
SpeechCorpusTools Documentation

Release 0.1.0

SpeechCorpusTools

July 09, 2016

1	Introduction	3
1.1	General Background	3
2	Speech Corpus Tools: Tutorial and examples	5
2.1	Introduction to Tutorial	5
3	Navigation Tour	7
3.1	Queries (1)	7
3.2	Discourse (2)	10
3.3	Connection (3)	10
3.4	Details/Acoustics/Help (4)	11
4	Connection	13
4.1	IP address (or localhost)	14
4.2	Port	15
4.3	Username and Password	15
4.4	Connect	15
4.5	Find local audio files	15
4.6	Corpora	15
4.7	Import local corpus	15
5	Building Queries	17
5.1	Linguistic Objects	17
5.2	Filters	17
6	Exporting Queries	21
7	Viewing Discourses	27
8	Viewing Results	31
8.1	Utterance	31
8.2	Word	31
8.3	Phone	31
8.4	Syllable	32
9	Example: Connecting to Servers	33
10	Enrichment	37

11 Filters Explained	39
12 Indices and tables	43

Contents:

Introduction

1.1 General Background

Speech Corpus Tools is an application for interacting with large scale datasets. It uses PolyglotDB as the underlying data storage, which allows for consistent queries across a wide range of possible input formats.

Speech Corpus Tools is written in Python, which allows for Python scripts to be written using its API, so advanced users can create their own queries using Python, rather than SQL or Cypher (the underlying database languages).

In addition, Speech Corpus Tools provides a graphical user interface for easily displaying annotations and speech in the database and the results of queries.

Speech Corpus Tools: Tutorial and examples

2.1 Introduction to Tutorial

Speech Corpus Tools is a system for going from a raw speech corpus to a data file (CSV) ready for further analysis (e.g. in R), which conceptually consists of a pipeline of four steps:

1. **Import the corpus into SCT**

- Result: a structured database of linguistic objects (words, phones, discourses).

2. **Enrich the database**

- Result: Further linguistic objects (utterances, syllables), and information about objects (e.g. speech rate, word frequencies).

3. **Query the database**

- Result: A set of linguistic objects of interest (e.g. utterance-final *words* ending with a stop),

4. **Export the results**

- Result: A CSV file containing information about the set of objects of interest

Ideally, the import and enrichment steps are only performed once for a given corpus. The typical use case of SCT is performing a query and export corresponding to some linguistic question(s) of interest.

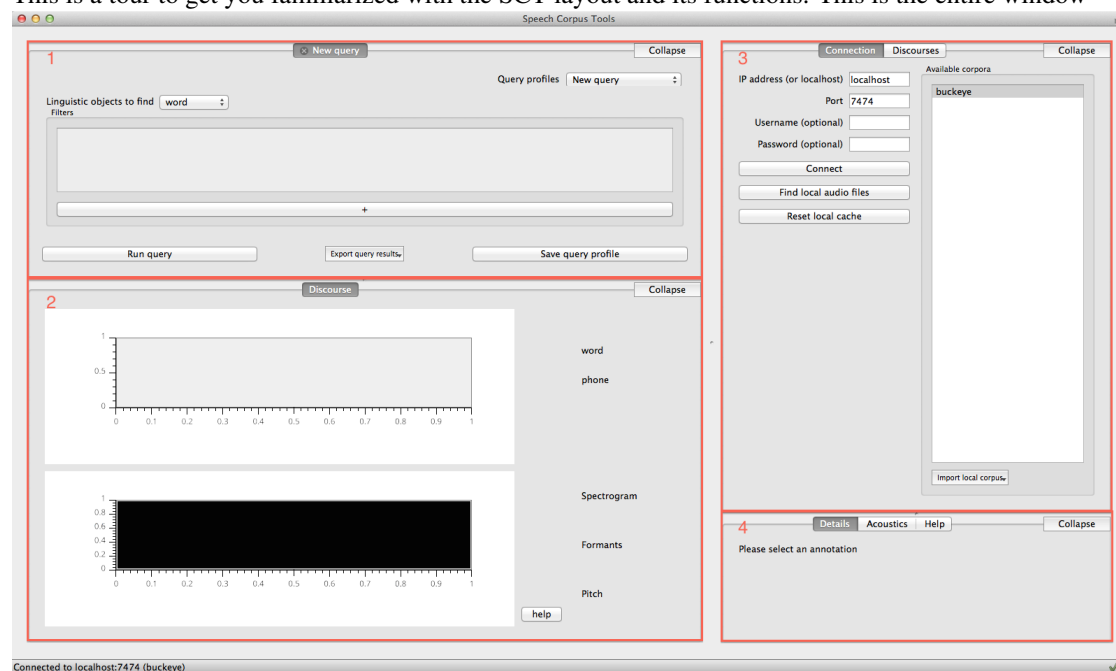
This document is structured as follows:

- **Installation:** Install necessary software
- **Librispeech database:** Obtain a database for the Librispeech Corpus where the *import* and *enrichment* steps have been completed, either by using **premade** or doing the import and enrichment steps **yourself**.
- **Examples:** Two worked examples illustrating the *Query* and *Export* steps, including creating “Query profiles” and “Export profiles”.
- **Next steps :** Next steps with SCT after the tutorial: pointers to different places in the documentation and presentations where SCT is described. Intended to kickstart carrying out your own analyses, or applying SCT to your own corpus.

Next

Navigation Tour

This is a tour to get you familiarized with the SCT layout and its functions. This is the entire window



The numbers of the panels surrounded by red rectangles correspond to:

3.1 Queries (1)

In the upper left corner, you will find the query panel

× New query
 Collapse

Query profiles
 New query

Linguistic objects to find
 utterance

Filters

+

Basic Filters

Simple queries
Complex queries

☐ utterance-initial words

☐ utterance-final words

☐ penultimate syllables

☐ syllable-initial phones

☐ syllable-final phones

☐ phones before a consonant

Run query
Export query results
Save query profile

You will begin by selecting a target type in the dropdown menu next to “Linguistic objects to find”. You can add filters by pressing the long “+” bar at the bottom of the panel. If you want to use a saved query, you can do so by selecting it from the dropdown menu on the top right of the panel.

Additionally, you can use premade templates that can be selected by checking them. Both simple queries and complex queries have been incorporated. Checking these boxes will add a fixed set of filters which correspond to that query:

Complex queries generally consist of more filters and can be checked and run just like simple queries.

× New query Collapse

Query profiles New query

Linguistic objects to find utterance

Filters

+

Basic Filters

Simple queries Complex queries

☐ all vowels in monosyllabic words

☐ phones before a word-final consonant

Run query Export query results Save query profile

Running, exporting, and saving a query are all done using the respective buttons along the bottom of the panel. *

* **NB** Running, exporting, and saving a query are all different functions. Running a query simply executes the query on the database and returns a default set of results to an in-app tab. Exporting a query runs the query on the database but allows the user to choose what information is returned, in the form of a file written to the computer. Saving a query allows the user to save a query profile and re-use it later.

