next section.

raise_error_for_extra_selector(*selection*)
Raise an error if any passed selectors begin after the end of the text passage.

Return type None

rangedict()
Return a RangeDict matching text spans to Enactment attributes.

Return type RangeDict

property section
Get "section" part of node identifier.

select(*selection=True*, *start=0*, *end=None*)
Select text from Enactment.

Return type *EnactmentPassage*

select_all()
Return a passage for this Enactment, including all subnodes.

Return type *EnactmentPassage*

property sovereign
Get "sovereign" part of node identifier.

property span_length: int
Return the length of the span of this Enactment.

Return type int

property text
Get all text including subnodes, regardless of which text is "selected".

text_sequence(*include_nones=True*)
Get a sequence of text passages for this provision and its subnodes.

Return type TextSequence

property title
Get "title" part of node identifier.

tree_selection()
Return set of selectors for selected text in this provision and its children.

Return type TextPositionSet

class legislice.enactments.**EnactmentPassage**(***data*)

An Enactment with selectors indicating which text is being referenced.

as_quotes()

Return quote selectors for the selected text.

Return type List[TextQuoteSelector]

property child_passages: List[*legislice.enactments.EnactmentPassage*]

Return a list of EnactmentPassages for this Enactment's children.

Return type List[*EnactmentPassage*]

clear_selection()

Deselect any Enactment text, including in child nodes.

Return type None

property code

Get "code" part of node identifier.

implies(*other*)

Test whether `self` has all the text passages of `other`.

Return type bool

property is_federal

Check if self is from a federal jurisdiction.

property jurisdiction

Get "sovereign" part of node identifier.

property level: str

Get level of code for this Enactment, e.g. "statute" or "regulation".

Return type str

limit_selection(*start=0*, *end=None*)

Limit selection to the range defined by start and end points.

Return type None

means(*other*)

Find whether meaning of `self` is equivalent to that of `other`.

`Self` must be neither broader nor narrower than `other` to return True.

Return type bool

Returns whether `self` and `other` represent the same text issued by the same sovereign in the same level of *Enactment*.

property node

Get the node that this Enactment is from.

property section

Get "section" part of node identifier.

select(*selection=True*, *start=0*, *end=None*)

Select text from Enactment.

Return type None

select_all()
Select all text of Enactment.

Return type None

select_more(*selection*)
Select text, in addition to any previous selection.

Return type None

select_more_text_at_current_node(*added_selection*)
Select more text at this Enactment's node, not in child nodes.

Return type None

select_more_text_from_changed_version(*other*)
Select more text from a different text version at the same citation path.

Parameters **other** (*EnactmentPassage*) – An *Enactment* representing different text enacted at a different time, at the same *node* (or USLM path citation) as self. This Element's node attribute must be the same string as self's node attribute. It's not sufficient for *other* to have an Enactment listed in its _children attribute with the same node attribute, or for *other* to have the same node attribute as an ancestor of self.

Return type None

select_more_text_in_current_branch(*added_selection*)
Select more text within this Enactment's tree_selection, including child nodes.

Return type None

selected_text()
Return this provision's text that is within the ranges described by self.selection.

Based on creating an anchorpoint.textsequences.TextSequence from this Enactment's text content and the ranges in its selection attribute.

Return type str

property sovereign
Get "sovereign" part of node identifier.

property text
Get all text including subnodes, regardless of which text is "selected".

text_sequence(*include_nones=True*)
List the phrases in the Enactment selected by TextPositionSelectors.

Parameters **include_nones** (bool) – Whether the list of phrases should include *None* to indicate a block of unselected text

Return type TextSequence

property title
Get "title" part of node identifier.

class legislice.enactments.**CrossReference**(***data*)
A legislative provision's citation to another provision.

Parameters

- **target_uri** – the path to the target provision from the document root.
- **target_url** – the URL to fetch the target provision from an API.

- **reference_text** – The text in the citing provision that represents the cross-reference. Generally, this text identifies the target provision.
- **target_node** – an identifier for the target URI in the API.

class legislice.enactments.**InboundReference**(***data*)
A memo that a TextVersion has a citation to a specified target provision.

latest_location()
Get most recent location where the citing text has been enacted.

Return type *CitingProvisionLocation*

classmethod search_citations_for_reference_text(*values*)
Get reference_text field from nested "citations" model.

Return type Dict

class legislice.enactments.**CitingProvisionLocation**(***data*)
Memo indicating where an Enactment can be downloaded.

Parameters

- **node** – location of the citing provision in a legislative code
- **start_date** – start date of the citing version of the provision
- **heading** – heading text for the citing provision

legislice.enactments.**consolidate_enactments**(*enactments*)
Consolidate any overlapping *Enactment*s in a list.

Parameters **enactments** (Sequence[Union[*Enactment*, *EnactmentPassage*]]) – a list of *Enactment*s that may include overlapping legislative text within the same section

Return type List[*EnactmentPassage*]

Returns a list of *Enactment*s without overlapping text

3.3 Citations

class legislice.citations.**CodeLevel**(*value*)
An enumeration.

class legislice.citations.**Citation**(***data*)
A citation style for referring to an *Enactment* in written text.

Intended for use with Citation Style Language (CSL).

static csl_date_format(*revision_date*)
Convert event date to Citation Style Language format.

Return type Dict[str, List[List[Union[str, int]]]]

csl_dict()
Return citation as a Citation Style Language object.

Return type Dict[str, Union[str, int, List[List[Union[str, int]]]]]

csl_json()
Return citation as Citation Style Language JSON.

Return type str

legislice.citations.**identify_code**(*jurisdiction*, *code*)
Find code name and type based on USLM citation parts.

Return type Tuple[str, str]

3.4 Enactment Groups

class legislice.groups.**EnactmentGroup**(***data*)
Group of Enactments with comparison methods.

__add__(*other*)
Combine two EnactmentGroups, consolidating any duplicate Enactments.

Return type *EnactmentGroup*

__ge__(*other*)
Test whether `self` implies `other` and `self` != `other`.

Return type bool

__gt__(*other*)
Test whether `self` implies `other` and `self` != `other`.

Return type bool

__hash__ = None

__iter__()
so *dict(model)* works

__repr__()
Return repr(self).

Return type str

__str__()
Return str(self).

implies(*other*)
Determine whether self includes all the text of another Enactment or EnactmentGroup.

Return type bool

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

A

C

E

F

G

I

J

K

L

M

N

P

R

S

T

U