OLD Documentation

Release 1.0.0

Joel Dunham

February 16, 2013

CONTENTS

Contents:

Contents

CHAPTER

ONE

1 OLD USER GUIDE

This guide provides general information about what the Online Linguistic Database (OLD) is and how it should be used.

CHAPTER

TWO

2 ABOUT THE OLD

The OLD is a web application designed to be used by a group of individuals in order to document, analyze and learn a particular natural language.

As a linguistics PhD student whose research involves the documentation and analysis of understudied languages, I was originally motivated to create the OLD because I saw a lack of open source, cross-platform and multi-user database applications suitable for documenting linguistic data. The OLD is my effort to fill that void.

2.1 2.1 What the OLD Will Allow You To:

- build a collaborative and ever-growing online database of linguistic data on a particular language
- create a dictionary-like interface to your language data
- organize your linguistic data into an intelligent structure
- perform powerful searches on your data, utilizing regular expressions and boolean operators with multiple restrictors on multiple fields
- export data to a variety of formats (plain text, XML, LaTeX)
- incorporate non-textual data types (audio/video/images as recordings or stimuli) and specify the relationships between textual and non-textual data types
- document multi-sentence texts such as stories or records of elicitations
- collaborate and share data with other researchers authorized to use your OLD application
- control access to your data via password-protected accounts for registered users

2.2 2.2 Technical Specifications:

- programming language is Python (2.6), using the Pylons (0.9.7) web framework
- model is a relational database (usually MySQL or SQLite) abstracted by SQLAlchemy
- user interface is HTML, CSS and Javascript

CHAPTER

THREE

3 HOW TO GET THE OLD

The OLD is open source software licensed under the GPL. That means you can download it, use it and alter its source code, but you cannot sell it for profit.

The web page of the OLD is www.onlinelinguisticdatabase.org and the source code is hosted on Google Code's Project Hosting (code.google.com/p/onlinelinguisticdatabase).

In order to create an OLD application, you must download the OLD, install it on a server and customize it to suit the requirements of your language. It is possible to install the OLD on, and run/test it from, your own computer without the need for a server.

The OLD web page provides instructions on downloading the OLD and setting up a language-specific OLD application.

CHAPTER

FOUR

4 HOW TO USE THE OLD

This section describes how to use an OLD application, i.e., how to add, structure, search for, export, and otherwise interact with language data.

4.1 4.1 Basics

At the heart of the OLD are three types of entity: Forms, Files and Collections.

Figure 4.1: The three entities of the OLD

A Form is a morpheme, word, phrase or sentence of the object language. Forms minimally consist of a transcription and a gloss (i.e., translation in the metalanguage). Optional data include a morphemic analysis, general comments, date elicited, etc. As the diagram indicates, a Form may be associated to (i.e., reference or point to) one or more Files (see below).

A File is an audio, image, video, or text file. A File could represent a recording of an utterance, a depiction of a context, or a stimulus used in elicitation.

A Collection is an ordered list of Forms. A Collection might represent a story or a record of an elicitation or any other multi-sentence (and therefore multi-Form) piece of discourse. Like a Form, a Collection can be associated to one or more Files.

Interacting with an OLD application typically involves creating, searching for or updating a Form, File or Collection. Other common actions include adding or modifying people (users or speakers), tags (keywords, syntactic categories or elicitation methods) or sources.

4.2 4.2 User Interface

The primary menu contains the database, dictionary, help and settings options.

The contents of the secondary menu depend on the currently active primary menu option. When the database option is active in the primary menu, the secondary menu contains the people, tags, sources and memory options. When the

dictionary option is active, the secondary menu contains browse and search options. The help and settings primary menu options make the secondary menu disappear.

The side menu contains links to common actions on OLD entities, namely adding and searching for Forms, Files and Collections.

Figure 4.2: The basic OLD interface

CHAPTER

FIVE

5 FORMS

As stated above, a Form is a grouping of data that are about a morpheme, word, phrase or sentence of the object language.

5.1 5.1 Form Data

This subsection describes the types of data that comprise a Form. In the context of the relational database model, these types of data are the columns of the Form table.

5.1.1 5.1.1 ID

The ID is a unique integer assigned by the RDBMS to each Form upon creation. Knowing the ID of a Form comes in handy when you want to associate that Form to a Collection. It is also good to know for when you want to quickly access a Form.

For example, enter "domain_name/form/view/11" (where "domain_name" is your OLD application's domain name, e.g., "www.old.org") in your browser's address bar to view form with ID 11.

You can also enter a comma-separated list of Form IDs to quickly access several Forms: "domain_name/form/view/11,12,13,14"

5.1.2 5.1.2 transcription

The transcription is a textual representation of the sound of a Form.

A transcription is an object language string and as such it should be written using the graphs of the input orthography. (If not specified in the current user's settings, the input orthography is the default input orthography specified in the application settings.) Before being stored in the database, the transcription will be converted from the input orthography to the storage orthography.

The recommended position of the transcription along the "broad-narrow" "phonetic/phonemic" spectrum should be stated in your particular OLD application's help section.

A multi-sentence discourse should not be entered as a single form, but as multiple Forms, grouped together and ordered into a Collection.

It is recommended that transcriptions contain only the following (strings of) characters:

- graphs from the input orthography
- standard punctuation: " ' ( ) ! ? , ; :

(In future versions of the OLD, functionality may be created that would output a warning after, or even disallow, entry of transcriptions violating these conditions.)

5.1.3 5.1.3 grammaticality

The grammaticality indicates whether a particular object language Form is grammatical (), ungrammatical ('*') or questionable ('?'). (Either more grammaticality options should be added as standard or administrators/contributors should be able to customize the forced-choice grammaticality field.)

When adding a Form, grammaticality judgments should not be indicated in the text of the transcription. Instead, use the forced choice select field to the left of the transcription input field.

5.1.4 5.1.4 morpheme break

The morpheme break field contains a morphological analysis of the form. The system is currently preset to expect '-' and '=' as morpheme delimiters and ' ' as the word delimiter, but this could/should be made customizable on an application-specific basis.

The morpheme break field may or may not be specified as an object language string field. Such specification is made by administrators in the application settings page. If the morpheme break field is set up as an object language string field, then morpheme break input will be converted to the storage orthography for storage and converted to the output orthography for display (just like the data in the transcription field.) If the morpheme break field is not set up as an object language string field, then no conversion will be applied.

Whenever a Form is entered or updated, the OLD attempts to identify all of that Form's morpheme-gloss pairs and searches for matches in existing Forms. If one or more matches are found, then the morpheme and its gloss are displayed as HTML links the match(es). This allows users to immediately see the extent to which their morphological analyses are consistent with the rest of the data in the system.

To understand this morphological linking in detail, imagine an OLD application containing the following two Forms:

ID	1
transcription	chien
morpheme break	chien
morpheme gloss	dog
gloss	dog, mutt

ID	2
transcription	s
morpheme break	s
morpheme gloss	plrl
gloss	plural marker

Now, when the following Form is entered,

transcription	chiens
morpheme break	chien-s
morpheme gloss	dog-PL
gloss	dogs

the system identifies the following morpheme-gloss pairs ('chien'-'dog' and 's'-'PL'). It first searches the database for a Form with 'chien' as its morpheme break value and 'dog' as its morpheme gloss value. It finds such a match in Form 1 and as a result it displays both 'chien' and 'dog' in our newly entered Form as links to Form 1. The link is displayed in blue font to indicate a perfect match.

Its second search is for a Form with 's' as its morpheme break value and 'PL' as its morpheme gloss value. A match is found in Form 2, but it is partial because the morpheme gloss value of Form 2 is 'plrl' and not 'PL'. Therefore, 's'

in our new Form will be displayed as a green link (green to indicate a partial match) to Form 2 and 'PL' will not be displayed as a link.

5.1.5 5.1.5 morpheme gloss

The morpheme gloss field should contain a gloss in the metalanguage for each object language morpheme listed in the morpheme break field. The same delimiters should be used between the morpheme glosses as were used between the morphemes in the morpheme break line.

Researchers of an OLD application might want to work toward a consensus on how morphemes should be glossed.

Morpheme glosses will be displayed as links to matching Forms in the manner described above in the section on morpheme breaks.

5.1.6 5.1.6 gloss

The gloss is a translation of the Form into the metalanguage. When the Form represents a spatio-temporally located utterance, whenever possible the gloss should be something that the speaker offered, or would at least consent to, as a translation.

The OLD allows multiple (up to four) glosses for a single Form. Each gloss has its own gloss grammaticality field. This makes it possible to document a Form as compatible with certain glosses but not with others. For example, a form about the French word 'banque' might have 'bank (financial institutition)' as its first gloss and '*riverbank' as its second gloss.

5.1.7 5.1.7 gloss grammaticality

As discussed in the gloss section above, each gloss may have its own grammaticality. This grammaticality indicates the acceptability of the Form with a particular translation into the metalanguage. At present, the OLD allows three choices: compatible (''), incompatible ('*') and questionable ('?').