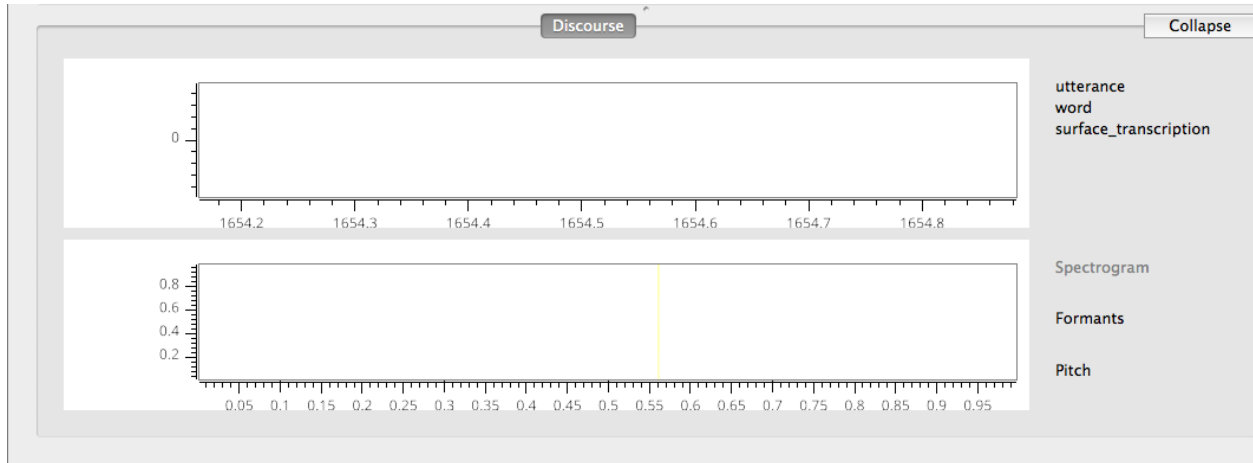
For more information see the following pages:

[Building Queries](#)

[Exporting Queries](#)

3.2 Discourse (2)

The discourse panel shows the waveform and spectrogram views of the audio for a given file (if there is audio) as well as the alignment of words, phones, and utterances (if they have been encoded) overlaid onto the waveform. For more information on viewing discourses, see [Viewing discourses](#)



3.3 Connection (3)

This panel is used to establish connections with existing databases, or to construct a new database by ‘importing’ a corpus from the hard drive. Connect to a Neo4j server by filling in the host and port information and pressing “Connect”. Import a database from the hard drive by pressing “Import Local Corpus”. If a database has already been used in SCT it does not need to be imported again. Select a corpus by clicking on it (it will then be highlighted in blue or grey). For more information, see [Connecting to servers](#)

The screenshot shows a software interface with two tabs: 'Connection' (selected) and 'Discourses'. A 'Collapse' button is in the top right. The 'Connection' panel contains input fields for 'IP address (or localhost)' (set to 'localhost'), 'Port' (set to '7474'), 'Username (optional)', and 'Password (optional)'. Below these are three buttons: 'Connect', 'Find local audio files', and 'Reset local cache'. To the right, a list titled 'Available corpora' contains 'buckeye', 'globalphone', 'sotc', and 'timit'. At the bottom of this list is an 'Import local corpus' button with a dropdown arrow.

Field	Value
IP address (or localhost)	localhost
Port	7474
Username (optional)	
Password (optional)	

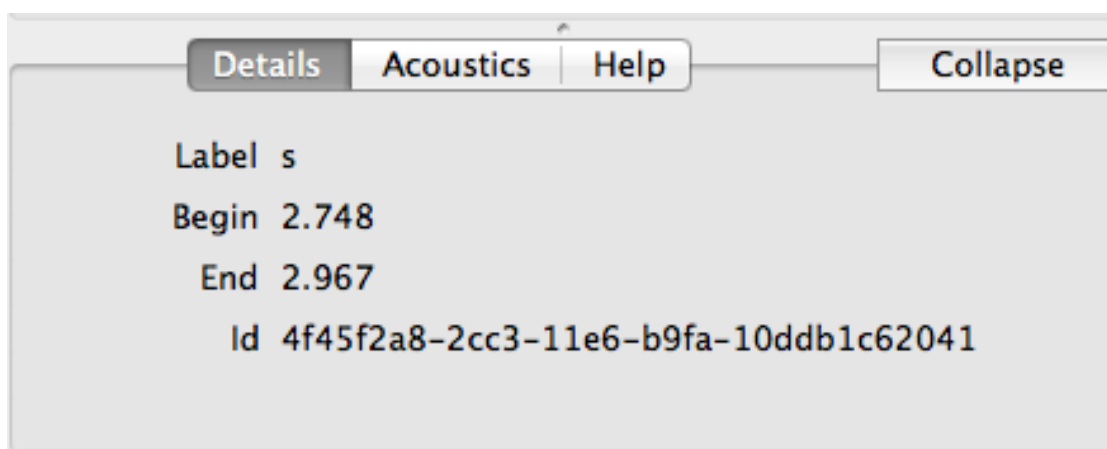
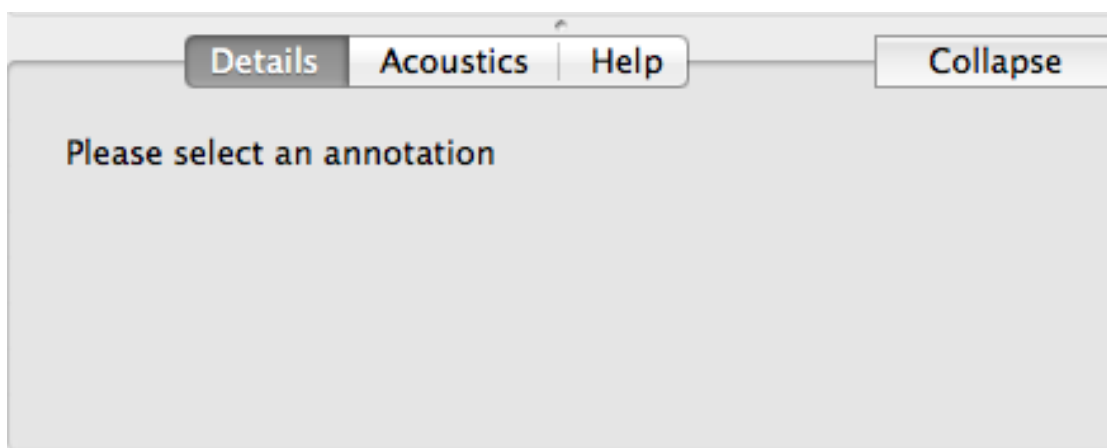
Available corpora

- buckeye
- globalphone
- sotc
- timit

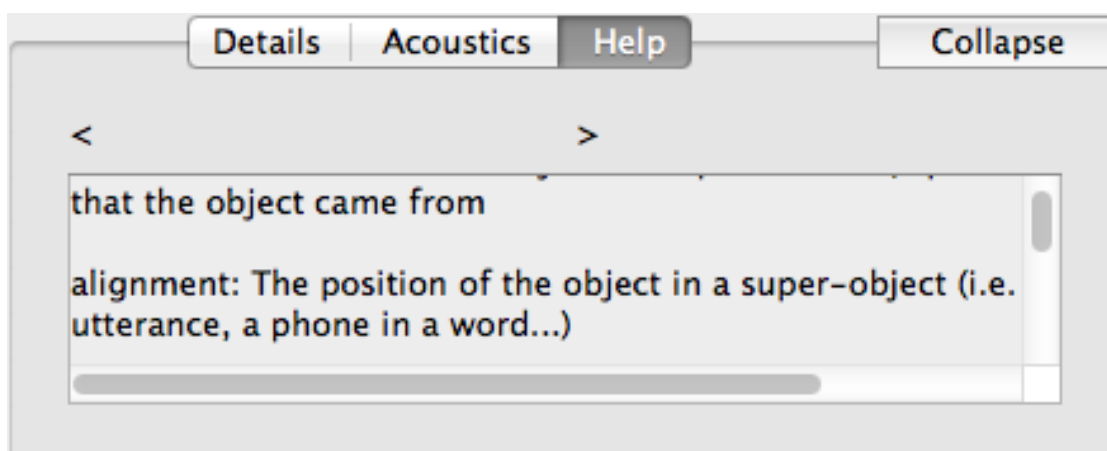
Buttons: Connect, Find local audio files, Reset local cache, Import local corpus

3.4 Details/Acoustics/Help (4)

This panel will give you details about your file, as well as precise acoustic information and help for a selected feature in the program



additional/acoustics.png



Connection

To see an example connection, go to [Connection example](#)

In the connection tab, there are various features.

