Sefaria Documentation

Release 1.2

The Sefaria Team

Oct 09, 2017

Contents

- genindex
- search

CHAPTER 1

Library

class sefaria.model.text.**Library**

Operates as a singleton, through the instance called `library`.

Stewards the in-memory and in-cache objects that cover the entire collection of texts.

Exposes methods to add, remove, or register change of an index record. These are primarily called by the dependencies mechanism on Index Create/Update/Destroy.

add_index_record_to_cache(*index_object=None*, *rebuild=True*)

Update library title dictionaries and caches with information from provided index. Index can be passed with primary title in *index_title* or as an object in *index_object* :param index_object: Index record :param rebuild: Perform a rebuild of derivative objects afterwards? False only in cases of batch update. :return:

all_index_records()

all_titles_regex(*lang='en'*, *with_terms=False*, *citing_only=False*)

Returns A regular expression object that will match any known title in the library in the provided language

Parameters

- **lang** – "en" or "he"
- **with_terms** (*bool*) – Default False. If True, include shared titles ('terms'). (Will have no effect if citing_only is True)
- **citing_only** – Match only those texts which have is_cited set to True

Raise InputError: if lang == "he" and commentary == True

Uses re2 if available. See https://github.com/Sefaria/Sefaria-Project/wiki/Regular-Expression-Engines

all_titles_regex_string(*lang='en'*, *with_terms=False*, *citing_only=False*)

Parameters

- **lang** – "en" or "he"
- **with_terms** – Include terms in regex. (Will have no effect if citing_only is True)

- **citing_only** – Match only those texts which have is_cited set to True
- **for_js** –

Returns

build_full_auto_completer()

build_ref_auto_completer()

category_id_dict(*toc=None*, *cat_head=''*, *code_head=''*)

citing_title_list(*lang='en'*)

Parameters **lang** – "he" or "en"

Returns list of all titles that can be recognized as an inline citation

delete_index_from_toc(*indx*)

full_auto_completer(*lang*)

full_title_list(*lang='en'*, *with_terms=False*)

Returns list of strings of all possible titles

Parameters

- **lang** – "he" or "en"
- **with_terms** – if True, includes shared titles ('terms')

get_content_nodes()

Returns list of all content nodes in the library

get_dependant_indices(*book_title=None*, *dependence_type=None*, *structure_match=False*, *full_records=False*)

Replacement for all get commentary title methods :param book_title: Title of the base text. If book_title is None, returns all matching dependent texts :param dependence_type: none, "Commentary" or "Targum" - generally used to get Commentary and leave out Targum. If none, returns all indexes. :param structure_match: If True, returns records that follow the base text structure :param full_records: If True, returns an IndexSet, if False returns list of titles. :return: IndexSet or List of titles.

get_index(*bookname*)

Factory - returns a *Index* object that has the given bookname

Parameters **bookname** (*string*) – Name of the book.

Returns

get_index_forest()

Returns list of root Index nodes.

get_indexes_in_category(*category*, *include_dependant=False*, *full_records=False*)

Parameters

- **category** (*string*) – Name of category
- **include_dependant** (*bool*) – If true includes records of Commentary and Targum
- **full_records** (*bool*) – If True will return the actual :class: 'IndexSet' otherwise just the titles

Returns *IndexSet* of *Index* records in the specified category

get_indices_by_collective_title(*collective_title*, *full_records=False*)

get_multi_title_regex_string(*titles*, *lang*, *for_js=False*, *anchored=False*)
Capture title has to be true. :param titles: :param lang: :param for_js: :param anchored: :return:

get_refs_in_string(*st*, *lang=None*, *citing_only=False*)
Returns an list of Ref objects derived from string

Parameters

- **st** (*string*) – the input string
- **lang** – "he" or "en"
- **citing_only** – boolean whether to use only records explicitly marked as being referenced in text.