5.1.8 5.1.8 general comments

The general comments field is intended to contain notes pertaining to the Form in question. If you don't know where else to document something about a Form, enter it in the general comments field.

5.1.9 5.1.9 speaker comments

The speaker comments field is intended to contain quotations (or paraphrases) from the speaker of a particular Form. Often a comment from the speaker is not appropriate as a gloss yet it contains valuable information about some aspect of the Form.

5.1.10 5.1.10 elicitation method

The elicitation method refers to the means via which a particular Form was obtained. Often, for example, it is useful to know whether the speaker translated a metalanguage utterance of the elicitor, or described a visually represented context or judged the grammaticality of an object language utterance made by the elicitor, or whether the Form was obtained in some other manner.

The elicitation method field is a forced-choice user-populated field. That is, researchers must choose from a list of possible elicitation methods, but that list can be modified by researchers. By default there are no elicitation methods

predefined by an OLD application. Users must click on “database” in the primary menu and then “tags” in the secondary menu in order to add (or possibly update) the list of elicitation methods. (See the elicitation methods section). The intention behind forced-choice user-populated fields is to encourage intra- and inter-user consistency.

5.1.11 5.1.11 keywords

Keywords provide users with a general-purpose way of tagging Forms. A single Form may be associated to zero, one or many keywords. Keywords are defined by users of the OLD application in question. Click on “database” in the primary menu and then “tags” in the secondary menu to add new keywords.

5.1.12 5.1.12 category

The category refers to the syntactic or morphological category of the Form. Like elicitation method, category is a forced-choice user-populated field which is initially empty in a new OLD application. Researchers can add new categories (e.g., S, N, V, A, Adv, etc.) by clicking on “database” in the primary menu and “tags” in the secondary menu. (See the category section.)

5.1.13 5.1.13 category string

The category string is a string representing the morpho-syntactic categories of the morphemes within the Form. This string is generated by the system based on the morpheme break and morpheme gloss data entered by the user. For example, suppose that the following form has just been entered.

transcription	chiens
morpheme break	chien-s
morpheme gloss	dog-PL
gloss	dogs

Suppose further that when the system searches for the morpheme-gloss pairs ‘chien’-‘dog’ and ‘s’-‘PL’ it finds and exact match for each. In that scenario, the categories of the ‘chien’-‘dog’ and ‘s’-‘PL’ Forms (lets say they are ‘N’ and ‘Agr’) will be used to generate the category string of ‘chiens’ and the result will be ‘N-Agr’.

When an OLD application contains many Forms whose morpheme break and morpheme gloss fields are consistent with the system’s own lexical Forms, many category strings will be generated. When this is the case, users can search the category strings to reveal high-level morpho-syntactic patterns.

5.1.14 5.1.14 speaker

The speaker is the individual who uttered the object language token that the Form represents. To view the list of speakers documented in an OLD application, click on “database” in the primary menu and “people” in the secondary menu. Both administrators and contributors may add new speakers to the system (see the speaker section).

5.1.15 5.1.15 elicitor

The elicitor is the researcher who elicited the Form, that is, the person who recorded and/or transcribed the utterance of a speaker. Entering an elicitor involves choosing from a list of people registered as researchers for the OLD application in question. To view the list of registered researchers of an OLD application, click on “database” in the primary menu and “people” in the secondary menu.

5.1.16 5.1.16 enterer

The enterer field is automatically populated with the name of the OLD researcher who is adding the Form. In order to add a Form, a person must be logged in to the OLD application.

5.1.17 5.1.17 verifier

The verifier of a Form is another (perhaps more experienced) researcher who has already elicited a near-identical utterance and wants to indicate her agreement about the accuracy of the first researcher's documentation of that utterance. (The verifier field might be seldom used in practice...)

5.1.18 5.1.18 source

This category refers to the textual source of a Form, if applicable. Researchers can add new sources by clicking on "database" in the primary menu and "sources" in the secondary menu.

5.1.19 5.1.19 date elicited

The date when the Form was elicited (if applicable) is documented in the date elicited field in mm/dd/yyyy format.

5.1.20 5.1.20 date and time entered

The date and time when the Form was entered into the OLD application is automatically generated upon entry by the system.

5.1.21 5.1.21 date and time last modified

The date and time when the Form was last updated (modified) is automatically generated by the system during each update.

5.2 5.2 Adding a Form

To add a Form, click on "add" under "forms" in the side menu or use the keyboard shortcut "a".

Users can press the tab button to focus each of the fields in top-to-bottom, left-to-right order. If more than one gloss needs to be added, click on the "+" button to the right of the first visible gloss field. The only mandatory fields are the transcription and gloss fields. Once all necessary data has been entered, click the "Add Form" button at the bottom of the page.

Some general guidelines for entering Forms:

- in the transcription field, only use the characters (or character sequences) specified in the input orthography (plus standard punctuation). Remember, a user can specify her own input orthography and said orthography may differ from the system-wide default input orthography.
- in the morpheme break field, only use the characters (or character sequences) specified in the input orthography of your OLD application plus standard punctuation and morpheme delimiters. (This assumes your OLD application is treating morpheme break data as strings of the object language; see the XXX section for details.)

- try to be consistent in your transcriptions, your spelling of morphemes and your spelling of glosses, if not with the practice of other users, then at least with your own past practice; such consistency will help you (and others) to find quickly the data you need
- if you forget how a morpheme or gloss has been transcribed, open a Form search page in a new browser window/tab and search for your morpheme/gloss to find out past spellings (hint: it is often useful to have serveral OLD browser tabs open at once)

What to do if the database already contains a Form very similar to the one you are about to add:

- if your Form represents a recording of an utterance event, it is usually best to enter your Form despite the quasi-duplication it will cause
- if your Form represents an abstraction (e.g., a morpheme or word), then it is usually best to not enter a duplicate Form

5.3 5.3 Searching Forms

The OLD allows you to perform powerful searches on your Form data. The screenshot below shows the OLD Form search page.

Figure 5.1: The standard OLD Form search page

5.3.1 5.3.1 search expressions

Users can enter one or two search expressions and these expressions can be coordinated via conjunction ('and'), disjunction ('or') or negated conjunction ('and not').

Each search expressions is comprised of (i) a text input field (ii) a search type select field and (iii) a search location select field.

The text input field is where one enters the pattern that the result data must match.

The search type select field indicates the way in which the search is to be implemented. The search type options are 'as a phrase', 'all of these', 'any of these', 'as a reg exp' and 'exactly'. These will be discussed in more detail below.

The search location select field lists options for where (i.e., which column of the Form table) the system should look to match the pattern. The search location options are 'transcription', 'gloss', 'morpheme break', 'morpheme gloss', 'general comments', 'speaker comments', 'syntactic category string' and 'ID'.

5.3.2 5.3.2 order by expression

The order by expression allows one to state the order in which the matching results should be returned. One thing to note about the order by expression is that it will (probably) not order your results according to the order of graphs in the orthography specified in your OLD application settings. The order by expression uses the code points (in utf-8 encoding) of the characters in order to determine order. If your orthography contains characters outside of the 26

standard English ones, then the code points of those characters will likely be quite high and those characters will be understood by the system as coming after 'z'.

5.3.3 5.3.3 additional search filters

Additional search filters can be added to your search by clicking on the '+' button next to 'additional search filters'. Here one can further refine a Form search by putting conditions on the speaker, elicitor, enterer, verifier, source, grammaticality, gloss grammaticality, elicitation method, (syntactic) category, keywords, date elicited, date entered and/or date modified.

5.3.4 5.3.4 previous searches

One can repeat or make modifications to a previous search by clicking on the "previous searches" button at the bottom of the Form search page. The OLD keeps a record of each user's last ten searches and displays them when this button is clicked. When a past search is clicked, the search Form is returned with the appropriate fields set so that clicking "Search Forms" will repeat the search. This functionality is good for repeating searches as well as for making modifications to previous searches without having to re-enter all the search criteria.

The following subsections describe how to use each of the different search types available in a search expression.

5.3.5 5.3.5 as a phrase

The 'as a phrase' search type option causes the system to return all forms where the search location contains the specified search term as a substring. Thus, searching for the pattern 'the' as a phrase in the transcription field will return all Forms where the string 'the' is in the transcription, e.g., 'the dog', 'they left', 'another thing', etc.

5.3.6 5.3.6 all of these

The 'all of these' search type option causes the search pattern to be split by whitespace into sub-patterns and returns all Forms where the chosen location contains all of the sub-patterns. For example, searching for 'the and' with the 'all of these' search type option in the transcription field will return all Forms where both 'the' and 'and' are in the transcription, e.g., 'the sand', 'the cat and the dog', 'Mandy hit her brother', etc.

5.3.7 5.3.7 any of these

The 'any of these' search type option causes the search pattern to be split by whitespace into sub-patterns and returns all Forms where the chosen location contains any of the sub-patterns. For example, searching for 'the and' with the 'any of these' search type option in the transcription field will return all Forms where either 'the' and 'and' are in the transcription, e.g., 'the dog', 'Mandy ran', 'John and Mary smiled', 'other people', etc.