The screenshot shows a software window titled "SpeechCorpusTools" with two tabs: "Connection" (selected) and "Discourses". In the top right corner, there is a "Collapse" button. The "Connection" tab contains the following elements:

- IP address (or localhost):
- Port:
- Username (optional):
- Password (optional):
- Buttons: "Connect", "Find local audio files", and "Reset local cache".

On the right side of the window, there is a section titled "Available corpora" containing a list box with the following items:

- buckeye
- globalphone
- sotc
- timit

Below the list box is a button labeled "Import local corpus".

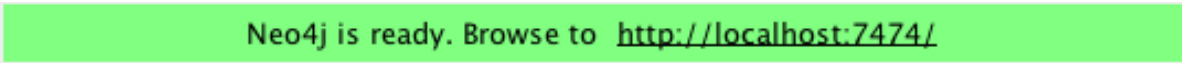
These are detailed below

4.1 IP address (or localhost)

This is the address of the Neo4j server. In most cases, it will be 'localhost'

4.2 Port

This is the port through which a connection to the Neo4j server is made. By default, it is 7474. It must always match the port shown in the console output:



Neo4j is ready. Browse to <http://localhost:7474/>

4.3 Username and Password

These are by default not required, but available should you need authentication for your Neo4j server

4.4 Connect

This button will actually connect the user to the specified server.

4.5 Find local audio files

Pressing this allows the user to browse his/her file system for directories containing audiofiles that correspond to files in a corpus.

4.6 Corpora

The user select a corpus (for running queries, viewing discourses, enrichment, etc.) by clicking that corpus in the “Available corpora” menu. The selected corpus will be highlighted in blue or grey.

4.7 Import local corpus

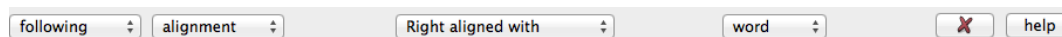
This is strictly for constructing a new relational database in Neo4j that does not already exist. Any corpus that has already been imported can be accessed by pressing “Connect” and selecting it instead. Re-importing the same corpus will overwrite the previous corpus of the same name, as well as remove any enrichment the user has done on the corpus.

When importing a new corpus, the user selects the directory of the overall corpus, not specific files or subdirectories.

Building Queries

In this panel, the user constructs queries by adding filters (these will be explained more thoroughly in a moment). There are two key concepts that drive a query in SCT:

- **Linguistic Object** A linguistic object can be an utterance, word, or phone. By selecting a linguistic object, the user is specifying the set of elements over which the query is to be made. For example, selecting “phones” will cause the program to look for phones with properties specified by the user (if “words” were selected, then the program would look for words, etc.)
- **Filters** Filters are statements that limit the data returned to a specific set. Each filter added provides another constraint on the data. Click [here](#) for more information on filters. Here’s an example of a filter:



This filter specifies all the object (utterance, phone, syllable) which are followed by an object of the same type that shares its rightmost boundary with a word.

Now you’re ready to start building queries. Here’s an overview of what each dropdown item signifies

5.1 Linguistic Objects

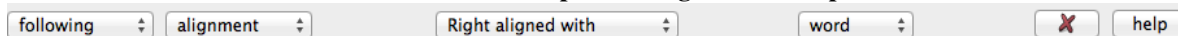
- **Utterance:** An utterance is (loosely) a group of sounds delimited by relatively long pauses on either side. This could be a clause, sentence, or phrase. Note that utterances need to be encoded before they are available.
- **Syllables** Syllables currently have to be encoded before this option is available. The encoding is done through maximum attested onset
- **Word:** A word is a collection of phones that form a single meaningful element.
- **Phone:** A phone is a single speech segment.

The following is available only for the TIMIT database:

- **surface_transcription** This is the phonetic transcription of the utterance

5.2 Filters

Filters are conditions that must be satisfied for data to pass through. For example



is a filter



Many filters have dropdown menus. These look like this:

Generally speaking, the first dropdown menu is used to target a property. These properties are available without enrichment for all databases:

- **alignment** The position of the object in a super-object (i.e. a word in an utterance, a phone in a word...)
- **following** Specifies the object after the current object
- **previous** Specifies the object before the current object
- **subset** Used to delineate classes of phones and words. Certain classes come premade. Others are available through enrichment
- **duration** How much time the object occupies
- **begin** The start of the object in time (seconds)
- **end** The end of the object in time (seconds)
- **label** The orthographic contents of an object
- **word** Specifies a word (only available for Utterance, Syllable, and Phone)
- **syllable** Specifies a syllable
- **phone** Specifies a phone
- **speaker** Specifies the speaker
- **discourse** Specifies the discourse, or file
- **category** Only available for words, specifies the word category
- **transcription** Only available for words, specifies the phonetic transcription of the word in the corpus

These are available after enrichment:

- **utterance** Available for all objects except utterance, specifies the utterance that the object came from
- **syllable_position** Only available for phones, specifies the phone's position in a syllable
- **num_phones** Only available for words, specifies the number of phones in a word
- **num_syllables** Only available for words, specifies the number of syllables in a word
- **position_in_utterance** Only available for words, specifies the word's index in the utterance

These are only available for force-aligned database:

- **manner_of_articulation** Only available for phones
- **place_of_articulation** Only available for phones
- **voicing** Only available for phones
- **vowel_backness** Only available for phones
- **vowel_rounding** Only available for phones

- **vowel_height** Only available for phones
- **frequency** Only available for words, specifies the word frequency in the corpus
- **neighborhood_density** Only available for words, specifies the number of phonological neighbours of a given word.
- **stress_pattern** Only available for words, specifies the stress pattern for a word

The second filter will depend on which filter you chose in the first column. For example, if you chose phone you will get all of the

- **alignment**

- **right aligned with** This will filter for objects whose rightmost boundary lines up with the rightmost boundary of the object you will select in the third column of dropdown menus (**utterance, syllable, word, or phone**).
- **left aligned with** This will filter for objects whose leftmost boundary lines up with the left most boundary of the object you will select in the third column of dropdown menus (**utterance, syllable, word, or phone**).
- **not right aligned with** This will exclude objects whose rightmost boundary lines up with the rightmost boundary of the object you will select in the third column of dropdown menus (**utterance, syllable, word, or phone**).
- **not left aligned with** This will exclude objects whose leftmost boundary lines up with the left most boundary of the object you will select in the third column of dropdown menus (**utterance, syllable, word, or phone**).