Returns list of *Ref* objects

get_regex_string(*title*, *lang*, *for_js=False*, *anchored=False*, *capture_title=False*)

get_schema_node(*title*, *lang=None*)

Parameters

- **title** (*string*) –
- **lang** – "en" or "he"

Returns a particular SchemaNode that matches the provided title and language

Return type sefaria.model.schema.SchemaNode

get_search_filter_toc()
Returns table of contents object from cache, DB or by generating it, as needed.

get_search_filter_toc_json()
Returns JSON representation of TOC.

get_simple_term_mapping()

get_term_dict(*lang='en'*)

Returns dict of shared titles that have an explicit ref

Parameters **lang** – "he" or "en"

get_text_categories()

Returns List of all known text categories.

get_text_titles_json(*lang='en'*)

Returns JSON of full texts list, (cached)

get_title_node_dict(*lang='en'*)

Parameters **lang** – "he" or "en"

Returns dictionary of string titles and the nodes that they point to.

Does not include bare commentator names, like *Rashi*.

get_titles_in_string(*s*, *lang=None*, *citing_only=False*)
Returns the titles found in the string.

Parameters

- **s** – The string to search
- **lang** – "en" or "he"

Return list titles found in the string

get_toc()
Returns table of contents object from cache, DB or by generating it, as needed.

get_toc_json()
Returns JSON representation of TOC.

get_toc_tree()

get_wrapped_refs_string(*st*, *lang=None*, *citing_only=False*)
Returns a string with the list of Ref objects derived from string wrapped in <a> tags

Parameters

- **st** (*string*) – the input string
- **lang** – "he" or "en"
- **citing_only** – boolean whether to use only records explicitly marked as being referenced in text

Returns string:

rebuild(*include_toc=False*)

rebuild_toc()

recount_index_in_toc(*indx*)

ref_auto_completer(*lang*)

ref_list()

Returns list of all section-level Refs in the library

refresh_index_record_in_cache(*index_object*, *old_title=None*)
Update library title dictionaries and caches for provided index :param title: primary title of index :return:

remove_index_record_from_cache(*index_object=None*, *old_title=None*, *rebuild=True*)
Update provided index from library title dictionaries and caches :param index_object: :param old_title: In the case of a title change - the old title of the Index record :param rebuild: Perform a rebuild of derivative objects afterwards? :return:

reset_simple_term_mapping()

update_index_in_toc(*indx*, *old_ref=None*)

Parameters

- **indx** –
- **old_ref** –

Returns

CHAPTER 2

Ref

class sefaria.model.text.**Ref**(*tref=None*, *_obj=None*)

A Ref is a reference to a location. A location could be to a *book*, to a specific *segment* (e.g. verse or mishnah), to a *section* (e.g chapter), or to a *range*.

Instanciated with a string representation of the reference, e.g.:

```
>>> Ref("Genesis 1:3")
>>> Ref("Rashi on Genesis 1:3")
>>> Ref("Genesis 1:3-2:4")
>>> Ref("Shabbat 4b")
>>> Ref("Rashi on Shabbat 4b-5a")
```

Displaying Refs

Ref.**normal**()

Return string Normal English string form

Ref.**he_normal**()

Return string Normal Hebrew string form

Ref.**url**()

Return string normal url form

Inspecting Refs

static Ref.**is_ref**(*tref*)

Static method for testing if a string is valid for instanciating a Ref object.

Parameters **tref** (*string*) – the string to test

Return bool

Ref.**is_commentary**()
Is this a commentary reference?

Return bool

Ref.**is_talmud**()
Is this a Talmud reference?

Return bool

Ref.**is_bavli**()
Is this a Talmud Bavli or related text reference? :return bool:

Ref.**is_range**()
Is this reference a range?

A Ref is range if it's starting point and ending point are different, i.e. it has a dash in its text form. References can cover large areas of text without being a range - in the case where they are references to chapters.

```
>>> Ref("Genesis 3").is_range()
False
>>> Ref("Genesis 3-5").is_range()
True
```

Return bool

Ref.**is_spanning**()

Return bool True if the Ref spans across text sections.

```
>>> Ref("Shabbat 13a-b").is_spanning()
True
>>> Ref("Shabbat 13a:3-14").is_spanning()
False
>>> Ref("Job 4:3-5:3").is_spanning()
True
>>> Ref("Job 4:5-18").is_spanning()
False
```

Ref.**is_section_level**()
Is this Ref section (e.g. Chapter) level?

```
>>> Ref("Leviticus 15:3").is_section_level()
False
>>> Ref("Leviticus 15").is_section_level()
True
>>> Ref("Rashi on Leviticus 15:3").is_section_level()
True
>>> Ref("Rashi on Leviticus 15:3:1").is_section_level()
False
>>> Ref("Leviticus 15-17").is_section_level()
True
```