5.3.8 5.3.8 exactly

The 'exactly' search type option returns all Forms where the selected search location contains nothing but the search pattern. For example, searching for 'dog' in the transcription with 'exactly' as the search type option will return all Forms where the transcription value contains the string 'dog' and nothing else.

5.3.9 5.3.9 as a reg exp

The 'as a reg exp' search type option causes the search term to be interpreted as a regular expression. Regular expressions use a certain syntax which allows you to specify complex patterns. For example, using regular expressions one could search for the word 'the' and avoid matching 'other' or 'they' or one could search for strings that begin with 't'. Regular expression searches are very powerful but require learning a bit of the regular expression syntax. See the Regular Expressions section for more details.

5.3.10 5.3.10 searching and orthographies

The text input of a search expression whose location is transcription (or morpheme break, depending on the system settings) will be converted from the input orthography to the storage orthography before the query is performed.

5.4 5.4 Browsing Forms

There are two ways to browse all forms:

1. Click on "forms" in the side menu. This will return all Forms in the database, ordered by transcription ascending.
2. Enter an empty search, i.e., just go to the Form search page and hit enter or click on "Search Forms". This method allows you to change the default ordering

(You can also browse forms via the dictionary interface to the database (see the Dictionary section). However, the dictionary lists only those Forms whose transcriptions contain no white space.)

5.5 5.5 Other Form Actions

Beneath each Form are six buttons used to perform actions related to that Form: 'update', 'delete', 'history', 'associate', 'remember' and 'export'.

Figure 5.2: An example Form showing the six action buttons

5.5.1 5.5.1 update

Clicking on the update button brings up the update Form page which looks almost identical to the add Form page except that the fields are populated with values.

Before a Form is updated, it is first saved in a form_backup table of the database. This means that all versions of a Form are recorded. The history button (see below) allows one to view the previous versions of a Form.

5.5.2 5.5.2 delete

The delete button deletes a Form, i.e., removes it from the form table of the database. If one clicks the delete button, a confirm dialog appears in order to ensure that Forms are not mistakenly deleted.

As with the update action, before a Form is deleted it is saved in the form_backup table of the database. This means that a mistakenly deleted Form can be restored. (But this requires advanced knowledge of the system...)

5.5.3 5.5.3 history

The history button reveals the Form followed by its previous versions, ordered by newest to oldest. Thus, if a Form has been changed, one can see how it used to be and change it back if desired.

(It might be useful to have "revert" buttons next to each previous version. It might also be useful to use string comparisons and highlighting to illustrate the exact nature of the changes.)

5.5.4 5.5.4 associate

The associate button directs to the associate Form page which allows one to associate the Form to one or more Files by entering one or more comma-separated File IDs in a text input field. Associated Files appear beneath the Form data when the Form is displayed.

5.5.5 5.5.5 remember

Clicking the remember button causes the Form to be remembered in the current user's Memory. Memory is simply a user-specific ordered list of references to Forms. See the Memory section.

5.5.6 5.5.6 export

The export button brings up the export options page. This page provides a number of export types. At the time of writing, there are these 7:

1. transcription only (.txt)
2. transcription and gloss
3. interlinear gloss text
4. interlinear gloss text +
5. tab-delimited: everything
6. XeLaTeX IGT (Covington)
7. XeLaTeX IGT (Covington) +
8. XeLaTeX IGT (Covington) ++

Choosing an option and clicking "Export" will generate an export file and return the export complete page with a link for downloading your generated export file.

New export options can be created with a minimal amount of coding. Let the developer know what kind of export file you want to generate and he may try to implement it. (Better yet, study the lib/exporter.py module and write your own exporter object based on the examples therein – and don't forget to share.)

5.5.7 5.5.7 export all

When the results of a Form search are displayed, an 'export all' button appears in the upper left portion of the page. Clicking this button brings up the export page where one can choose the export type (see above). Export all works just like export except that in this case the output can contain data from more than one Form.

5.5.8 5.5.8 remember all

When the results of a Form search are displayed, a 'remember all' button appears in the upper left portion of the page. Clicking this button will put all the search results into the current user's memory (see Memory)

CHAPTER

SIX

6 FILES

An OLD File is a grouping of data about a digital file. The file is uploaded to an OLD application and data about the file (such as its name, type and size) are inferred in the process. Other data (such as a description) may be entered by the user.

6.1 6.1 File Data

This section describes the type of data that comprise an OLD File.

6.1.1 6.1.1 ID

The ID of a File is a unique integer automatically generated by the database. Knowing the ID of a File can be useful when you want to search for it, associate it to a Form or Collection or when you want to quickly view a certain file (just enter enter "domain_name/file/view/1" in your browser's address bar to view the File with ID 1).

6.1.2 6.1.2 name

The name of the uploaded file becomes the name of the OLD File. If the system already contains a file with that name, a digit is appended to the file name and incrememted until a unique name is arrived at.

6.1.3 6.1.3 MIME type

The MIME type of an OLD File is inferred from the a uploaded file. The MIME type is a string that indicates whether the file is text, an image, a video, a sound file, a PDF, a Word document, etc. For more details, see the MIME type Wikipedia entry

6.1.4 6.1.4 size

The size of the OLD File is the size of the uploaded file.

6.1.5 6.1.5 enterer

The enterer is automatically added as a reference to the user who entered the File.

6.1.6 6.1.6 description

The description is a general-purpose string of text entered by the user. The description might contain information about what happens on an audio/video recording or what a certain audio/video/image stimulus was designed to elicit.

6.1.7 6.1.7 speaker

If a File is about a recording of a linguistic event, the speaker field can be used to indicate who was speaking the object language during the recording.

(Functionality should probably exist to associate more than one speaker to a File...)

6.1.8 6.1.8 elicitor

If a File is about a recording of a linguistic event, the elicitor field can be used to indicate the researcher who was eliciting during the recording.

6.1.9 6.1.9 date elicited

If a File is about a recording of a linguistic event, the date when the recording was made can be entered in the date elicited field in mm/dd/yyyy format.

6.1.10 6.1.10 utterance type

If a File is about a recording of a linguistic event, then one can choose one of the following four options for utterance type:

1. None: the type of utterance is left unspecified (the default)
2. Object Language Utterance: the recording contains only utterances of the object language
3. Metalanguage Utterance: the recording contains only utterances of the metalangauge
4. Mixed Utterance: the recording contains utterances of both the object language and metalanguage

(Choosing the appropriate utterance type for a File representing a recording will be useful for certain planned OLD features, e.g., language learning games that might play a recording of an object language utterance and have players try to guess the meaning.)

6.1.11 6.1.11 date and time entered

When a File is added, the system automatically records the date and time the File was entered.

6.1.12 6.1.12 date and time modified

When a File is updated, the system automatically records the date and time in the date and time modified field.

6.2 6.2 Adding a File

To add a File, go to 'add' under 'files' in the side menu (or use the keyboard shortcut 'q'). Adding a File requires that you upload a digital file to the OLD application. Just click on the 'browse' button and choose a file from your file system. After choosing the correct file, click the 'Add File' button at the bottom of the page.

One may also optionally add a description, specify a speaker, an elicitor, a date elicited and/or an utterance type.

An OLD application will accept for upload digital files with the following MIME types:

- text/plain (plain text)
- application/x-latex (LaTeX document)
- application/msword (MS Word document)
- application/vnd.oasis.opendocument.text (Open Document Format)
- application/pdf (PDF)
- image/gif
- image/jpeg
- image/png
- audio/mpeg
- audio/ogg
- audio/x-wav
- video/mpeg
- video/mp4
- video/ogg
- video/quicktime
- video/x-ms-wmv

6.3 6.3 Searching Files

The File search page is very similar to the Form search page (see Searching Forms). Note the following differences:

- Files and Forms have different search location options. Currently one can search Files by name, description and ID.
- Files do not have previous searches functionality
- Files have different additional search filters; note in particular the MIME type (see MIME type) and size filters

To search for a File, click on 'search' under 'files' in the side menu or use the keyboard shortcut 'w'.

6.4 6.4 Browsing Files

To browse all Files, click on 'files' in the side menu or use the keyboard shortcut 'r'. To browse all Files in a particular order, perform an empty File search and alter the order by expression as appropriate.

6.5 6.5 Other File Actions

When a File is displayed, three action buttons appear at the bottom: ‘update’, ‘delete’ and ‘associate’.

Figure 6.1: An example File showing the three action buttons

6.5.1 6.5.1 update

Clicking on the update button displays the File update page which looks very similar to the File add page except that some of the fields may contain data and the uploaded digital file cannot be changed. Make changes and click ‘Save Changes’.

6.5.2 6.5.2 delete

The delete button will both delete the data about the File from the database as well as remove the appropriate digital file form the OLD application’s ‘files’ directory. A confirm dialog appears in order to ensure that Files are not mistakenly deleted.

6.5.3 6.5.3 associate

The associate button brings up the associate File page which allows one to associate the File to one or more Forms. Just enter the IDs (comma-separted) of all Forms that should be associated to the File in question.