- **subset**

- **==** This will filter for objects that are in the class that you select in the third dropdown menu.

- **begin/end/num_phones/num_syllables/ position_in_utterance/frequency/ neighborhood_density/duration**

- **==** This will filter for objects whose property is equal to what you have specified in the text box following this menu.
- **!=** This will exclude objects whose property is equal to what you have specified in the text box following this menu.
- **>=** This will filter for objects whose property is greater than or equal to what you have specified in the text box following this menu.
- **<=** This will filter for objects whose property is less than or equal to what you have specified in the text box following this menu.
- **>** This will filter for objects whose property is greater than what you have specified in the text box following this menu.
- **<** This will filter for objects whose property is less than what you have specified in the text box following this menu.

- **stress_pattern/category/label/ speaker + name/discourse + name/ transcription/vowel_height/ vowel_backness/vowel_**

- **==** This will filter for objects whose property is equivalent to what you have specified in the text box or dropdown menu following this menu.
- **!=** This will exclude objects whose property name is equivalent to what you have specified in the text box or dropdown menu following this menu.
- **regex** This option allows you to input a regular expression to match certain properties.

Experiment with combining these filters. Remember that each time you add a filter, you are applying further constraints on the data.

Some complex queries come pre-made. These include “all vowels in mono-syllabic words” and “phones before word-final consonants”.

- **All vowels in mono-syllabic words**

- Since we’re looking for vowels, we know that the linguistic object to search for must be “phones”
- **To get mono-syllabic words, we have to go through three phases of enrichment**
 - * First, we need to encode syllabic segments
 - * Second, we need to encode syllables
 - * Finally, we can encode the hierarchical property: count of syllables in word
- Now that we have this property, we can add a filter to look for monosyllabic words:

```
word: count_of_syllable_in_word == 1
```

- Notice that we had to select “word” for “count_of_syllable_in_word” to be available
- The next filter we want to add would be to get only the vowels from this word.

```
subset == syllabic
```

- This will get the syllabic segments (vowels) that we encoded earlier

- **Phones before word-final consonants**

- Once again, it is clear that we are looking for “phones” as our linguistic object.
- The word “before” should tip you off that we will need to use the “following” or “previous” property.
- We start by getting all phones that are in the penultimate position in a word.

```
following phone right-aligned with word
```

- This will ensure that the phone after the one we are looking for is the word-final phone
- Now we need to limit it to consonants

```
following phone subset != syllabic
```

- This will further limit the results to only phones before non-syllabic word-final segments (word-final consonants)

Exporting Queries

While getting in-app results can be a quick way to visualize data, most often the user will want to further manipulate the data (i.e. in R, MatLab, etc.) To this end, there is the “Export query results” feature. It allows the user to specify the information that is exported by adding columns to the final output file. This is somewhat similar to [building queries](#), but not quite the same. Instead of filters, pressing the “+” button will add a column to the exported file.

For example, if the user wanted the timing information (begin/end) and lables for the object found and the object before it, the e

Linguistic objects to find phone

Columns

label		Output name:	label	X
begin		Output name:	begin	X
label		Output name:	label	X
previous	label	Output name:	previous_label	X
previous	begin	Output name:	previous_begin	X
previous	end	Output name:	previous_end	X

Perhaps a researcher would be interested in knowing whether word-initial segments in some word categories are longer than in

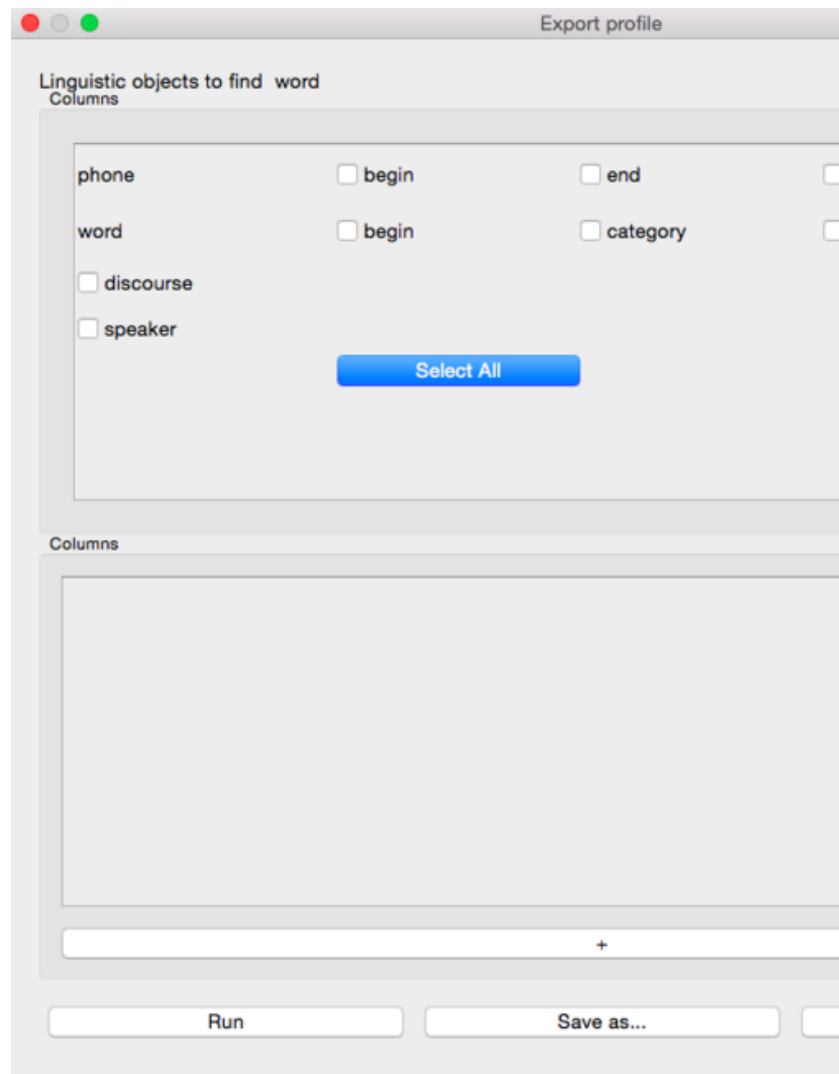
Linguistic objects to find phone
Columns

label	Output name:	label	X
begin	Output name:	begin	X
end	Output name:	end	X
duration	Output name:	duration	X
word	category	Output name: word_category	X