Return bool

Ref.**is_segment_level**()
Is this Ref segment (e.g. Verse) level?

```
>>> Ref("Leviticus 15:3").is_segment_level()
True
>>> Ref("Leviticus 15").is_segment_level()
False
>>> Ref("Rashi on Leviticus 15:3").is_segment_level()
False
>>> Ref("Rashi on Leviticus 15:3:1").is_segment_level()
True
```

Return bool

Ref.**range_depth**()
How deep is the range?

```
>>> Ref("Leviticus 15:3 - 17:12").range_depth()
2
>>> Ref("Leviticus 15-17").range_depth()
2
>>> Ref("Leviticus 15:17-21").range_depth()
1
>>> Ref("Leviticus 15:17").range_depth()
0
```

Return int

Ref.**range_index**()
At what section index does the range begin?

```
>>> Ref("Leviticus 15:3 - 17:12").range_index()
0
>>> Ref("Leviticus 15-17").range_index()
0
>>> Ref("Leviticus 15:17-21").range_index()
1
>>> Ref("Leviticus 15:17").range_index()
2
```

Return int

Ref.**range_list**()

Returns list of *Ref* objects corresponding to each point in the range of this *Ref*

Ref.**range_size**()
How large is the range?

Return int

Ref.**span_size**()
How many sections does the span cover?

```
>>> Ref("Leviticus 15:3 - 17:12").span_size()
3
>>> Ref("Leviticus 15-17").span_size()
```

```
3
>>> Ref("Leviticus 15:17-21").span_size()
1
>>> Ref("Leviticus 15:17").span_size()
1
```

Return int

Comparing Refs

`Ref.`**`contains`**(*other*)
Does this Ref completely contain `other` Ref?

Parameters **`other`** –

Return bool

`Ref.`**`follows`**(*other*)
Does this Ref completely follow `other` Ref?

Parameters **`other`** –

Return bool

`Ref.`**`precedes`**(*other*)
Does this Ref completely precede `other` Ref?

Parameters **`other`** –

Return bool

`Ref.`**`overlaps`**(*other*)
Does this Ref overlap `other` Ref?

Parameters **`other`** –

Return bool

`Ref.`**`in_terms_of`**(*other*)
Returns the current reference sections in terms of another, containing reference.

Returns an array of ordinal references, not array indexes. (Meaning first is 1)

Must be called on a point Reference, not a range

“”

```
>>> Ref("Genesis 6:3").in_terms_of("Genesis 6")
[3]
>>> Ref("Genesis 6:3").in_terms_of("Genesis")
[6,3]
>>> Ref("Genesis 6:3").in_terms_of("Genesis 6-7")
[1,3]
>>> Ref("Genesis 6:8").in_terms_of("Genesis 6:3-7:3")
[1, 6]
```

Parameters **`other`** – *Ref*

Returns array of indexes

Ref.**__eq__**(*other*)

Ref.**__ne__**(*other*)

Deriving Refs from Refs

Ref.**padded_ref**()

Returns *Ref* with 1s inserted to make the *Ref* specific to the section level

```
>>> Ref("Genesis").padded_ref()
Ref("Genesis 1")
```

If this *Ref* is already specific to the section or segment level, it is returned unchanged.

```
>>> Ref("Genesis 1").padded_ref()
Ref("Genesis 1")
```

Ref.**subref**(*subsections*)

Returns a more specific reference than the current Ref

Parameters **subsection** – int or list - the subsection(s) of the current Ref

Returns *Ref*

Ref.**subrefs**(*length*)

Return a list of *Ref* objects one level deeper than this *Ref*, from 1 to *length*.

Parameters **length** – Number of subrefs to return

```
>>> Ref("Genesis").subrefs(4)
[Ref('Genesis 1'),
 Ref('Genesis 2'),
 Ref('Genesis 3'),
 Ref('Genesis 4')]
```

Returns List of *Ref*

Ref.**all_subrefs**(*lang='all'*)

Return a list of all the valid *Ref* objects one level deeper than this *Ref*.

```
>>> Ref("Genesis").all_subrefs()
[Ref('Genesis 1'),
 Ref('Genesis 2'),
 Ref('Genesis 3'),
 Ref('Genesis 4'),
 ...]
```

Returns List of *Ref*

Ref.**context_ref**(*level=1*)

Returns *Ref* that is more general than this *Ref*.