6.5.4 6.5.4 export all

When the File search results are displayed, there is a button near the top left of the page labelled ‘export all’. Clicking on this button will create an archived folder (.zip) containing the digital files corresponding to the result Files. The page displayed will contain a link to the archived folder and a listing of its contents. Click on the link to download the archived folder.

CHAPTER

SEVEN

7 COLLECTIONS

Collections represent stories, records of elicitations, conversations or any other multi-sentence discourse. At its core, a Collection is a set of ordered references to OLD Forms. A Collection may also be associated to zero or more Files (e.g., recordings of an entire story or elicitation session).

7.1 7.1 Collection Data

7.1.1 7.1.1 ID

The ID is a unique integer assigned by the RDBMS to each Collection upon creation. Knowing the ID of a Collection comes in handy when you want to quickly access a Collection: enter 'domain_name/collection/view/1' to view the Collection with ID 1.

7.1.2 7.1.2 title

The title is a name for your Collection. This field is mandatory.

7.1.3 7.1.3 type

The type indicates what the Collection represents. At present there are four possible Collection types: story, discourse, elicitation and other.

7.1.4 7.1.4 description

The description contains general-purpose textual data describing the Collection.

7.1.5 7.1.5 speaker

If the Collection has a speaker, use this field to reference an OLD speaker for the Collection.

7.1.6 7.1.6 elicitor

The elicitor field references the name of the researcher who elicited the Collection, if appropriate.

7.1.7 7.1.7 source

If the Collection is from a textual source, one should use the source field to make the reference.

7.1.8 7.1.8 date elicited

The date the Collection was elicited in mm/dd/yyyy format.

7.1.9 7.1.9 content

The content of the Collection is a string of text. That string can contain the following types of markup:

- OLD Form Embed Markup
- OLD Entity Link Markup
- reStructuredText

Consider the following example Collection content:

```
Elicitation About Dogs
======================

The following sentence shows A, B and C.

form[1]

The above sentence was extracted from the audio recording in file(7)

The following sentence shows X and Y.

form[2]

Compare this discourse to that in collection(2)

The end.
```

The line underlined with '='s will be displayed as a level-one HTML header and lines of text surrounded by blank lines will be displayed as HTML paragraphs (see reStructuredText). Depending on the data in Forms 1 and 2, this content will be rendered similar to below:

Figure 7.1: Illustration of how the content of a Collection is rendered

Notice the following: 'form[71]' and 'form[72]' generate representations of the appropriate forms; '# Elicitation About Dogs' has been rendered as a header; and 'file(7)' and 'collection(2)' have generated as links to the File 7 and Collection 2 respectively.

When adding content to (or altering the content of) a Collection, there is a button labelled 'insert contents of memory'. Clicking this button will append to the content text a list of references to each Form in the user's memory. It is often convenient to gather relevant Forms into memory and then save them as a Collection using this functionality.

7.1.10 7.1.10 associated files

Collections can be associated to zero or more Files. To associate Files to a Collection, use the 'associate' button that appears when a Collection is displayed.

7.1.11 7.1.11 associated forms

Collections can be associated to zero or more Forms. The Forms associated to a particular Collection are those referenced in the content field of the Collection.

7.2 7.2 Viewing a Collection

The content of a Collection can be displayed in three ways: long, short and columns. Long view displays the embedded Forms of a Collection's content just as Forms are displayed by default, i.e., in interlinear gloss format with all of their data visible. Short view displays only the transcription and glosses of embedded Forms. Columns view is different from long and short in that the embedded Forms are displayed but any prose or other markup is not displayed. Columns view lists the Forms in a two-column table with the transcription on the left and glosses on the right.

7.3 7.3 Adding a Collection

To add a Collection, click 'add' under 'collections' in the side menu or use the keyboard shortcut 'z'.

The only obligatory field in a Collection is the title. Enter the data as necessary and format the Collection content as described above. Then click "Add Collection".

7.4 7.4 Searching Collections

To search Collections, click 'search' under 'collections' in the side menu or use the keyboard shortcut 'x'.

Searching Collections is nearly identical to the process of searching Forms and Files. Refer to the Searching Forms section for more details.

7.5 7.5 Browsing Collections

To browse all Collections, click 'collections' in the side menu or use the keyboard shortcut 'c'.

7.6 7.6 Other Collection Actions

When a Collection is displayed, three buttons appear: 'update', 'delete' and 'associate'.

7.6.1 7.6.1 update

The update button directs to the update Collection page which allows users to alter the Collection's data. Unlike with Forms, there is currently no collection_backup table to store previous versions of Collections.

7.6.2 7.6.2 delete

The delete button destroys the Collection. Unlike when deleting Forms, deleted Collections are not backed up in a collection_backup table.

7.6.3 7.6.3 associate

The associate button allows users to associate one or more Files to the Collection by entering a list of one or more comma-delimited File IDs.

CHAPTER

EIGHT

8 PEOPLE

The ‘people’ section lists information about the OLD entities representing individuals, i.e., speakers and users. To view the people of an OLD application, click ‘database’ in the primary menu and ‘people’ in the secondary menu (or use the keyboard shortcut ‘p’).

8.1 8.1 Speakers

Speakers are the consultants/colleagues who speak the object language and have been kind enough to share their knowledge with the researchers who study and document their language.

In order to document the speaker of a Form, File or Collection, that speaker must first be represented as an OLD speaker entity. The addition (or updating) of a speaker immediately results in that speaker being available in the speaker fields of Forms, Files and Collections.

8.1.1 8.1.1 view a speaker

To view a speaker, click on the speaker’s name in the speakers table of the people page.

8.1.2 8.1.2 add a speaker

Users (administrators and contributors) can add a new speaker by clicking the ‘add’ button. A new speaker requires at least a first name and a last name.

If the object language has dialects, specifying the dialect spoken by the speaker may be useful when other researchers are trying to interpret your data.

The page content section contains the text of the speaker’s OLD web page. reStructuredText syntax can be used to format the speaker’s web page. The page content might include detailed information about the speaker’s biography and proficiency with the object language.

8.1.3 8.1.3 update a speaker

To update the data about a speaker, click on the speaker’s name in the speakers table of the people page and click the ‘edit’ button.

8.1.4 8.1.4 delete a speaker

To delete a speaker, click on the speaker's name in the speakers table of the people page and click the 'delete' button. A confirmation dialog will arise to ensure that speakers are not mistakenly deleted. As a speaker may be referenced by multiple OLD entities (some of them possibly entered by other users), consider carefully the possible ramifications before deleting a speaker.

8.2 8.2 Users

Users are the individuals who are authorized to access a particular OLD application. The list of users determines the individuals available as enterers, elicitors and verifiers when adding Forms, Files and Collections.

There are three types of users with different levels of authorization: administrators, contributors and viewers. The permissions of these user types are listed below

8.2.1 8.2.1 administrators

Administrators are authorized to perform the following actions:

- view and update application settings
- view, add, update and delete users
- view, add, update and delete Forms, Files and Collections
- view, add, update and delete speakers, tags and sources

8.2.2 8.2.2 contributors

Contributors contribute data to an OLD application. They are authorized to perform the following actions:

- update their own user information
- view other users' pages
- view, add, update and delete Forms, Files and Collections
- view, add, update and delete speakers, tags and sources

8.2.3 8.2.3 viewers

Viewers can view data in an OLD application. They are authorized to perform the following actions:

- update their own user information
- view other users' pages
- view Forms, Files and Collections
- view speakers, tags and sources

8.2.4 8.2.4 view a user

To view a user, click on that user's name in the users table of the people page.

8.2.5 8.2.5 add a user

Only administrators can add users. To add a user, click the add button in the users section of the people page.

New users require a username, a password, a first name, a last name, an email and a role, i.e., administrator, contributor or viewer.

If appropriate, enter an affiliation, i.e., the name of a university, college, school, community, organization, etc., with which the user is affiliated.

The personal page content is a string of text that comprises the OLD page of the user. The personal page content can contain the following types of markup:

- OLD Entity Link Markup
- OLD File file Markup
- OLD Image Embed Markup
- reStructuredText

8.2.6 8.2.6 edit a user

Administrators can edit users and users can edit their own user data. To edit a user, click the user's name in the users table of the people page and then click the edit button.

8.2.7 8.2.7 delete a user

Only administrators can delete users. To delete a user, click the user's name in the users table of the people page and then click the delete button. A confirm dialog will be displayed so that users are not mistakenly deleted.

8.2.8 8.2.8 user settings

Each user has their own settings page. To view your settings, click on your name in the people page and then click the settings button. At present, one can set a default view type for Collection contents (see) as well as a mode (choice of database or dictionary). (I don't think the mode setting actually has any effect right now. Make it do something or remove it!)

CHAPTER

NINE

9 TAGS

Tags are OLD entities which can be used to tag Forms and which are user-defined. At present, there are three types of tag: keywords, syntactic categories and elicitation methods. Users of a particular OLD application can add new tags. To view the tags, click on 'database' in the primary menu and 'tags' in the secondary menu (or use the keyboard shortcut 't').