+

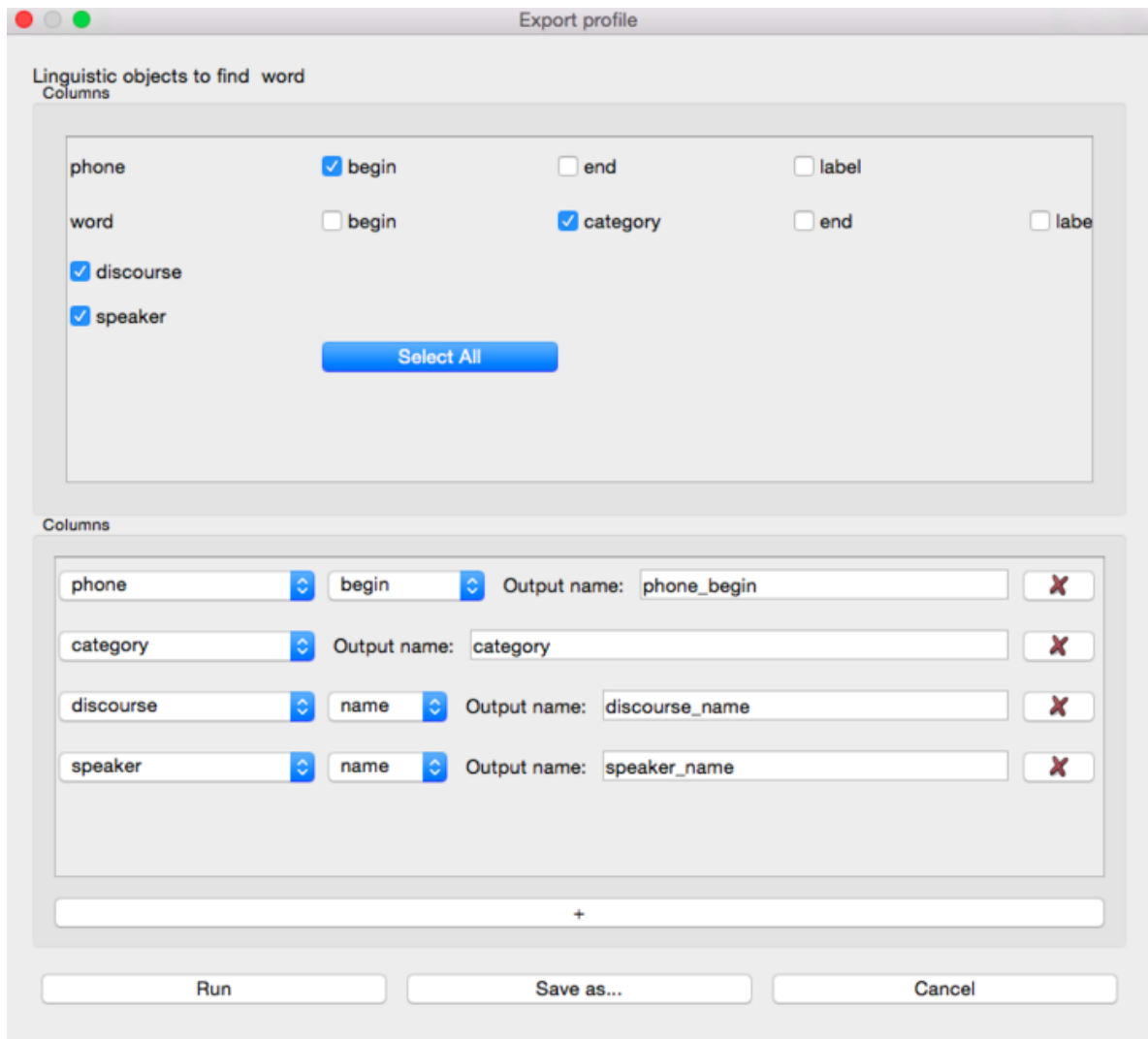
Run Save as... Cancel

Here, “phone” has been selected as the linguistic object to find (since that is what we’re interested in) so any property without a preceding dropdown menu is a property of the target phone – in this case, alignment would have been used to specify “word-initial phones”.



Another option is to use the “simple export” window.

Here, there are several common options that can be selected by checking them. Once checked, they will appear as columns in the



While many of the column options are the same as ones available for [building queries](#) there are some differences :

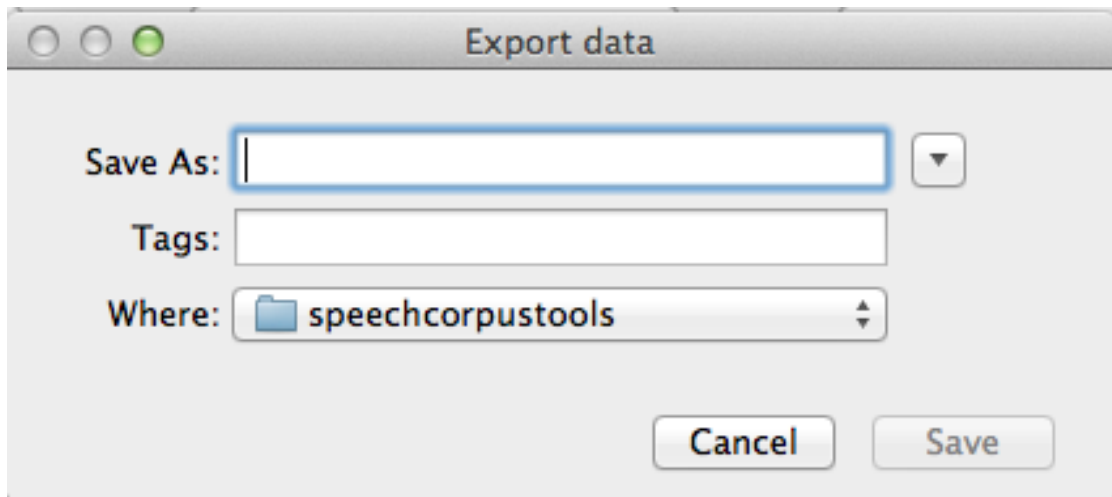
- “alignment” and “subset” are not valid column options
- **column options do not change depending on the linguistic object that was chosen earlier**
 - instead, you can select “word” and then “label” (or some other option) or “phone” + options, etc.
- you can edit the column name by typing what you would like to call it in the “Output name:” box. These names are by default very descriptive, but perhaps too long for the user’s purposes.

Since the options are similar but not all identical, here is a full list of all the options available:

- **following** Specifies the object after the current object. There will be another dropdown menu to select a property of this following object.
- **previous** Specifies the object before the current object. There will be another dropdown menu to select a property of this preceding object.
- **duration** Adds how much time the object occupies as a column
- **begin** Adds the start of the object in time (seconds) as a column
- **end** Adds the end of the object in time (seconds) as a column

- **label** Adds the orthographic contents of an object as a column
- **word** Specifies a word (another dropdown menu will become available to specify another property to add as a column). The following properties are available:
 - **category** Adds the word category as a column
 - **transcription** Adds the underlying phonetic transcription of the word in the corpus as a column
 - **surface_transcription** Adds the surface transcription of the word in the corpus as a column
 - **utterance** Specifies the utterance that the word came from (another dropdown menu will become available to specify another property to add as a column)
- **phone** Specifies a phone (another dropdown menu will become available to specify another property to add as a column)
- **speaker** Specifies the speaker (another dropdown menu will become available to specify another property to add as a column)
- **discourse** Specifies the discourse, or file (another dropdown menu will become available to specify another property to add as a column)

Once the profile is ready, pressing “run” will open the following window:



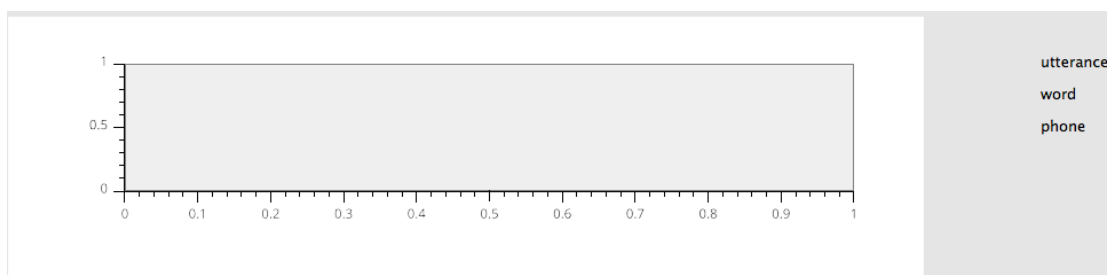
Here the user can pick a name and location for the final file. After pressing save, the query will run and the results will be written in the desired columns to the file.

Viewing Discourses

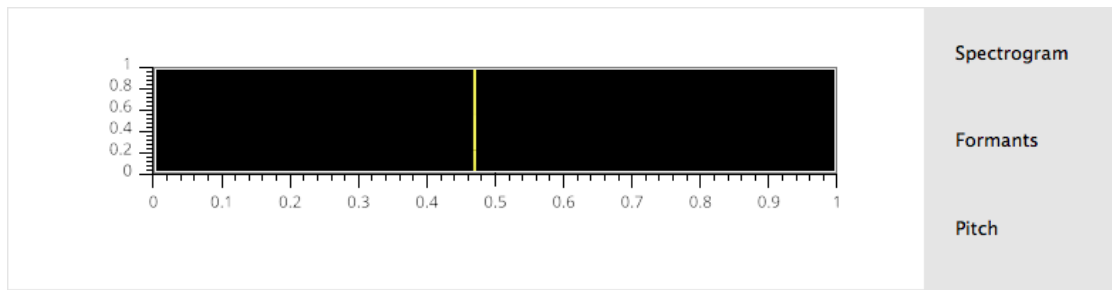
After completing a query, it might be useful to take a closer look at the discourse, or file, that a result came from. To this end, SCT has the ‘Discourse’ window on the bottom left.



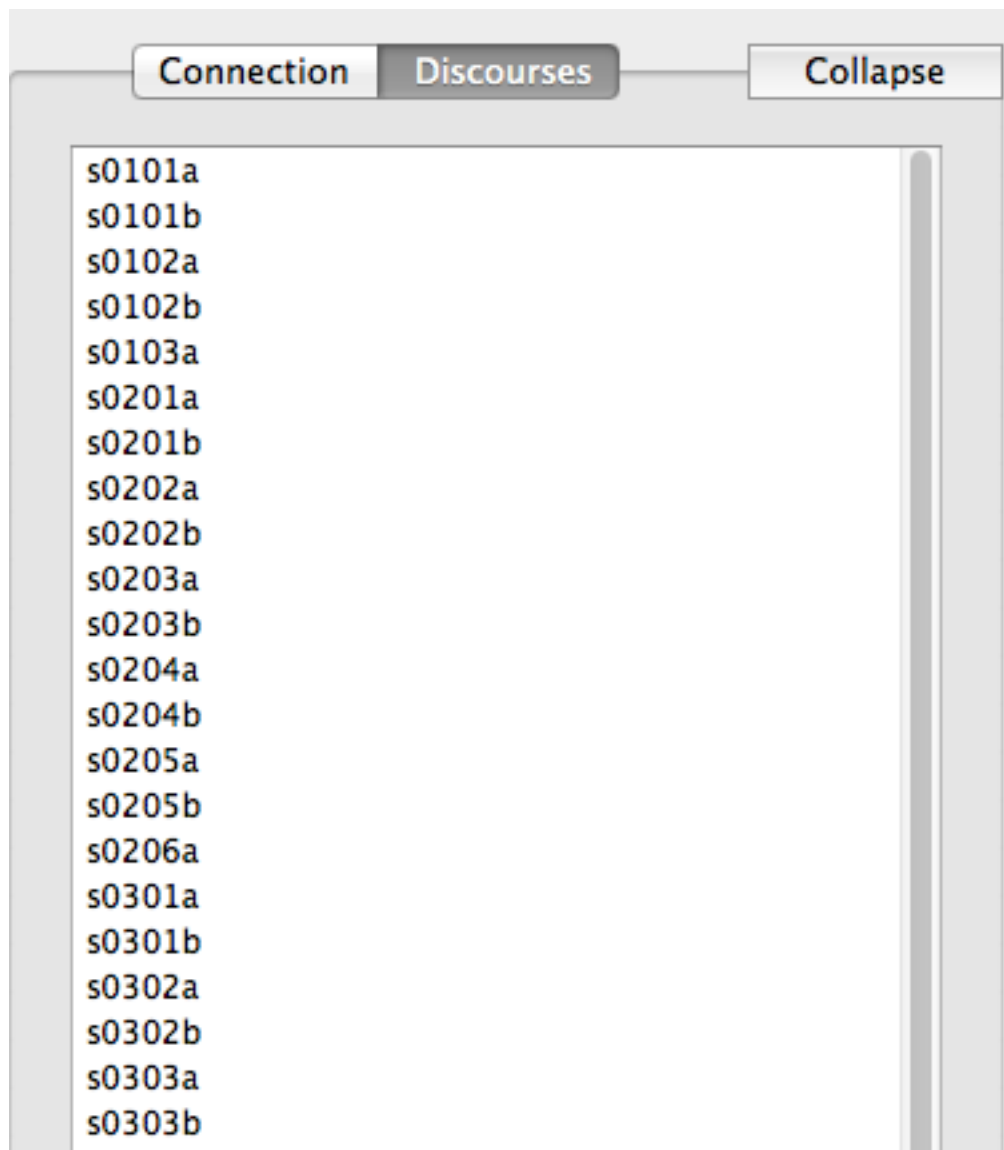
The user is presented with two windows inside of the ‘Discourse’ window. The top one shows the waveform of the file as well as the transcriptions of words and phones.



The bottom window is a spectrogram. This maps time and frequency on the X and Y axes respectively, while the darkness of an area indicates the amplitude. Lines generated by the software also indicate pitch and formants when available.



Pitch and formants will only become available by first selecting “Analyze acoustics” in the enrichment menu. Viewing one of the discourses’ acoustic information can be done by clicking on a discourse either in the “Discourse” tab of the top right window (right next to “Connection”),