Parameters **level** – how many levels to 'zoom out' from the most specific possible *Ref*

```
>>> Ref("Genesis 4:5").context_ref(level = 1)
Ref("Genesis 4")
>>> Ref("Genesis 4:5").context_ref(level = 2)
Ref("Genesis")
```

If this *Ref* is less specific than or equally specific to the level given, it is returned as-is.

Ref.**section_ref**()
Return the section level Ref

For texts of depth 2, this has the same behavior as *top_section_ref()*

```
>>> Ref("Rashi on Genesis 2:3:1").section_ref()
Ref("Rashi on Genesis 2:3")
>>> Ref("Genesis 2:3").section_ref()
Ref("Genesis 2")
```

Returns *Ref*

Ref.**top_section_ref**()
Return the highest level section Ref.

For texts of depth 2, this has the same behavior as *section_ref()*

```
>>> Ref("Rashi on Genesis 2:3:1").top_section_ref()
Ref("Rashi on Genesis 2")
>>> Ref("Genesis 2:3").top_section_ref()
Ref("Genesis 2")
```

Returns *Ref*

Ref.**starting_ref**()
For ranged Refs, return the starting Ref

Returns *Ref*

Ref.**ending_ref**()
For ranged Refs, return the ending Ref

Returns *Ref*

Ref.**split_spanning_ref**()
Return list of non-spanning *Ref* objects which completely cover the area of this Ref

```
>>> Ref("Shabbat 13b-14b").split_spanning_ref()
[Ref("Shabbat 13b"), Ref("Shabbat 14a"), Ref("Shabbat 14b")]
>>> Ref("Shabbat 13b:3 - 14b:3").split_spanning_ref()
[Ref('Shabbat 13b:3-50'), Ref('Shabbat 14a'), Ref('Shabbat 14b:1-3')]
```

Ref.**next_section_ref**()
Returns a Ref to the next section (e.g. Chapter).

If this is the last section, returns `None`

Returns *Ref*

Ref.**prev_section_ref**()
Returns a Ref to the previous section (e.g. Chapter).

If this is the first section, returns `None`

Returns *Ref*

Ref.**next_segment_ref**()
Returns a *Ref* to the next populated segment.

If this ref is not segment level, will return `self`

Returns *Ref*

Ref.**prev_segment_ref**()
Returns a *Ref* to the next previous populated segment.

If this ref is not segment level, will return `self`

Returns *Ref*

Ref.**last_segment_ref**()
Returns *Ref* to the last segment in the current book (or complex book part).

Not to be confused with *ending_ref()*

Returns

Ref.**surrounding_ref**(*size=1*)
Return a reference with 'size' additional segments added to each side.

Currently does not extend to sections beyond the original ref's span.

Parameters **size** (*int*) –

Returns *Ref*

Ref.**to**(*toref*)
Return a reference that begins at this *Ref*, and ends at toref

Parameters **toref** – *Ref* that denotes the end of the new ranged *Ref*

Returns *Ref*

Getting other data with Refs

Ref.**is_text_fully_available**(*lang*)

Parameters **lang** – "he" or "en"

Returns True if at least one complete version of ref is available in lang.

Ref.**is_text_translated**()

Returns True if at least one complete version of this *Ref* is available in English.

Ref.**regex**(*as_list=False*, *anchored=True*)

Return string for a Regular Expression which will find any refs that match this Ref exactly, or more specifically.

E.g., "Genesis 1" yields an RE that match "Genesis 1" and "Genesis 1:3"

Ref.**text**(*lang='en'*, *vtitle=None*, *exclude_copyrighted=False*)

Parameters

- **lang** – "he" or "en"
- **vtitle** – optional. text title of the Version to get the text from

Returns *TextChunk* corresponding to this Ref

Ref.**versionset**(*lang=None*)
VersionsSet of *Version* objects that have content for this Ref in lang, projected

Parameters **lang** – "he", "en", or None

Returns *VersionSet*

Ref.**version_list**()
A list of available text versions titles and languages matching this ref. If this ref is book level, decorate with the first available section of content per version.

Return list each list element is an object with keys 'versionTitle' and 'language'

Ref.**linkset**()

Returns LinkSet for this Ref

Ref.**noteset**(*public=True*, *uid=None*)

Returns NoteSet for this Ref

Ref.**condition_query**(*lang=None*)
Return condition to select only versions with content at the location of this Ref. Usage:

```
VersionSet(oref.condition_query(lang))
```

Can be combined with *part_projection()* to only return the content indicated by this ref:

```
VersionSet(oref.condition_query(lang), proj=oref.part_projection())
```

Returns dict containing a query in the format expected by VersionSet

Ref.**part_projection**()
Returns the slice and storage address to return top-level sections for Versions of this ref

Used as:

```
Version().load({...},oref.part_projection())
```

Regarding projecting complex texts: By specifying a projection that includes a non-existing element of our dictionary at the level of our selection, we cause all other elements of the dictionary to be unselected. A bit non-intuitive, but a huge savings of document size and time on the data transfer. http://stackoverflow.com/a/15798087/213042

Ref.**storage_address**()
Return the storage location within a Version for this Ref.

Return string

Ref.**get_state_ja**(*lang='all'*)

Parameters **lang** – "all", "he", or "en"

Returns sefaria.datatype.jagged_array

Ref.**get_state_node**(*meta=None*, *hint=None*)

Returns sefaria.model.version_state.StateNode

CHAPTER 3

TextChunk and TextFamily

class sefaria.model.text.**TextChunk**(*oref*, *lang='en'*, *vtitle=None*, *exclude_copyrighted=False*)

A chunk of text corresponding to the provided *Ref*, language, and optionall version name. If it is possible to get a more complete text by merging multiple versions, a merged result will be returned.

Parameters

- **oref** – *Ref*
- **lang** – "he" or "en"
- **vtitle** – optional. Title of the version desired.

TextChunk.text: The text itself

TextChunk.is_merged: (Boolean) is this a merged result?

TextChunk.sources: List of sources used to create this TextChunk

class sefaria.model.text.**TextFamily**(*oref*, *context=1*, *commentary=True*, *version=None*, *lang=None*, *pad=True*, *alts=False*, *wrapLinks=False*)

A text with its translations and optionally the commentary on it.

Can be instanciated with just the first argument.

Parameters

- **oref** – *Ref*. This is the only required argument.
- **context** (*int*) – Default: 1. How many context levels up to go when getting commentary. See *Ref.context_ref()*
- **commentary** (*bool*) – Default: True. Include commentary?
- **version** – optional. Name of version to use when getting text.
- **lang** – None, "en" or "he". Default: None. If None, included both languages.
- **pad** (*bool*) – Default: True. Pads the provided ref before processing. See *Ref.padded_ref()*
- **alts** (*bool*) – Default: False. Adds notes of where alternate structure elements begin

TextFamily.text: The English language text

TextFamily.he: The Hebrew language text

contents()

Return dict Returns the contents of the text family.

CHAPTER 4

Version

class sefaria.model.text.**Version**(*attrs=None*)

A version of a text.

Relates to a complete single record from the texts collection.

required_attrs = ['language', 'title', 'versionSource', 'versionTitle', 'chapter']

optional_attrs = ['status', 'priority', 'license', 'licenseVetted', 'versionNotes', 'digitizedBySefaria', 'method', 'hevers

class sefaria.model.text.**VersionSet**(*query={}, page=0, limit=0, sort=[['priority', -1], ['_id', 1]], proj=None*)

A collection of *Version* objects

CHAPTER 5

Index, and CommentaryIndex

class sefaria.model.text.**Index**(*attrs=None*)

Index objects define the names and structure of texts stored in the system. There is an Index object for every text.

required_attrs = ['title', 'categories']

optional_attrs = ['titleVariants', 'schema', 'sectionNames', 'heTitle', 'heTitleVariants', 'maps', 'alt_structs', 'defaul

class sefaria.model.text.**IndexSet**(*query={}, page=0, limit=0, sort=[('_id', 1)], proj=None, hint=None*)

A set of *Index* objects.

Python Module Index

S

Index

Symbols

A

B

C

D

E

F

G