9.1 9.1 Keywords

Keywords are general-purpose tags that users can define. A single form can be associated to multiple keywords. Some examples of potential keywords: 'SVO', 'passive', 'weak quantifier', etc.

9.1.1 9.1.1 add a keyword

To add a new keyword, click the add button under the keywords heading of the tags page. A keyword requires a name. The name is what will be listed in the keywords field when adding/updating a Form. Use the optional description field to describe the keyword in detail.

9.1.2 9.1.2 edit a keyword

To edit a keyword, click the name of the keyword in the keywords table of the tags page and then click edit.

9.1.3 9.1.3 delete a keyword

To delete a keyword, click the name of the keyword in the keywords table of the tags page and then click delete. A confirm dialog will appear to discourage mistakenly deleting a keyword. Use caution when deleting a keyword as doing so will effect Forms associated to that keyword.

9.2 9.2 Syntactic Categories

Syntactic categories are user-definable tags that are intended to be used to indicate the morphological or syntactic category of the Form. A Form can only have one syntactic category. Some examples of possible syntactic categories: 'S', 'N', 'V', 'Asp', 'Root', 'Agr', etc.

9.2.1 9.2.1 add a syntactic category

To add a new syntactic category, click the add button under the syntactic categories heading of the tags page. A syntactic category requires a name. The name is what will be listed in the category field when adding/updating a Form. Use the optional description field to describe the syntactic category in detail.

9.2.2 9.2.2 edit a syntactic category

To edit a syntactic category, click the name of the syntactic category in the syntactic categories table of the tags page and then click edit.

9.2.3 9.2.3 delete a syntactic category

To delete a syntactic category, click the name of the syntactic category in the syntactic categories table of the tags page and then click delete. A confirm dialog will appear to discourage mistakenly deleting a syntactic category. Use caution when deleting a syntactic category as doing so will effect Forms associated to that syntactic category.

9.3 9.3 Elicitation Methods

Elicitation methods are user-definable tags that are intended to be used to indicate the manner in which the Form was elicited from a speaker. A Form can only have one elicitation method. Some examples of possible elicitation methods: 'volunteered form', 'used image stimulus', 'part of a story', etc.

9.3.1 9.3.1 add an elicitation method

To add a new elicitation method, click the add button under the elicitation methods heading of the tags page. An elicitation method requires a name. The name is what will be listed in the elicitation method field when adding/updating a Form. Use the optional description field to describe the elicitation method in detail.

9.3.2 9.3.2 edit an elicitation method

To edit an elicitation method, click the name of the elicitation method in the elicitation methods table of the tags page and then click edit.

9.3.3 9.3.3 delete a syntactic category

To delete an elicitation method, click the name of the elicitation method in the elicitation methods table of the tags page and then click delete. A confirm dialog will appear to discourage mistakenly deleting an elicitation method. Use caution when deleting an elicitation method as doing so will effect Forms associated to that elicitation method.

CHAPTER

TEN

10 SOURCES

A source is an OLD entity representing a textual source, e.g., a book, a dictionary, a paper, etc. Both Forms and Collections can have sources. To view the sources of an OLD application, click ‘database’ in the primary menu and then ‘sources’ in the secondary menu.

10.1 10.1 Add a Source

To add a source, click the add button in the sources page. Sources require the following data: author first name, author last name, year of publication and title. Optional data include a full bibliographic reference for the source and the ID of an OLD File representing a digital copy of the source.

10.2 10.2 Edit a Source

To edit a source, click the name of the source in the sources table of the sources page and then click the edit button.

10.3 10.3 Delete a Source

To delete a source, click the name of the source in the sources table of the sources page and then click the delete button. A confirm dialog will appear in order to avoid mistakenly deleting a source. Use caution when deleting a source since it may be referenced by Forms and Collections.

CHAPTER

ELEVEN

11 MEMORY

Each OLD application user has her own private memory. Memory is simply an ordered list of OLD Forms that a particular user has chosen to remember. To view your memory, click on ‘database’ in the primary menu and then ‘memory’ in the secondary menu (or use the ‘m’ keyboard shortcut).

At the top of the memory page are two buttons: ‘forget all’ and ‘export all’. Each Form in memory is displayed with its ID, transcription, morpheme break, morpheme gloss and gloss values. Beneath the Form data are three buttons: ‘view’, ‘export’ and ‘forget’

11.1 11.1 forget all

The forget all button removes all Forms from memory.

11.2 11.2 export all

The export all button works just like the export all button that is displayed with Form search results. That is, clicking on this button will bring up the Form export page from where a user may choose an export type, click ‘export’ and recieve their memorized Forms in the chosen format.

CHAPTER

TWELVE

12 DICTIONARY

Clicking on ‘dictionary’ in the primary menu (or using the ‘d’ keyboard shortcut) brings up a dictionary-like interface to the Forms in the database. The dictionary excludes all Forms whose transcription contains a whitespace character and in so doing makes a stab at getting only words. The Forms listed in the dictionary are sorted according to the order of graphs in the object language orthography as specified in the OLD application’s settings.

12.1 12.1 Browse

To browse the dictionary, click ‘browse’ in the secondary menu. The dictionary browse page displays the object language name, a table of ordered graphs from the object language orthography, the metalanguage name, and a table of ordered graphs from the metalanguage orthography.

12.1.1 12.1.1 object language to metalanguage

Clicking on the object language name will display all of the entries of the dictionary in object-language-to-metalanguage format. These entries will be ordered according to the order of graphs in the object language orthography.

Clicking on an object language graph will display all dictionary entries that begin with that graph. These entries will be ordered according to the order of graphs in the object language orthography.

12.1.2 12.1.2 metalanguage to object language

Clicking on the metalanguage name will display all of the entries of the dictionary in metalanguage-to-object-language format. These entries will be ordered according to the order of graphs in the metalanguage orthography.

Clicking on a metalanguage graph will display all dictionary entries that begin with that graph. These entries will be ordered according to the order of graphs in the metalanguage orthography.

12.2 12.2 Search

To search the dictionary, click ‘search’ in the secondary menu. The dictionary search page displays a single text input field and a dropdown menu where one can choose to search the object language or the metalanguage.

12.2.1 12.2.1 object language to metalanguage

If one enters a search term, chooses 'object language to metalanguage' from the dropdown menu and clicks 'Search', the system will return all Forms with a transcription which matches the search term and which lacks whitespace. The results displayed will be ordered according to the ordering of graphs in the object language orthography.

12.2.2 12.2.2 metalanguage to object language

If one enters a search term, chooses 'metalanguage to object language' from the dropdown menu and clicks 'Search', the system will return all Forms with a gloss which matches the search term and with a transcription which lacks whitespace. The results displayed will be ordered according to the ordering of graphs in the metalanguage orthography.

CHAPTER

THIRTEEN

13 APPLICATION SETTINGS

The application settings are the global settings for an OLD application. All users can view the application settings but only administrators can alter them. To view the application settings, click 'settings' in the primary menu.

13.1 13.1 Object Language

The object language is the language that is being studied and documented with the help of a particular OLD application.

Administrators should specify a name for the object language. This name will be used throughout the application to refer to the object language.

Administrators can also specify the ISO 639-3 three-letter code for the language so that it can be unambiguously identified. This is especially important if the chosen object language name is one of many variants. The OLD has all the ISO 639-3 codes stored. When one begins to type in the ISO 639-3 Code text input field, suggestions of valid codes appear along with the standard ('reference') name of the corresponding language. If the ISO 639-3 code cannot be found, try searching the Ethnologue web site.

13.2 13.2 Storage, Input & Output

It is possible that an object language will have multiple orthographies in use. In such a case, it would be nice to permit each user to interact with an OLD application using the orthography of their choice. On the other hand, it is also desirable that all object language data be stored in the same orthography. To resolve this conflict, the OLD facilitates automatic translation of object language strings from one orthography to another.

For this reason, an OLD application requires the specification of three types of object language orthography:

1. a storage orthography
2. a default input orthography
3. a default output orthography

The storage orthography is immutable, but the input and output orthographies are 'default' because each user can override these settings in their user-specific settings. Henceforward, 'input orthography' will refer to the user-specific input orthography if specified, or the system-wide default otherwise. Same thing, mutatis mutandis, for 'output orthography'.

The storage orthography is what input strings will be converted into for storage and output strings converted from for display. Object language input from the user will be converted from the input orthography to the storage orthography. Object language output will be converted from the storage orthography to the output orthography for display.

The simplest setup (and the recommended one) is to specify a single orthography for storage, input and output. This will result in no conversions being made when object language data are entered or displayed. That way, whenever a user wants to view or export (or enter) their data in a different orthography, they can alter their user-specific settings as they please.

In special circumstances, it may be desirable to set the default input and default output orthographies to orthography X and the storage orthography to orthography Y. (For example, X may be common and familiar to users yet Y is chosen for storage because it is more expressive.) With this setup, the OLD application will handle all orthographic conversions and users using the system defaults will not necessarily know (or need to know) about the existence of the distinct storage orthography Y.