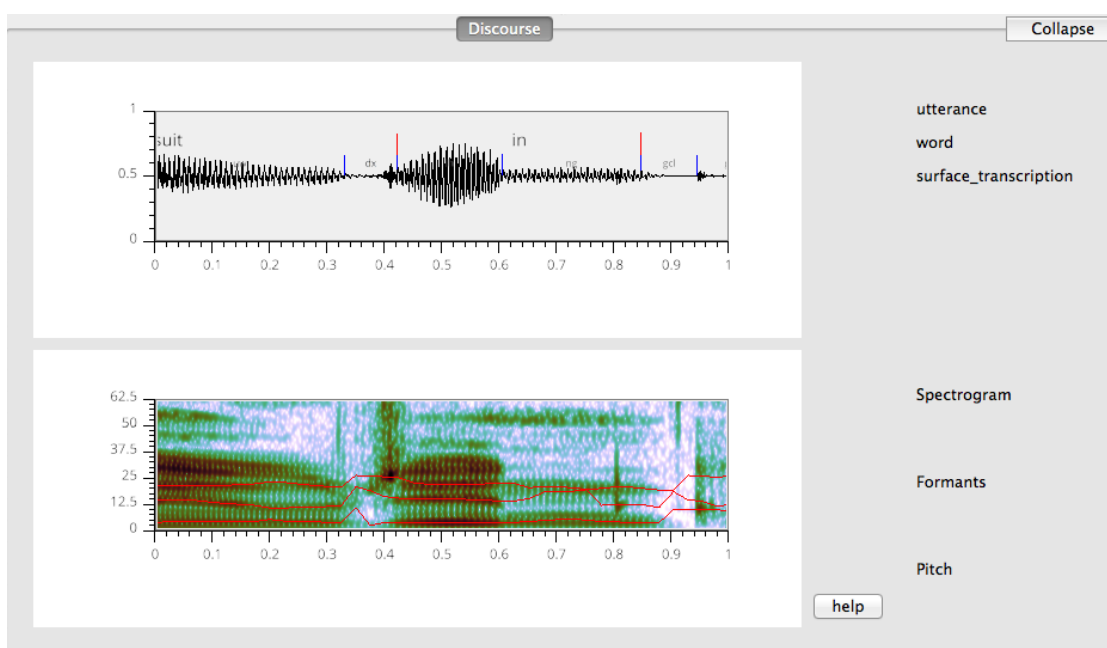
New query Query 1

	begin ▼	category	end	label
1	165.15	IN	165.354	that
2	164.941	VBP	165.15	mean
3	164.881	PRP	164.941	i
4	164.26	NULL	164.881	<LAUGH>
5	166.23	RB	166.67	very
6	166.056	VBP	166.23	are
7	165.679	RB	166.056	just
8	165.354	PRP	165.679	they

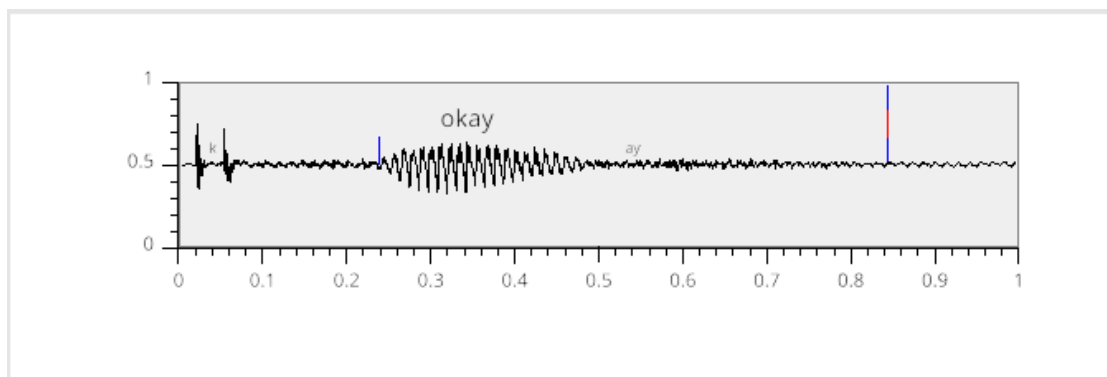
or by double-clicking on a result from a query in the “Query #” tab*.

* **NB** A successful query must first be run for this tab to appear

Now something like this should be displayed:

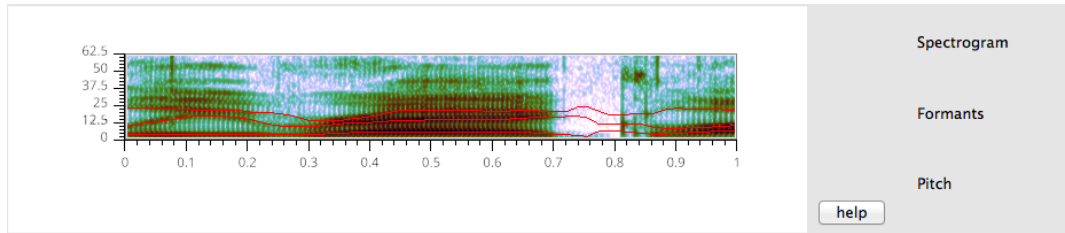


The waveform is displayed, with annotations

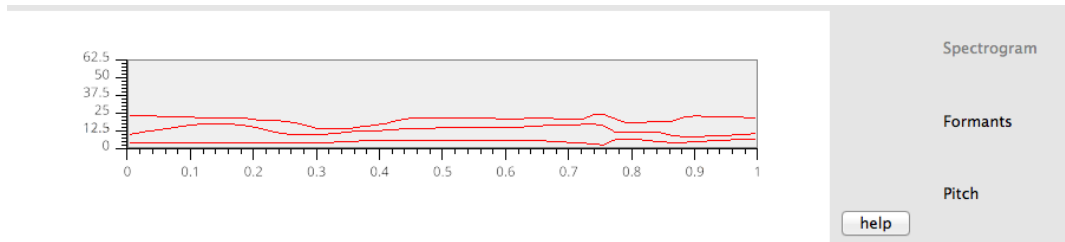


as well as the spectrogram, whose features can be toggled on and off by clicking on them.

- Spectrogram On



- Spectrogram Off (just formants and pitch)



Viewing Results

Having run a query, a user will want to make sense of the results. These can be found in the “Query #” that will appear as soon as the query has finished running.

Within this tab, based on the linguistic objects the user was searching for (utterance, word, phone, or syllable) there will be different columns*. Here is a list of the default columns

8.1 Utterance

- **begin**
- **end**
- **discourse**
- **speaker**

8.2 Word

- **begin**
- **category** * only in buckeye
- **end**
- **label**
- **surface_transcription** * only in buckeye
- **transcription**
- **discourse**
- **speaker**

8.3 Phone

- **begin**
- **end**

- **label**
- **discourse**
- **speaker**

8.4 Syllable

- **begin**
- **end**
- **label**
- **discourse**
- **speaker**

* **NB** Scrolling horizontally may be required to view all of these options.

Example: Connecting to Servers

If you already have Neo4j open and started, you're ready to start connecting to servers.

Go to the upper right panel in SCT

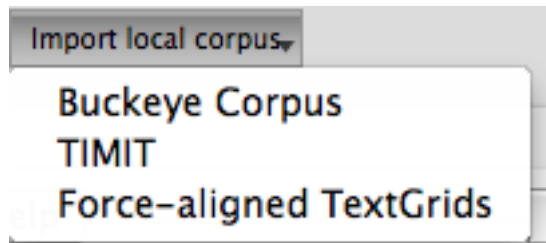
The screenshot shows the 'Connection' tab of the SpeechCorpusTools application. The interface has a light gray background. At the top, there are two tabs: 'Connection' (selected) and 'Discourses'. To the right of these tabs is a 'Collapse' button. Below the tabs, on the left side, there are four input fields: 'IP address (or localhost)' with the value 'localhost', 'Port' with the value '7474', 'Username (optional)', and 'Password (optional)'. Below these fields are three buttons: 'Connect', 'Find audio', and 'Reset local cache'. On the right side, there is a large rectangular area labeled 'Available corpora'. At the bottom of this area is a button labeled 'Import local corpus'.

You're not connected to the Neo4j graph database the first time you start the program. Let's fix that. Make sure that the port is the same as in your Neo4j window.

Neo4j is ready. Browse to <http://localhost:7474/>

If they match, you're ready to proceed. Press connect. Because it is your first time using the program, nothing will appear in "Available Corpora", but the "Reset Local Cache" button should now be clickable.

Next, go to “Import Local Corpus” at the bottom center and click on it.



Press “Buckeye Corpus”. This was included with the tutorial. Go to the tutorial folder and select “buckeyeDataFor-Tutorial”. You will have to wait for the corpus to be imported.

When the process has completed, you are ready to make some queries. Simply select the corpus by clicking on it under “Available corpora” and begin adding filters.

Enrichment

Databases can be enriched by encoding various elements. Usually, the database starts off with just words and phones, but by using enrichment options a diverse range of options will become available to the user. Here are some of the options:

- **Encode non-speech elements** this allows the user to specify for a given database what should not count as speech
- **Encode utterances** After encoding non-speech elements, we can use them to define utterances (segments of speech separated by a .15-.5 second pause)
- **Encode syllabic segments** This allows the user to specify which segments in the corpus are counted as syllabic
- **Encode syllables** if the user has encoded syllabic segments, syllables can now be encoded using maximum attested onset
- **Encode hierarchical properties** These allow the user to encode such properties as number of syllables in each utterance, or rate of syllables per second
- **Enrich lexicon** This allows the user to assign certain properties to specific words. For example the user might want to encode word frequency. This can be done by having words in one column and corresponding frequencies in the other column of a column-delimited text file.
- **Enrich phonological inventory** Similar to lexical enrichment, this allows the user to add certain helpful features to phonological properties – for example, adding ‘fricative’ to ‘manner_of_articulation’ for some phones
- **Encode subsets** Similar to how syllabic phones were encoded into subsets, the user can encode other phones in the corpus into subsets as well
- **Analyze acoustics** This will encode pitch and formants into the corpus. This is necessary to view the waveforms and spectrogram.

Filters Explained

So far, there has been a lot of talk about objects, filters, and alignment, but these can be a difficult-to-grasp concepts. These illustrated examples might be helpful in gleaning a better understanding of what is meant by “object”, “filter” and “alignment”.

The easiest way to start is with an example. Let’s say the user wanted to search for **word-final fricatives in utterance-initial words**

While to a person this seems like a fairly simple task that can be accomplished at a glance, for SCT it has to be broken up into its constituent steps. Let’s see how this works on this sample sentence:

Utterance	These reasons for this dive seemed foolish now																												
Word	These			reasons				for		this		dive		seemed			foolish				now								
Phone	DH	IY	Z	R	IY	Z	AX	N	Z	F	AO	R	DH	IH	S	D	AI	V	S	IY	M	D	F	UW	L	IH	SH	N	AW

Here, each level (utterance, word, phone) corresponds to an object. Since we are ultimately looking for fricatives, we would want to select “phones” as our linguistic object to find.

Right now we have all phones selected, since we haven’t added any filters. Let’s limit these phones by adding the first part of our desired query: word-final phones. To accomplish this, we need to grasp the idea of alignment.

Each object (utterances, words, phones) has two boundaries, left and right. These are represented by the walls of the boxes containing each object in the picture. To be “aligned”, two objects must share a boundary. For example, the non-opaque objects in the next 2 figures are all aligned. Shared boundaries are indicated by thick black lines. Parent objects (for example, words in which a target phone is found) are outlined in dashed lines. In the first picture, the words and phones are “left aligned” with the utterance (their left boundaries are the same as that of the utterance) and in the second image, words and phones are “right aligned” with the utterance.

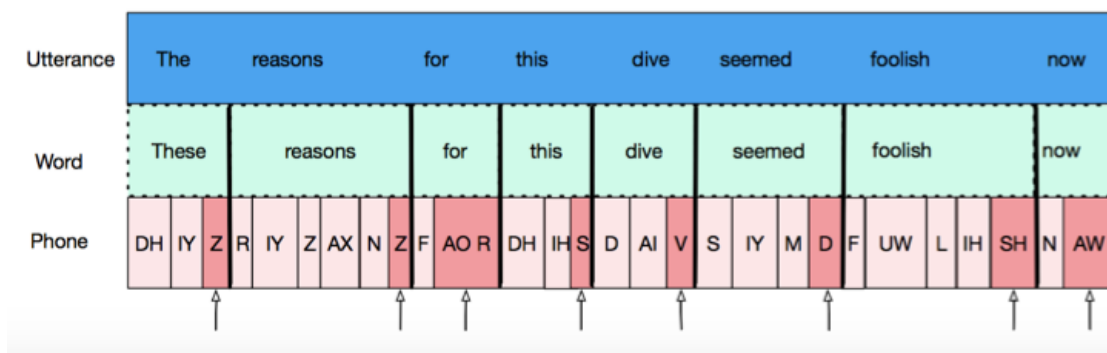


Now that we understand alignment, we can use it to filter for **word-final** phones, by adding in this filter:

Linguistic objects to find

Filters

By specifying that we only want phones which share a right boundary with a word, we are getting all **word-final** phones.



However, recall that our query asked for **word-final fricatives**, and not all phones. This can easily be remedied by adding another filter *:

Linguistic objects to find

Filters

alignment word

subset fricative

* **NB** the “fricative” property is only available through [enrichment](#)

Now the following phones are found:

Utterance	The reasons for this dive seemed foolish now																												
Word	These			reasons					for		this		dive		seemed			foolish			now								
Phone	DH	IY	Z	R	IY	Z	AX	N	Z	F	AO	R	DH	IH	S	D	AI	V	S	IY	M	D	F	UW	L	IH	SH	N	AW
			↑						↑						↑			↑									↑		

Finally, in our query we wanted to specify only **utterance-initial** words. This will again be done with alignment. Since English reads left to right, the first word in an utterance will be the leftmost word, or the word which shares its leftmost boundary with the utterance. To get this, we add the filter:

Linguistic objects to find

Filters

alignment word

subset fricative

word alignment utterance

This gives us the result we are looking for: **word-final fricatives in utterance-initial words**

Utterance	The reasons for this dive seemed foolish now																												
Word	These		reasons					for		this		dive		seemed			foolish			now									
Phone	DH	IY	Z	R	IY	Z	AX	N	Z	F	AO	R	DH	IH	S	D	AI	V	S	IY	M	D	F	UW	L	IH	SH	N	AW

↑

Another thing we can do is specify previous and following words/phones and their properties. For example: what if we wanted the final segment of the second word in an utterance?

Utterance	The reasons for this dive seemed foolish now																												
Word	These		reasons					for		this		dive		seemed			foolish			now									
Phone	DH	IY	Z	R	IY	Z	AX	N	Z	F	AO	R	DH	IH	S	D	AI	V	S	IY	M	D	F	UW	L	IH	SH	N	AW

↑

This is where the “following” and “previous” options come into play. We can use “previous” to specify the object before the one we are looking for. If we wanted the last phone of the second word in our sample utterance (the “s” in “reasons”) we would want to specify something about the previous word’s alignment. If we wanted to get the final phone of the words in this position, our filters would be:

Linguistic objects to find phone

Filters

alignment

Right aligned with

word

✕
help

previous

word

alignment

Left aligned with

utterance

✕
help

For a full list of filters and their uses, see the section on [building queries](#)

Indices and tables

- `genindex`
- `modindex`
- `search`