13.3 13.3 Object Language Orthography(/ies)

An object language orthography is an ordered list of graphs (characters or character sequences) for representing object language data. Depending on how things are set up, an OLD application can be configured to automatically transform object language data from one orthography to another (see Storage, Input & Output for details).

Administrators should specify at least one object language orthography. As discussed above (Storage, Input & Output), the simplest setup is to specify a single orthography and make that orthography be the storage, default input and default output orthography.

If multiple object language orthographies are defined, then all orthographies must have the same structure and the same number of graphs. The OLD will not allow multiple object language orthographies to be defined if these requirements are not met.

13.4 13.4 Object Language Data

Certain data types are assumed by the OLD to contain nothing but strings of graphs from the input orthography; these I call 'pure object language fields'. Certain data types are optionally assumed to contain nothing but input orthography strings; these I call 'optionally object language fields'. Finally, some data types can contain object language tags ('<obl></obl>') and the data withing such tags will be considered object language data; these fields I call 'object language tag fields'.

13.4.1 13.4.1 Pure Object Language Fields

When entering data into, or when searching for data in, the pure object language fields listed below, the input is converted from the input orthography to the storage orthography. When the data from such fields are displayed, they are converted from the storage orthography to the output orthography. (Also, when data from these fields are displayed for updating, the data are converted from the storage orthography to the input orthography.)

The following fields are pure object language fields:

- transcriptions of Forms

13.4.2 13.4.2 Optionally Object Language Fields

The following fields will be treated like pure object language fields if so specified in the application settings. To alter these settings, click on 'settings' in the primary menu, click the edit button and scroll down to the 'object language strings' fieldset. In the dropdown menu, choose 'yes' to treat morpheme breaks as object language fields or 'no' not to.

- morpheme breaks of Forms

13.4.3 13.4.3 Object Language Tag Fields

The fields listed below may contain substrings enclosed in object language tags, i.e., '<obl>' to the left and '</obl>' to the right. These substrings will be treated like object language data for the purposes of orthographic conversion, as described in the *Pure Object Language Fields* section above.

- general comments of Forms
- speaker comments of Forms

13.5 13.5 Testing Orthography Conversion

To test how well the OLD converts from one orthography to another, go to the settings orthography page by clicking on 'settings' in the primary menu, then scroll down to the bottom of the 'Storage, Input & Output' section and click the button labeled 'More on Orthographies'.

This page allows users to do two things:

1. experiment with converting strings from one orthography to another
2. see whether they are able to enter the graphs of a particular orthography

This page is pretty self-explanatory.

13.6 13.6 Metalanguage

Administrators should specify the name, ISO 639-3 code and orthography of the metalanguage as well. The metalanguage name will be used throughout the system to refer to the metalanguage and the metalanguage orthography will be used in the dictionary interface to the database.

CHAPTER

FOURTEEN

14 CHARACTER ENCODING

OLD applications handle all strings as unicode. This means that users should have no problems using obscure characters when entering data.

CHAPTER

FIFTEEN

15 FONTS

An OLD application may require the installation of certain fonts in order to display characters properly. If applicable, instructions for acquiring and installing this font should be provided in the help page of this OLD application (i.e., click on ‘help’ in the primary menu and then ‘help page’ in the secondary menu).

CHAPTER

SIXTEEN

16 REGULAR EXPRESSIONS

Regular expressions allow one to specify complex patterns when searching Forms, Files or Collections. I will not attempt to provide a complete tutorial on how to write regular expressions. There are many online resources devoted to that purpose. Some places to start looking:

- wikipedia page on regular expressions
- gnosis.cx/publish/programming/regular_expressions.html

16.1 16.1 Metacharacters

In the regular expression syntax, certain characters have special meanings. These characters are called metacharacters. In this section, I briefly review the meanings of some of these characters.

. Matches any single character. E.g., d.g matches 'dog', 'dig', 'dkg', etc. but not 'doog'

[] Matches any of the characters in the brackets. E.g., d[iou]g matches 'dog', 'dig' and 'dug' and nothing else

[^] Matches any character that is not in the brackets. E.g., d[^ae]g matches 'dog', 'dig', 'dug', 'dzg', etc. but not 'dag', 'deg' or 'doog'

^ Matches the beginning of the string. E.g., ^d matches 'dog' and 'dare him!' but not ' dog' or 'adrift'

**$** Matches the end of the string. E.g., k$ matches 'back' and 'he will walk' but not 'back.', 'he walks' or 'kid'

***** Matches the preceding element zero or more times. E.g., a*gh matches 'aaaaaaaaaagh', 'he yelled "aaagh"' and 'light' but not 'aaaagghh'. (The phrase 'preceding element' is used to include subexpressions as well as characters. E.g., [ae]*gh matches 'aagh!' or 'eegh!')

+ Matches the preceding element one or more times. E.g., a+gh matches 'aaaaaaaaagh' and 'he yelled "aaagh"' but not 'light'

? Matches the preceding element zero or one time. E.g., bee?t matches 'bet' and 'beet' and nothing else

{m,n} Matches the preceding element at least m and no more than n times. E.g., b[eo]{1,2}t matches 'bet', 'beet', 'bot', 'boot', 'beot', 'boet' and nothing else

{m} Matches the preceding element exactly m times

{m,} Matches the preceding element at least m times

{,m} Matches the preceding element no more than m times

| Matches the expression before the vertical bar or after it. E.g., d(i|o|u)g matches the exact same set of strings as d[iou]g does (see above). E.g., using parentheses to group subexpressions, ^((talk)|(speak))s$ matches 'talks' and 'speaks' but nothing else

16.2 16.2 Some Useful Regular Expressions

(^|[’”])word($|[’”]) Matches the word ‘word’. I.e., matches ‘a bad word’ and “‘word” was what I said’ without matching ‘words are interesting’, ‘she is wordy’ or ‘sword’, etc.

(^|[-=])mor($|[-=]) Matches the morpheme ‘mor’. I.e., matches ‘a-mor=b’ and ‘a=mor’ without matching ‘a-morp-b’ or ‘kmor=b’, etc.

CHAPTER

SEVENTEEN

17 OLD MARKUP

The OLD recognizes certain strings of characters as markup. Such strings, depending on where they occur, will be displayed in special ways. For example, the string 'form[23]' in the content of a Collection will be displayed as a formatted version of the OLD Form with ID 23.

17.1 17.1 OLD Form Embed Markup

An expression of the form 'form[x]', where 'x' is the ID of a Form, will be recognized as Form embed markup. Form embed markup is recognized in the content field of a Collection. The expression 'form[x]' in the content of a Collection will be displayed as a formatted representation of the primary data of the Form with ID x.

In addition, the expression 'form[x]' will cause Form x to be listed as one of the Forms that are associated to the Collection in question.

17.2 17.2 OLD Entity Link Markup

An expression of the form 'entity(x)', where 'entity' is the name (in lowercase) of an OLD entity (e.g., 'form', 'file', 'collection', 'speaker', 'user') and 'x' is the ID of such an entity, will be recognized as Entity link markup. Entity link markup creates HTML links to OLD entities. That is, expressions of the form 'form(22)' 'file(3102)' and 'collection(77)' will be displayed as HTML links to Form 22, File 3102 and Collection 77 respectively. Entity link markup is recognized in the content of a Collection and in the page content of users.

17.3 17.3 OLD File file Markup

An expression of the form '[x]link(y)', where 'x' is a string of text and 'y' is the name of an OLD File, will be recognized as File file markup. File file markup creates an HTML link to the digital file of an OLD File. That is, the expression '[interesting audio]link(elicitation_file.wav)' will be displayed as a link consisting of the words 'interesting audio' and which points to the audio file elicitation_file.wav. File file markup is recognized in the page content of users.

17.4 17.4 OLD Image Embed Markup

An expression of the form 'image(x)', where 'x' is the name of an OLD image File, will be recognized as image embed markup. Image embed markup embeds the appropriate image in the text. That is, the expression 'image(stimulus_photo.jpg)' will cause the image of the File whose name is 'stimulus_photo.jpg' to be displayed. Image embed markup is recognized in the page content of users.

17.5 17.5 OLD Metalanguage Markup

The OLD expects certain fields (e.g., Form transcriptions and optionally morpheme breaks) to contain data in the object language orthography. Depending on how the OLD application is set up, graphs of this orthography may be displayed differently from how they are entered (see Storage, Input & Output above). This can cause undesired character transformations if metalanguage strings (e.g., names, borrowings) are entered in object language fields. To avoid this potential complication, the OLD permits the use of metalanguage markup. Expressions of the form '<ml>string</ml>' will be interpreted as metalanguage strings and the data between the '<ml>' and '</ml>' tags will not be treated as object language data, i.e., no orthographic conversions will be applied.

For example, an OLD application might be set up to display the object language string 'ts' as 'c'. But if a Form transcription contains the English name 'Keats', it will be erroneously displayed as 'Keac'. To avoid this, simply enter the metalanguage (in this case, English) string as '<ml>Keats</ml>'. This will then be displayed as 'Keats', as desired.

CHAPTER

EIGHTEEN

18 RESTRUCTUREDTEXT

reStructuredText is a human-readable markup syntax that can be converted to HTML (as well as to LaTeX and other formats). For example, the string:

```
This is a Header
================
```

might be converted to '<h1>This is a Header</h1>' and displayed by a browser as a formatted top-level header. A good quick reference for reStructuredText syntax, is the Quick reStructuredText web page.

reStructuredText is the markup language of choice for creating OLD pages because it can be converted to a variety of formats (i.e., HTML, LaTeX, ODT).

CHAPTER

NINETEEN

19 MARKDOWN

Markdown is a human-readable markup syntax that can be displayed as HTML. For example, the string '# This is a Header' will be displayed as '<h1>This is a Header</h1>' which in turn will be displayed by the browser as a formatted top-level header. For more details on Markdown syntax, visit the Markdown syntax web page.

CHAPTER

TWENTY

20 KEYBOARD SHORTCUTS

HTML allows the use of keyboard shortcuts on links (anchors) via the “accesskey” attribute. The OLD uses accesskeys to create keyboard shortcuts for commonly used actions. Hold your cursor over an action link to see a description of what the link does. If that description is terminated by “<x>”, then “x” is the keyboard shortcut.

Accesskeys are implemented differently depending on the operating system and browser being used. With FireFox on a Mac, use Control + Key. With Safari or Chrome on a Mac, use Control + Option + Key.

List of keyboard shortcuts:

f browse all Forms

s search Forms

a add a Form

c browse all Collections

x search Collections

z add a Collection

r browse all Files

w search Files

q add a File

l login or logout

p view the people page

t view the tags page

m view the contents of one’s memory

CHAPTER

TWENTYONE

21 BROWSER SUPPORT

Web browsers do not render HTML, CSS and Javascript in a uniformly consistent manner. Therefore, parts of the OLD may not appear as intended, depending on the OS and browser.

The OLD works on a Mac (OS 10.6) with FireFox, Safari and Chrome. It has also been tested on Ubuntu with FireFox without any apparent problems. It has not yet been tested with Internet Explorer on a PC.

CHAPTER

TWENTYTWO

22 GLOSSARY OF TERMS

metalanguage The language used to gloss and otherwise document the object language.

OLD The Online Linguistic Database; a piece of software that can be used to create OLD applications tailored to the documentation of particular languages.

OLD application A language-specific OLD instance. For example, an OLD application used to study Esperanto might be called the Esperanto Online Linguistic Database.

object language The language under study. The language that is being documented with the help of a particular OLD application.

RDBMS Relational Database Management System. The database software, i.e., the thing that allows us to store, retrieve and alter data. An OLD application may be configured to use the following RDBMSs: MySQL, SQLite or PostgreSQL.

CHAPTER

TWENTYTHREE

23 FUTURE FEATURES

23.1 23.1 Priorities

- administrators/users should be able to alter/add to the list of possible grammaticalities/glossGrammaticalities and possible morpheme delimiters
- offer further export formats (sql, odt) (docutils offers odt creation!)
- clean up code (docstrings in controllers, make more readable, etc.)
- finish documentation (developer guide, api, user guide)
- is <ml></ml> markup available in the morpheme break field? If not, it should be
- test the OLD on a PC with various browsers and update the OLD User Guide

23.2 23.2 Relatively Easy Projects

- when entering data, certain values from past additions should be made default, i.e., elicitor and syntactic category are often repeatedly the same person when a user is entering multiple forms
- currently all searches are case insensitive (should this be changed?...)
- ability to search Forms by association to Files and vice versa
- allow the file of a File to be hosted on another server, e.g., use the embeddedFileMarkup column of the file table
- make forms searchable by dialect (or is this redundant since one can already search by (sets of) speaker(s)?)
- create 'forgot my password' functionality (and forgot my username functionality too?)
- I think an administrator might be able to delete her own user account. This should be prohibited

23.3 23.3 Big Projects

- integrate a general-purpose morphological parser (SFST, pySFST)
- ability to indicate vowel and consonant classes on orthographies
- warnings or errors when orthographically invalid transcriptions/morpheme breaks are entered
- orthography-specific ordering should be available via the database interface
- forums/discussion functionality

- create functionality that will allow users to create an arbitrary number of web pages (worth it?)

CHAPTER

TWENTYFOUR

DEVELOPER GUIDE

This software is documented in detail in the SimpleSite tutorial chapters of the book *The Definitive Guide to Pylons* available under an open source license at http://pylonsbook.com. You should read those chapters to discover how SimpleSite is developed.

CHAPTER

TWENTYFIVE

API DOCUMENTATION

This page contains some basic documentation for the SimpleSite project. To understand the project completely please refer to the documentation on the Pylons Book website at http://pylonsbook.com or read the source code directly.

25.1 The `simplesite` Module

Contains all the controllers, model and templates as sub-modules.

25.2 The `controllers` Module

Contains all the controllers. The most important of which is `PageController`.

class `simplesite.controllers.`**`PageController`**

The `PageController` is responsible for displaying pages as well as allowing users to add, edit, delete and list pages.

`PageController.`**`view`**(*self*[, *id=None*])

When a user visits a URL such as `/view/page/1` the `PageController` class's `view()` action is called to render the page.

The page controller makes use of a FormEncode schema to validate the page data it receives. Here is the schema it uses:

```
class NewPageForm(formencode.Schema):
    allow_extra_fields = True
    filter_extra_fields = True
    content = formencode.validators.String(
        not_empty=True,
        messages={
            'empty':'Please enter some content for the page. '
        }
    )
    heading = formencode.validators.String()
    title = formencode.validators.String(not_empty=True)
```

As you can see the schema includes validators for the title, heading and content.

25.3 The `utils` Module

25.3.1 The utils module is really cool!

```
class NewPageForm(formencode.Schema):
    allow_extra_fields = True
    filter_extra_fields = True
    content = formencode.validators.String(
        not_empty=True,
        messages={
            'empty':'Please enter some content for the page. '
        }
    )
    heading = formencode.validators.String()
    title = formencode.validators.String(not_empty=True)
```

class old.lib.utils.**ApplicationSettings**

ApplicationSettings is a class that adds functionality to a ApplicationSettings object.

The value of the applicationSettings attribute is the most recently added ApplicationSettings model. Other values, e.g., storageOrthography or morphemeBreakInventory, are class instances or other data structures built upon the application settings properties.

getAttributes()

Generate some higher-level data structures for the application settings model, providing sensible defaults where appropriate.

class old.lib.utils.**EventHook**

EventHook is for event-based (PubSub) stuff in Python. It is taken from http://www.voidspace.org.uk/python/weblog/arch_d7_2007_02_03.shtml#e616. See also http://stackoverflow.com/questions/1092531/event-system-in-python.

class old.lib.utils.**Inventory**(*inputList*)

An inventory is a set of graphemes/polygraphs/characters. Initialization requires a list.

This class should be the base class from which the Orthography class inherits but I don't have time to implement that right now.

getNonMatchingSubstrings(*string*)

Return a list of substrings of string that are not constructable using the inventory. This is useful for showing invalid substrings.

getRegexValidator(*substr=False*)

Returns a regex that matches only strings composed of zero or more of the graphemes in the inventory (plus the space character).

stringIsValid(*string*)

Return False if string cannot be generated by concatenating the elements of the orthography; otherwise, return True.

class old.lib.utils.**JSONOLDEncoder**(*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *encoding='utf-8'*, *default=None*, *use_decimal=True*, *namedtuple_as_object=True*, *tuple_as_array=True*, *bigint_as_string=False*, *item_sort_key=None*)

Permits the jsonification of an OLD class instance obj via

jsonString = json.dumps(obj, cls=JSONOLDEncoder)

Note: support for additional OLD classes will be implemented as needed ...

class old.lib.utils.**OrderBySchema**(**args*, ***kw*)

Messages

badDictType: The input must be dict-like (not a %(type)s: %(value)r)

badType: The input must be a string (not a %(type)s: %(value)r)

empty: Please enter a value

missingValue: Missing value

noneType: The input must be a string (not None)

notExpected: The input field %(name)s was not expected.

class old.lib.utils.**PaginatorSchema**(**args*, ***kw*)

Messages

badDictType: The input must be dict-like (not a %(type)s: %(value)r)

badType: The input must be a string (not a %(type)s: %(value)r)

empty: Please enter a value

missingValue: Missing value

noneType: The input must be a string (not None)

notExpected: The input field %(name)s was not expected.

class old.lib.utils.**State**

Empty class used to create a state instance with a 'full_dict' attribute that points to a dict of values being validated by a schema. For example, the call to FormSchema().to_python in controllers/forms.py requires this State() instance as its second argument in order to make the inventory-based validators work correctly (see, e.g., ValidOrthographicTranscription).

old.lib.utils.**addOrderBy**(*query*, *orderByParams*, *queryBuilder*, *primaryKey='id'*)

Add an ORDER BY clause to the query using the getSQLAOrderBy method of the supplied queryBuilder (if possible) or using a default ORDER BY <primaryKey> ASC.

old.lib.utils.**clearAllModels**(*retain=['Language']*)

Convenience function for removing all OLD models from the database. The retain parameter is a list of model names that should not be cleared.

old.lib.utils.**clearDirectoryOfFiles**(*directoryPath*)

Removes all files from the directory path but leaves the directory.

old.lib.utils.**commandLineProgramInstalled**(*command*)

Command is the list representing the command-line utility.

old.lib.utils.**compile_query**(*query*, ***kwargs*)

Return the SQLAlchemy query as a bona fide MySQL query. Taken from http://stackoverflow.com/questions/4617291/how-do-i-get-a-raw-compiled-sql-query-from-a-sqlalchemy-expression.

old.lib.utils.**createResearcherDirectory**(*researcher*, ***kwargs*)

Creates a directory named researcher.username in files/researchers/.

old.lib.utils.**dateString2date**(*dateString*)

Parse an ISO 8601-formatted date into a Python date object.

old.lib.utils.**datetimeString2datetime**(*datetimeString*)
Parse an ISO 8601-formatted datetime into a Python datetime object. Cf. http://stackoverflow.com/questions/531157/parsing-datetime-strings-with-microseconds

Previously called ISO8601Str2datetime.

old.lib.utils.**deleteKey**(*dict_*, *key_*)
Try to delete the **key_** from the **dict_**; then return the **dict_**.

old.lib.utils.**destroyAllResearcherDirectories**(***kwargs*)
Removes all directories from files/researchers/.

old.lib.utils.**destroyResearcherDirectory**(*researcher*, ***kwargs*)
Destroys a directory named researcher.username in files/researchers/.

old.lib.utils.**encryptPassword**(*password*, *salt*)
Use PassLib's pbkdf2 implementation to generate a hash from a password. Cf. http://packages.python.org/passlib/lib/passlib.hash.pbkdf2_digest.html#passlib.hash.pbkdf2_sha512

old.lib.utils.**escREMetaChars**(*string*)
Escapes regex metacharacters so that we can formulate an SQL regular expression based on an arbitrary, user-specified inventory of graphemes/polygraphs.

```
>>> escREMetaChars(u'-')
u'\\-'
```

old.lib.utils.**ffmpegEncodes**(*format_*)
Check if ffmpeg encodes the input format. First check if it's installed.

old.lib.utils.**ffmpegInstalled**()
Check if the ffmpeg command-line utility is installed on the host. Check first if the answer to this question is cached in app_globals.

old.lib.utils.**getCollectionBackupsByCollectionId**(*collectionId*)
Return all CollectionBackup models with collection_id = collectionId. WARNING: unexpected data may be returned (on an SQLite backend) if primary key ids of deleted collections are recycled.

old.lib.utils.**getCollectionBackupsByUUID**(*UUID*)
Return all CollectionBackup models with UUID = UUID.

old.lib.utils.**getCollectionByUUID**(*UUID*)
Return the first (and only, hopefully) Collection model with UUID.

old.lib.utils.**getConfig**(***kwargs*)
Try desperately to get a Pylons config object. The best thing is if a config object is passed in kwargs['config'].

old.lib.utils.**getDataForNewAction**(*GET_params*, *getterMap*, *modelNameMap*)
Return a dictionary whose values are lists of OLD model objects. GET_params is the dict-like object created by Pylons that is created on a GET request. The getterMap param is a dict from key names (e.g., 'users') to a getter function that retrieves that resource (e.g., getUsers). The modelNameMap is a dict from key names (e.g., 'users') to the relevant model (e.g., 'User').

If no GET parameters are provided (i.e., GET_params is empty), then retrieve all data (using getterMap) from the db and return them.

If GET parameters are specified, then for each parameter whose value is a non-empty string (and is not a valid ISO 8601 datetime), retrieve and return the appropriate list of objects.

If the value of a GET parameter is a valid ISO 8601 datetime string, retrieve and return the appropriate list of objects *only* if the datetime param does *not* match the most recent datetimeModified value of the relevant data store (i.e., model object). This makes sense because a non-match indicates that the requester has out-of-date data.

old.lib.utils.**getForeignWordTranscriptions**()
Returns a 4-tuple (fWNarrPhonTranscrs, fWBroadPhonTranscrs, fWOrthTranscrs, fWMorphTranscrs) where each element is a list of transcriptions (narrow phonetic, broad phonetic, orthographic, morphemic) of foreign words.

old.lib.utils.**getForeignWords**()
Return the forms that are tagged with a 'foreign word' tag. This is useful for input validation as foreign words may contain otherwise illicit characters/graphemes.

old.lib.utils.**getFormBackupsByFormId**(*formId*)
Return all FormBackup models with form_id = formId. WARNING: unexpected data may be returned (on an SQLite backend) if primary key ids of deleted forms are recycled.

old.lib.utils.**getFormBackupsByUUID**(*UUID*)
Return all FormBackup models with UUID = UUID.

old.lib.utils.**getFormByUUID**(*UUID*)
Return the first (and only, hopefully) Form model with UUID.

old.lib.utils.**getMorphemeDelimiters**()
Return the morpheme delimiters from app settings as a list.

old.lib.utils.**getMostRecentModificationDatetime**(*modelName*)
Return the most recent datetimeModified attribute for the model with the provided modelName. If the modelName is not recognized, return None.

old.lib.utils.**getSearchParameters**(*queryBuilder*)
Given an SQLAQueryBuilder instance, return (relative to the model being searched) the list of attributes and their aliases and licit relations relevant to searching.

old.lib.utils.**getStateObject**(*values*)
Return a State instance with some special attributes needed in the forms and oldcollections controllers.

old.lib.utils.**getSubprocess**(*command*)
Return a subprocess process. The command argument is a list. See http://docs.python.org/2/library/subprocess.html

old.lib.utils.**getUnicodeCodePoints**(*string*)
Returns a string of comma-delimited unicode code points corresponding to the characters in the input string.

old.lib.utils.**getUnicodeNames**(*string*)
Returns a string of comma-delimited unicode character names corresponding to the characters in the input string.

old.lib.utils.**getUnrestrictedUsers**()
Return the list of unrestricted users in app_globals.applicationSettings.applicationSettings.unrestrictedUsers.

old.lib.utils.**isLexical**(*form*)
Return True if the input form is lexical, i.e, if neither its morpheme break nor its morpheme gloss lines contain the space character or any of the morpheme delimiters. Note: designed to work on dict representations of forms also.

old.lib.utils.**jsonify**(*func*)
Action decorator that formats output for JSON

Given a function that will return content, this decorator will turn the result into JSON, with a content-type of 'application/json' and output it.

Adapted from pylons.decorators.

old.lib.utils.**makeDirectorySafely**(*path*)
Create a directory and avoid race conditions. Taken from http://stackoverflow.com/questions/273192/python-best-way-to-create-directory-if-it-doesnt-exist-for-file-write. Listed as make_sure_path_exists.

old.lib.utils.**normalize**(*unistr*)
Return a unistr using canonical decompositional normalization (NFD).

old.lib.utils.**normalizeDict**(*dict_*)
NFD normalize all unicode values in **dict_**.

old.lib.utils.**removeAllWhiteSpace**(*string*)
Remove all spaces, newlines and tabs.

old.lib.utils.**restrict**(**methods*)
Restricts access to the function depending on HTTP method

Just like pylons.decorators.rest.restrict except it returns JSON.

old.lib.utils.**secureFilename**(*path*)
Removes null bytes, path.sep and path.altsep from a path. From http://lucumr.pocoo.org/2010/12/24/common-mistakes-as-web-developer/

old.lib.utils.**sendPasswordResetEmailTo**(*user*, *newPassword*, ***kwargs*)
Send the "password reset" email to the user. **kwargs should contain a config object (with 'config' as key) or a config file name (e.g., 'production.ini' with 'configFilename' as key). If password_reset_smtp_server is set to smtp.gmail.com in the config file, then the email will be sent using smtp.gmail.com and the system will expect a gmail.ini file with valid gmail_from_address and gmail_from_password values. If the config file is test.ini and there is a test_email_to value, then that value will be the target of the email – this allows testers to verify that an email is in fact being received.

old.lib.utils.**toSingleSpace**(*string*)
Remove leading and trailing whitespace and replace newlines, tabs and sequences of 2 or more space to one space.

old.lib.utils.**userIsAuthorizedToAccessModel**(*user*, *modelObject*, *unrestrictedUsers*)
Return True if the user is authorized to access the model object. Models tagged with the 'restricted' tag are only accessible to administrators, their enterers and unrestricted users.

old.lib.utils.**userIsUnrestricted**(*user*, *unrestrictedUsers*)
Return True if the user is an administrator, unrestricted or there is no restricted tag.

CHAPTER

TWENTYSIX

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX