Kura

Boudewijn Rempt

Kura
by Boudewijn Rempt

Published November 2002

Although every reasonable effort has been made to incorporate accurate and useful information into this book, the copyright holders make no representation about the suitability of this book or the information therein for any purpose. It is provided “as is” without expressed or implied warranty.

Table of Contents

List of Examples

Preface

Kura is an application for linguists working with fieldwork or manuscript data. It supports the entry, analysis and presentation of linguistic data, be it recordings or manuscripts. it is a multi-user, multi-language, multi-project application, meaning that more linguists can simultaneously work with all data, and that the data is not restricted to one language. Besides, Kura has the notion of projects, and all data can be handled within any project.

All linguistic data is stored in parsed form, facilitating quick analysis, and the relations between data can also be stored. Kura uses a relational database for this, and the database can grow to enormous sizes (gigabytes) without any problems. Still, Kura is in the first place designed for linguists working on data gathered during fieldwork, and those corpora will in all likelyhood be relatively small.

Kura consists of three main parts: the database with a set of relatively sophisticated components that represent linguistic notions, such as text or lexeme, the desktop client that can be used to enter data and analysises, and the special-purpose webserver, that can present the linguistic data to the outside world.

Kura is network transparent. Given an internet connection, users of Kura can use their desktop client to connect to any Kura database anywhere in the world they have access to.

But there is also an option to work stand-alone. You don’t have to use the database: all data can also be stored in a file. This may be more comfortable for people who don’t want to install and maintain a database server or who prefer an easy way to exchange sets of data.

Kura is extensible. The application is written in Python, the programming language of choice for subject experts and it is easy to develop and use other modules, such as parsers. Because Kura is open source software, it is easy to adapt any component to the special needs of the user.

Kura’s datamodel is also extensible. Relations between elements in a text and lexemes can be created and tagged with a user extensible set of tags. Likewise, elements in a text can be related to each other as can lexemes. It is easy to create diachronic relations between lexemes from different languages, or synchronic relations between relations in a language. Finally, all elements in Kura, text, phrase, word or lexeme can be describe using a user-definable set of characteristics. The default distribution of Kura includes an example set of these characteristics.

Kura uses only one character set: Unicode. Currently restricted to the Basic Multilingual Plane, this already enables the use of the vast majority of scripts in current use. Kura supports that standard OS-supplied tools to enter text in non-western scripts, like XIM on X11 or IME’s on Windows.

Kura runs both on Windows and Unix/X11. However, all development is done using Unix/X11. Kura for Windows is free software, but the libraries need to run Kura on Windows (and the OS itself) are not free, and need to be acquired by the user who elects to use Windows.

1. Features

- multi-user
- multi-language
- multi-project
- network-transparent
- integration of recordings and manuscripts
- extensible both through a dynamic data model and through adding code.
- open-source
- 100% Unicode
- Support for standard tagsets
- Generation of documents in docbook (which can be used to produce html or pdf, in the future rtf, too).

2. Requirements

Kura 2.0 requires the following components to be present:

- Python 2.1(http://www.python.org) or greater
- optionally: MySQL 2.23.27(http://www.mysql.org) or greater and mysql-python (http://sourceforge.net/projects/mysql-python/).
- Qt 3.0.0(http://www.trolltech.com) or greater
- PyQt 3.0(http://www.riverbankcomputing.co.uk) or greater

I track the development of these runtime components pretty closely, so it's a good bet that the cvs version of Kura will always require the latest versions.

Chapter 1. Introduction

1.1. Principles

Kura is build around the idea of linguistic data. Linguistic data are stored in a database and relations between the data are then created, either by the linguist, or automatically.

There are four core types of linguistic data: texts, recordings, scans and lexicon. Of these, texts and lexicon are analyzed by Kura. Recordings of fieldwork sessions or scans of manuscripts are related to texts. You can generate documents or websites that include scanned images or point to field recordings.

1.2. Parsed text

In contrast to other systems, like Shoebox, Kura stores texts in parsed and tagged form in the database. This makes it easier to create relations, but more difficult to re-assemble complete texts. The relations between the words in a text and the lexemes in the lexicon are stored too; the lexicon can also be used to dynamically tag texts.

1.3. Multi-user

Kura stores facts about many languages and can be used by many users. Which linguistic fact has been entered by which user is recorded, too, so it is always possbile to account for the data in a scholarly way. Currently Kura does not record the change history of a linguistic fact.

1.4. Publishing

After data has been entered, it is possible to publish it. Kura can be used together with the Grammar project to create complex, dynamic documents in either html or pdf for. Using hyperlinks, users of the data can trace their own path through a language. For instance, when reading a certain text, a user might want to look up a word in the lexicon. He clicks on the word in the lexicon, and Kura shows the lexeme, and all the sentences where that lexeme occurs in the corpus.

1.5. tags

Kura is extensible: every important language fact can be annotated with user-defined tags. For instance, a text can be tagged with one or more references, and a word in a text with a glosse, a translation, a syntactical function or another bibliographic reference. Kura leaves you free in the creation of these tags, but provides an extensive set of basic tags.

1.6. Terminology

text

A text from a certain language; a connected narrative. Texts can be associated with recordings or scanned manuscripts.

stream

Sentences in a text, or phrases, at the descretion of the linguist. The main division of a text.

element

Words in a text; can be subdivided into subelements, like morphemes or phonemes. Technically, elements are the subdivision of streams and of themselves.

tag

A bit of information that’s ’tagged’ onto a text, a stream, an element or a lexeme. Tags can be either a short free-format text, a longer free-format text, an entry out of a predefined list or an entry out of the list of references. For instance, a word in a sentence can be tagged with the grammatical role of subject.

parse set

Currently de-implemented because of a pending redesign (that is, they have been removed for the current version of Kura, 2.0), parse sets are groupings of elements in a text.

1.7. Relational database

Kura uses a relational database model to store the linguistic data. Even the file-based datastores mimic a relational database and actually use the same datamodel as the database backend.

I’ve not yet succeeded in making this completely transparent to users, so I’d like to appeal to their intelligence and keep the following in mind:

- all data is stored in tables - long lists of records. A record is a set of related data.
- tables are related through the use of keys. For instance, every text is numbered (in the field ’textnr’), and Kura knows which sentence belongs to which text because every ’stream’ stores the textnr, too. This is a parent (the text) - child (the sentence) relation.
- You should not try to remove a text without removing the streams, for instance, because then the streams wouldn’t belong to any known text. If you remove a text, all the underlying data will be automatically removed, too.
- Installing and administering a database is a bit of a bother. It’s not for nothing that dba’s (database administrators) earn quite well. However, one Kura database is quite enough for a group of linguistis working on related languages. And if you cannot administer a database, you can always use the file based datastore.

1.8. Unicode

Kura supports Unicode throughout. You can use your OS’s input methods (kcharselect, XIM or keyboard layouts) to enter data in non-latin scripts.

Five minutes of work on the web with a search engine will find you many Unicode fonts. I am particularly satisfied with Microsoft’s Arial Unicode and the Gnu Unicode font. Both cover a large range of characters, but Arial Unicode will show combining characters (for instance a vowel and a diacritic mark) better. For presentation purposes that use the western script and IPA, the Thryomanes font might be a better choice.

When using combining characters, type the base character first, and the diacritic second.

I. Installation and Configuration

Table of Contents

Kura is a serious application. That means that installing Kura will take some effort. First of all, the PyQt GUI library is not available by default on all Unix systems. It is not very difficult to install, but you are urged to closely follow the installation instructions.

If you are wondering *why* I am using PyQt instead of GUI library like Tkinter, which is available almost everywhere where Python is available, the answer is as follows:

First, Tkinter did not offer the advanced widgets, such as the table widget, that Kura needs when I started coding Kura. Secondly, and more importantly, only PyQt has support for Unicode through and through. If I had stuck with Tkinter or wxWindows you wouldn’t have been able to use IPA in your data.

So, bite the bullet, and install PyQt. Once that’s done, you’ve had the worst, I promise.

Chapter 2. Installation

Two versions of Kura are made available: an rpm package for rpm-based Unix systems and a source distribution.

Installing the rpm is as simple as executing:

```
rpm --install kura*.rpm
```

The source distribution needs to be unpacked, after which you can run:

```
python setup.py install
```

Please note that Kura by default installs with the `/usr/local` prefix. If you want to install Kura to a different location, you will need to edit the `setup.cfg` file and the `kura` script to reflect your preferred location.

Chapter 3. Database Configuration

Out of the box Kura will use the file-based datastore. This is not multi-user and cannot be used as a central network-accessible database.

If you want to share your Kura data with other linguists, you need to install the MySQL database. Please consult the MySQL manual about installing MySQL.

When MySQL is installed, you have created a new database, you must run the `config.sql` script from the datamodel subdirectory:

```
 mysql mydatabase  config.sql
```

Previous versions of Kura used a slightly different datamodel, which is incompatible with the current datamodel. If you need to migrate data, please contact me (<`boud@valdyas.org`>), and I will try to help you.

Chapter 4. Settings and configuration options

Kura comes with a default configuration consisting of a number of pre-defined tags that are useful for tagging texts and lexical items, a number of sample texts and so on. Configuring Kura for a particular situation demands some care however - most aspects of Kura are configurable. See the notes to the individual menu options.

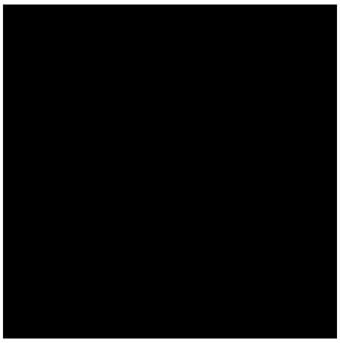
II. Tutorial

Table of Contents

You can start Kura by executing the command **kura** from the command line:

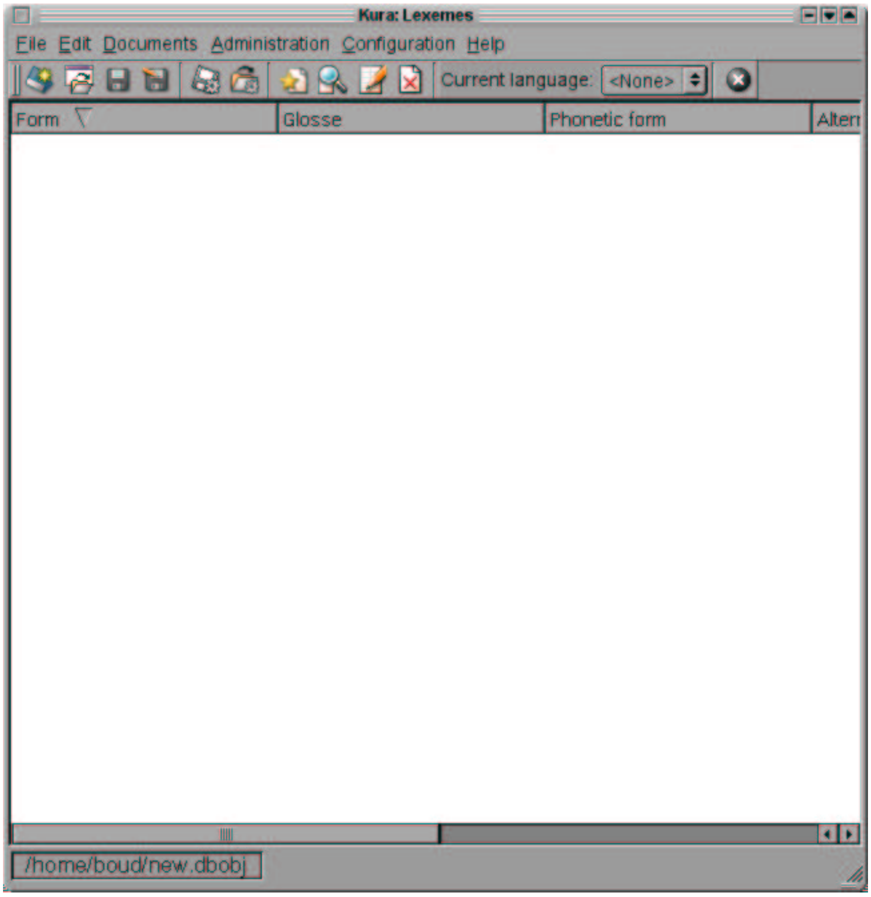
```
kura
```

After a while the splash screen will show:



Kura starts

Just before you have managed to recognize all the various scripts in this logo, Kura itself appears. If this is the first time you’ve run Kura, Kura will open with a completely empty database. This is generally not what you want; you want to start with a database that contains at least the default configuration data.



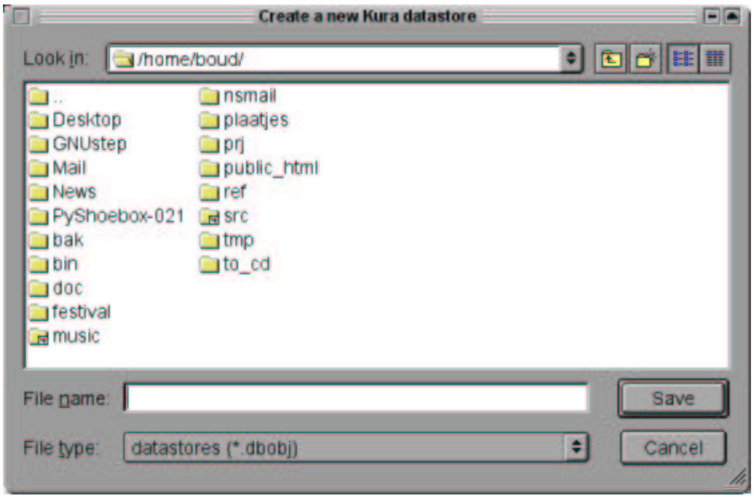
Current language: <None>

/home/boud/new.dbobj

Virgin Kura

If you are using the MySQL database backend, you must have configured the database and started the server by now.

If you want to use the file-based backend, this is the moment to create a new datastore file that will contain the basic configuration data. Choose CTRL-N or New from the File menu. Choose a filename, with the obligatory extension `.dbobj` (this extension is not automatically added if you forget it.):



Look in: /home/boud/

..
Desktop
GNUstep
Mail
News
PyShoebox-021
bak
bin
doc
festival
music
nsmail
plaatjes
prj
public_html
ref
src
tmp
to_cd

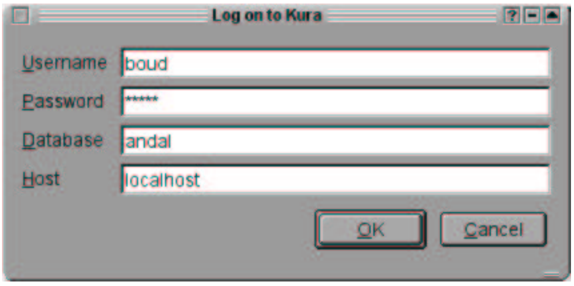
File name:

File type: datastores (*.dbobj)

New file dialog

> **Templates:** Kura uses a template file to determine the default configuration. If you want to use a custom template, first create a new datastore, enter the extra configuration details you need, and then save the datastore file and copy it to the Kura installation directory (most likely `/usr/local/share/kura`) under the name template.dbobj.
>
> In a future version of Kura, it will be possible to choose from different templates.

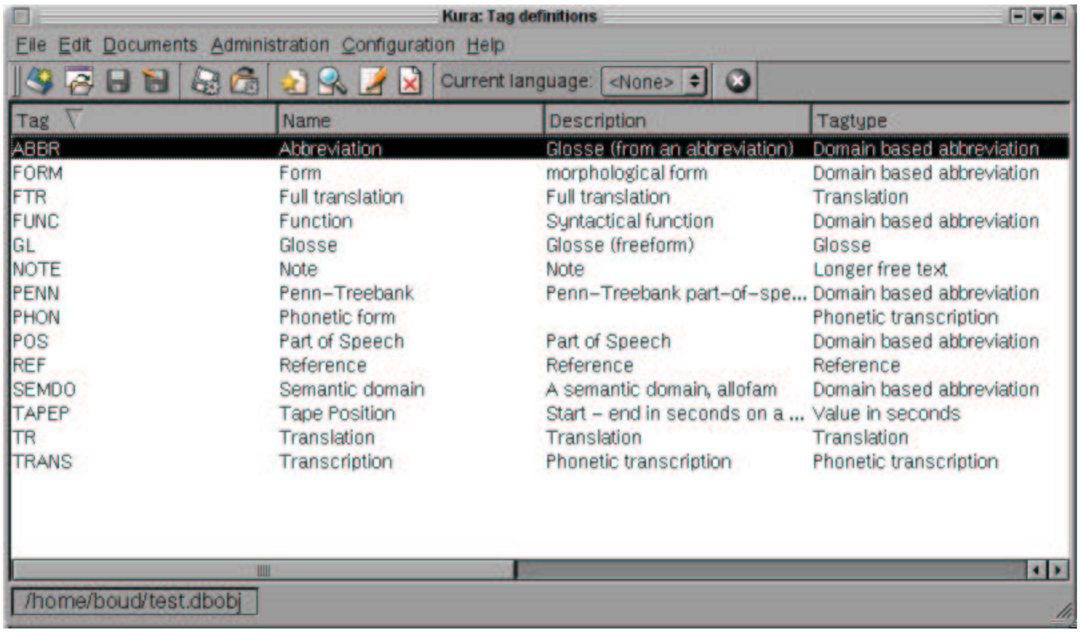
If you want to connect to a database, use Connect from the File to open the connect dialog:



Username: boud
Password: *****
Database: andal
Host: localhost

Connect to database

You will now have access to a working datastore or datastore. The statusbar is updated accordingly, and if you choose Configuration Tags you will see that the empty screen is filled with data:



Tag	Name	Description	Tagtype
ABBR	Abbreviation	Glosse (from an abbreviation)	Domain based abbreviation
FORM	Form	morphological form	Domain based abbreviation
FTR	Full translation	Full translation	Translation
FUNC	Function	Syntactical function	Domain based abbreviation
GL	Glosse	Glosse (freeform)	Glosse
NOTE	Note	Note	Longer free text
PENN	Penn–Treebank	Penn–Treebank part–of–spe...	Domain based abbreviation
PHON	Phonetic form		Phonetic transcription
POS	Part of Speech	Part of Speech	Domain based abbreviation
REF	Reference	Reference	Reference
SEMDO	Semantic domain	A semantic domain, allofam	Domain based abbreviation
TAPEP	Tape Position	Start – end in seconds on a ...	Value in seconds
TR	Translation	Translation	Translation
TRANS	Transcription	Phonetic transcription	Phonetic transcription

/home/boud/test.dbobj

A screenful of data.

If you’ve made it to this point, you’re ready to start exploring Kura in earnest. If you mess up, you can always create a new datastore. So feel free to explore a bit before coming back for the rest of the tutorial.

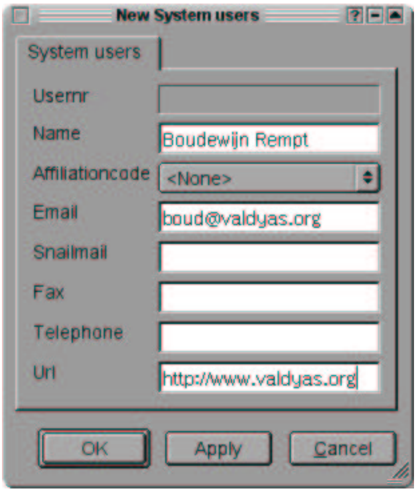
Chapter 5. Begin

Begin with reviewing the default configuration. Perhaps you need to add new tag definitions to Kura to suit your research. Do not, however, *remove* default tags from the datastore. Kura uses the values of some of these tags, especially **GLOSS** to construct interlinear text.

If you are puzzled by the relations between the configuration details and the contents of these tables, please review the chapter Configuration.

For now, choose from the Administration menu the entry User. This is an empty table. Kura k track of which user is responsible for which linguistic fact, and before you can start entering data, you have to create a user for yourself, and for every other linguist who will be using this Kura database or datastore.

Choose EditNew Item, or press the New Item toolbar button (a sheet of paper with a yellow star on it), or press CTRL-ALT-N or INS to open the **New User** dialog window:

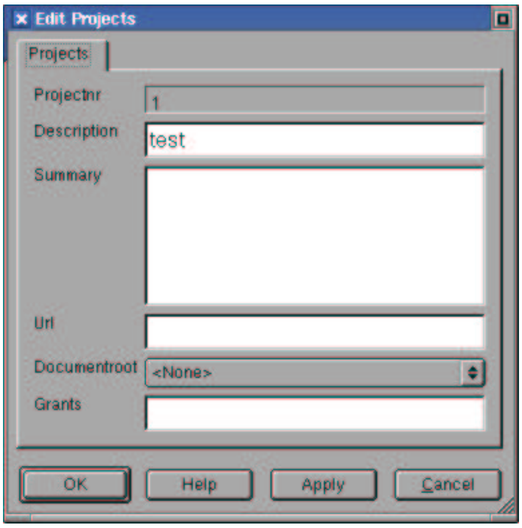


Usernr	
Name	Boudewijn Rempt
Affiliationcode	<None>
Email	boud@valdyas.org
Snailmail	
Fax	
Telephone	
Url	http://www.valdyas.org

Create a new user

Enter the relavant data in the fields, and press OK. The Affiliation Code dropdown list is empty, because we haven't filled any data in the affiliation table. If you wish, you can do that, and then add your academic affiliation to your user data.

Next, define a project. Kura is organized in projects. A project can be fieldwork on a certain language for a Phd, or work for a particular paper. Choose AdminstrationProject to open the projects table. Again, press INS to create a new project:

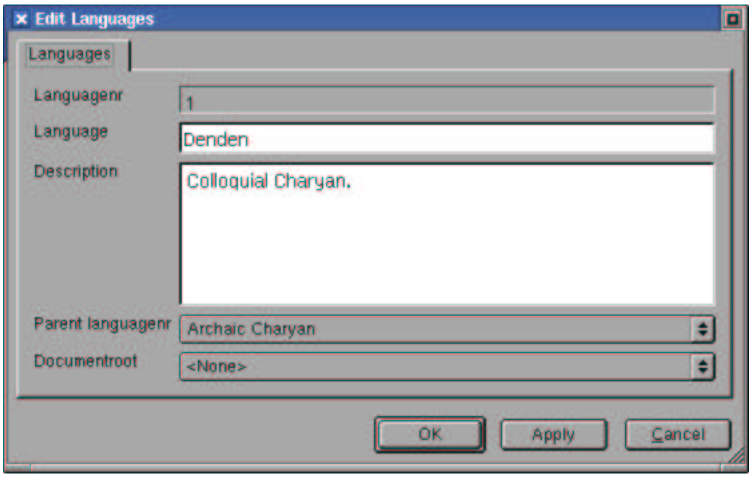


Projectnr: 1
Description: test
Summary:
Url:
Documentroot: <None>
Grants:

Create a new project

The information you enter here is rather freeform. The URL can point to the project website, and you can add any grants you've received, too. The summary can appear on automatically generated documents.

Now that we have a user and a project, it's time to make sure that Kura knows about the language we are working with. Open AdministrationLanguage and press INS to enter a new language:



Languagenr: 1
Language: Denden
Description: Colloquial Charyan.
Parent languagenr: Archaic Charyan
Documentroot: <None>

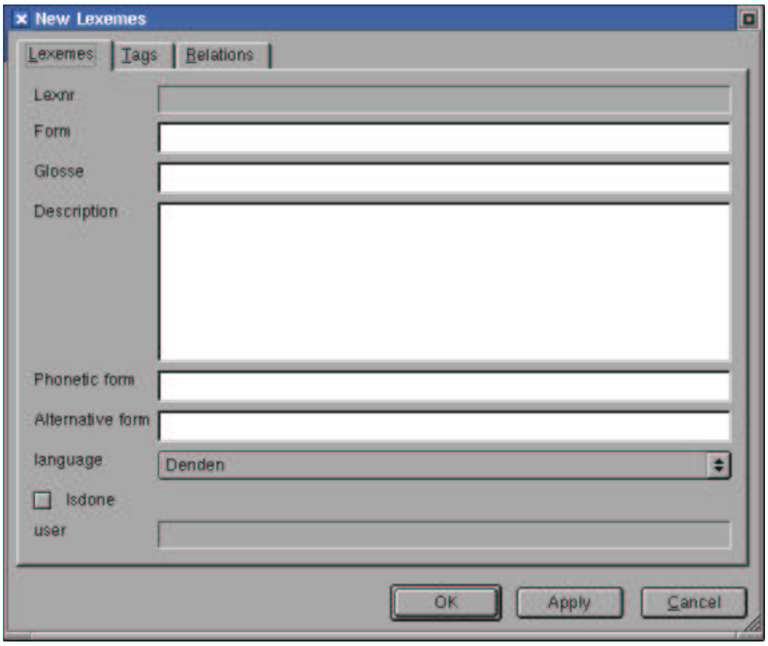
Create a new language

The language dialog has the option to define a parent language. If you enter more than one language in Kura, you can create language trees. In the dictionary module, you can use this to request Kura to search for related words in related languages.

When this data is entered, we are ready to create the lexicon and the database of analyzed texts.

Chapter 6. Entering Lexical Data

Let's start with entering a few words. Open the Documents menu, choose Lexicon menuitem. You will see an empty screen. By now you know what to do to enter a new record: press INS. The New Lexemes dialog window will open. The text, stream and element dialogs look a lot like this one. First one tab, with basic data, then a tab with tags and finally one or more tab pages with specific data.



language: Denden

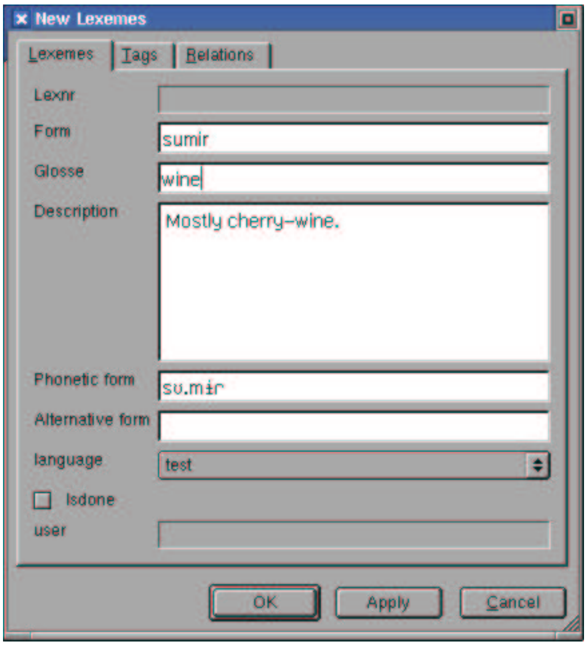
☐ Isdone

user

Create a new lexeme

The first tab presents the basic lexical data: the attested *form*, the basic *glosse*, a longer description, the *phonetic form*, an eventual *alternative form*, the language this lexeme comes from, and a checkbox that indicates whether you are done with this lexeme.

The alternative form is especially useful if your language has spelling variants or if you are converting from one transcription system to another, and want to be able to query on the old transcription.



Lexnr

Form: sumir

Glosse: wine

Description: Mostly cherry-wine.

Phonetic form: sʊ.mɨr

Alternative form

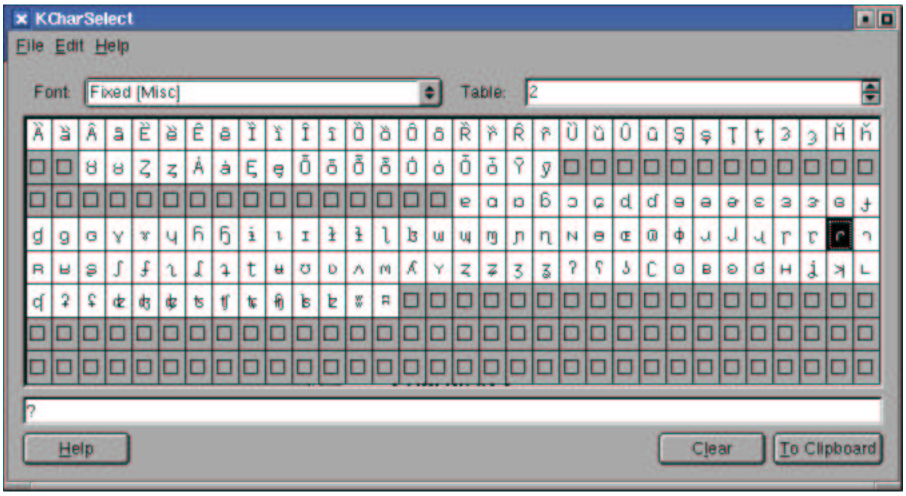
language: test

□ Isdone

user

A lexeme

Enter some likely data, if you would. You can copy and paste Unicode characters from a utility like KCharSelect, by either with your middle mousebutton, or with CTRL-V.



Font: Fixed [Misc]

Table: 2

ʔ

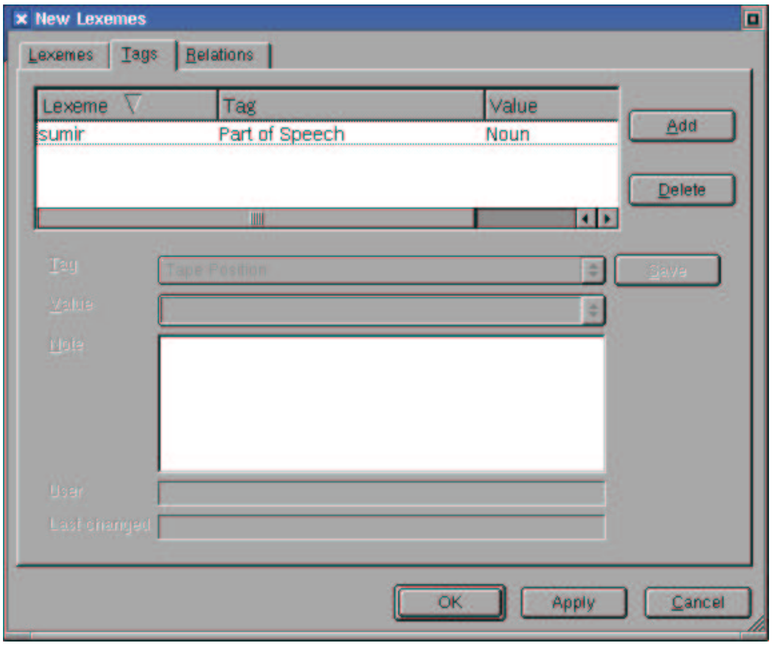
KCharSelect

The second tab, Tags, allows you to add extended data to the lexeme you’ve just entered. The top half is occupied by a list of all tags that have already been added; when you click on the **Add** button, a new tag record will be created which you can edit in the bottom half.

If you want to change a tag, click on the tag in the list, and it will be opened, too.

You can save your changes by pressing **Save**.

The delete button, finally, deletes. That’s no surprise.



Lexeme	Tag	Value
sumir	Part of Speech	Noun

Tag: Tape Position
Value
Note
User
Last changed

Adding a tag

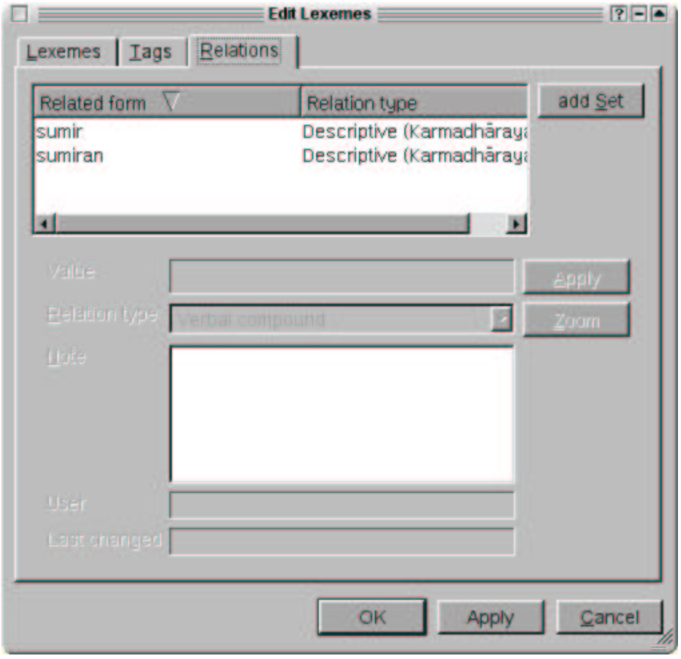
Let’s tell Kura the Part of Speech of the lexeme we just entered. As you can see, Part of Speech is one of the many possible tags that have already been defined in the configuration tables. If you select it from the combobox labeled Tag, the combobox labeled Value will be filled with all the possible parts of speech, that also have been entered in the configuration tables.

You can always add a comment to a tag, too, in the description field.

If you choose a tag that allows you to enter a short value, such as a glosse, the drop-down list can contain all the values you’ve entered before. This, however, makes filling the list slow, and is therefore a Preferences option. (File/Preferences/Show existing values for tags).

A minor inconvenience is that you have to press Apply before adding tags.

The final tab in the Lexeme dialog is Relations. Here, you can create relations between the lexemes in your lexicon. Let’s enter two more words, and create a relation between those.



Related form	Relation type
sumir	Descriptive (Karmadhāraya
sumiran	Descriptive (Karmadhāraya

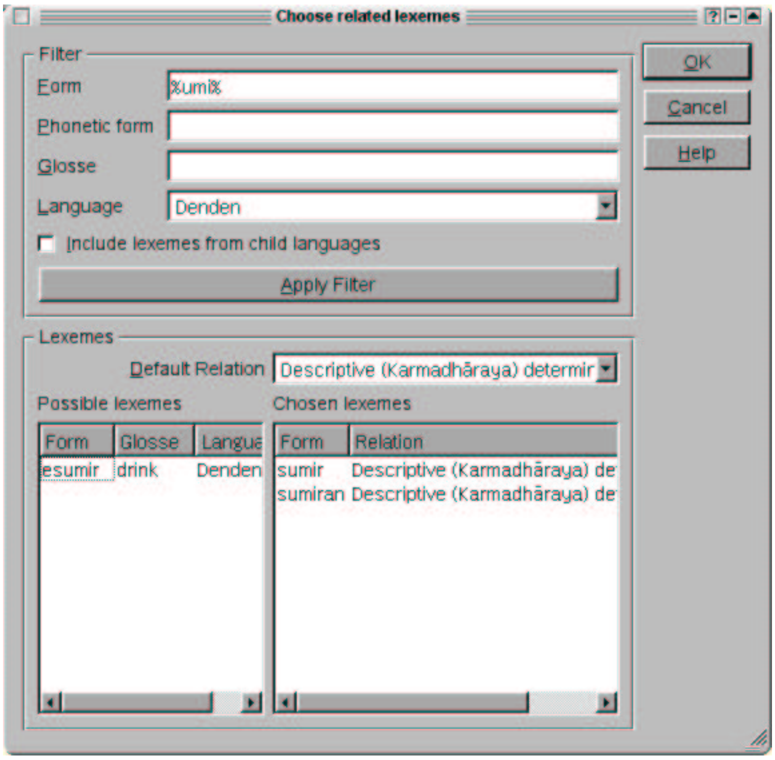
Value
Relation type: Verbal compound
Note
User
Last changed

Related lexemes.

In the filter section of the dialog you can enter search criteria for lexemes. You can use % wildcard to match anything, as is done here. When you press Apply Filter, the listbox with possible lexemes is filled.

You can then select a Default Relation: in this case, Kharamadarya compound. You can add new relations when you configure Kura.

When you double click on an entry in either listbox, the entry is moved to the other side. Here, I've selected the lexemes 'sumir', meaning wine, and 'sumiran', meaning drinker, that form the components of the Kharamadarya compound 'sumirsumiran', 'wine-bibber'.



Form: %umi%

Phonetic form:

Glosse:

Language: Denden

☐ Include lexemes from child languages

Default Relation: Descriptive (Karmadhāraya) determir

Possible lexemes

Form	Glosse	Langua
esumir	drink	Denden

Chosen lexemes

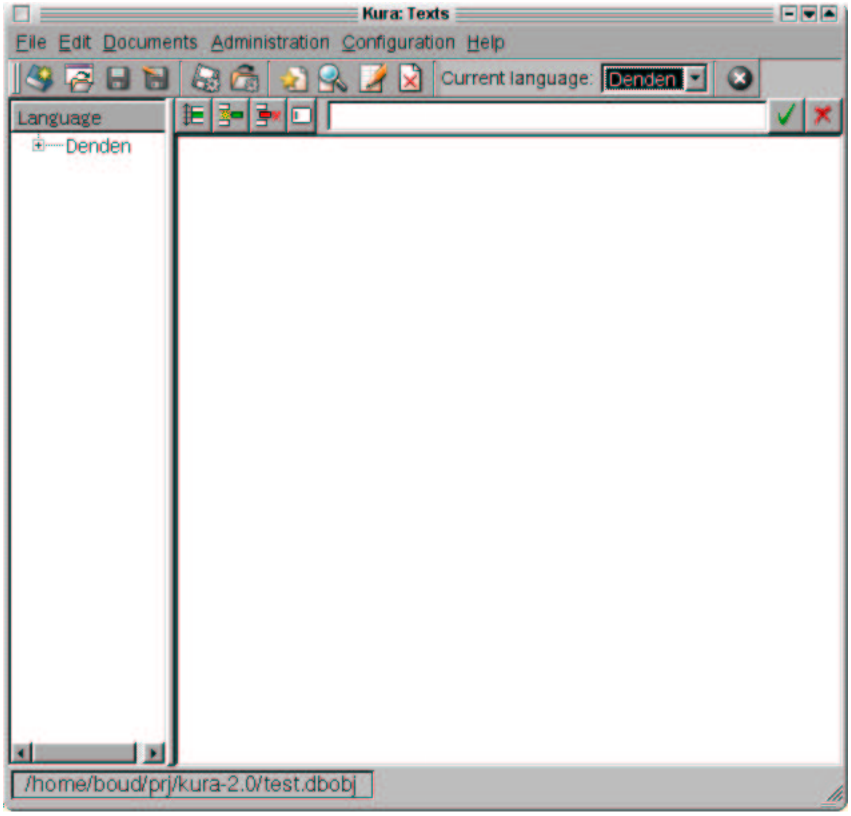
Form	Relation
sumir	Descriptive (Karmadhāraya) de
sumiran	Descriptive (Karmadhāraya) de

Creating a relation.

From the lexeme view (and references and scans), you can drag and drop a docbook-tagged reference to an editor.

Chapter 7. Glossing a Text

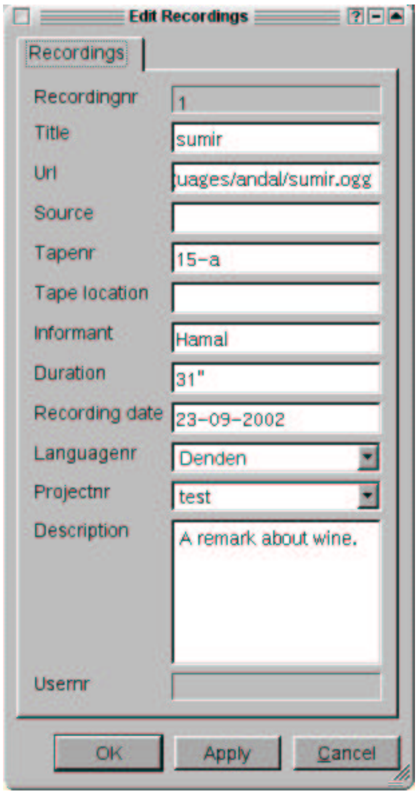
Having entered lexemes is helpful when glossing a text, and, vice versa, glossing a text results in new lexemes. In this chapter we will use the New Text Wizard to enter and gloss a new text. If you choose Documents/Text, you will see the following screen, one of the more complex in Kura:


Kura: Texts
File Edit Documents Administration Configuration Help
Current language: Denden
Language
Denden
/home/boud/prj/kura-2.0/test.dbobj

Texts in Kura.

The texts window is divided in two parts: on the left, there is a tree of languages and texts. On the right, there is a pane that shows the selected text interlinearly glossed.

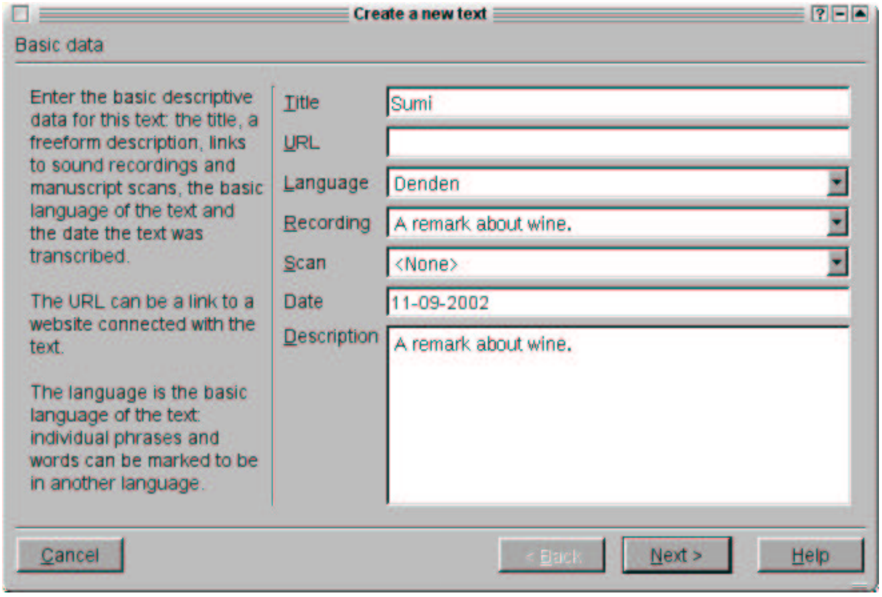
Texts generally have a source in fieldwork or in a manuscript. Kura offers facilities to record those sources and link the source to the text, and present the source in generated documents. In this case, the source is a recording (enter it in Documents/Recording:



Recordingnr 1
Title sumir
Url uages/andal/sumir.ogg
Source
Tapenr 15–a
Tape location
Informant Hamal
Duration 31"
Recording date 23–09–2002
Languagenr Denden
Projectnr test
Description A remark about wine.
Usernr

A tape recording.

Choose Edit/New Item (or press INS) to start the New Text Wizard, and fill in the blanks:



Enter the basic descriptive data for this text: the title, a freeform description, links to sound recordings and manuscript scans, the basic language of the text and the date the text was transcribed.

The URL can be a link to a website connected with the text.

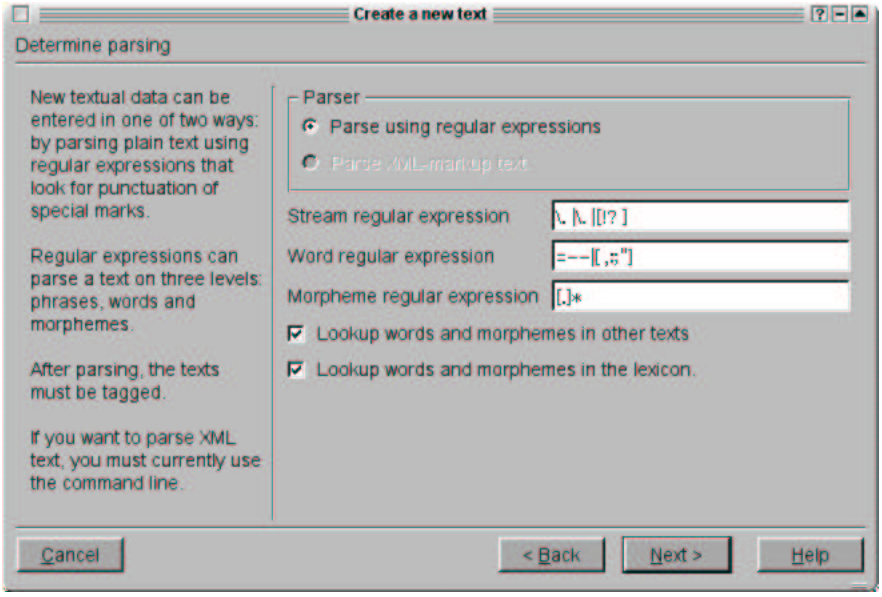
The language is the basic language of the text: individual phrases and words can be marked to be in another language.

Title Sumi
URL
Language Denden
Recording A remark about wine.
Scan <None>
Date 11-09-2002
Description A remark about wine.

First page of the New Text wizard.

The next window lets you determine the basic parameters used to parse the new text. Kura 2.0 can split plain text in words and sentences using common punctuation. The text import script can import ready-interlinearized text marked up in XML, but this dialog cannot handle that yet. I intend to create a mechanism where you can plug-in real parsers (probably stemmers and so on) for particular languages, but that's pie-in-the-sky at the moment.

You can also let Kura lookup glosses for words in the lexicon or in other texts. This is not quite foolproof, obviously, but it might give you a head-start.



Determine parsing

New textual data can be entered in one of two ways: by parsing plain text using regular expressions that look for punctuation of special marks.

Regular expressions can parse a text on three levels: phrases, words and morphemes.

After parsing, the texts must be tagged.

If you want to parse XML text, you must currently use the command line.

Parser

● Parse using regular expressions

○ Parse XML-markup text

Stream regular expression: \. |\. |[!?]

Word regular expression: =--|[,:;"]

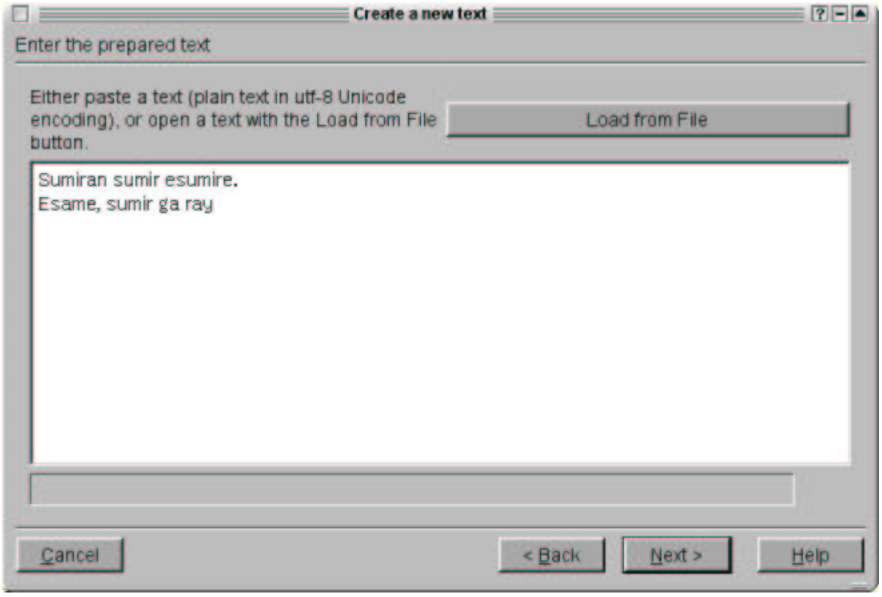
Morpheme regular expression: [.]*

☑ Lookup words and morphemes in other texts

☑ Lookup words and morphemes in the lexicon.

Parser parameters.

The third page of the new text wizard shows an editor where you can either type your text, or load the text from a file. The encoding of that file *must* be utf-8. Use an utf-8 capable editor, such as Kate(http://kate.kde.org) to create those texts.



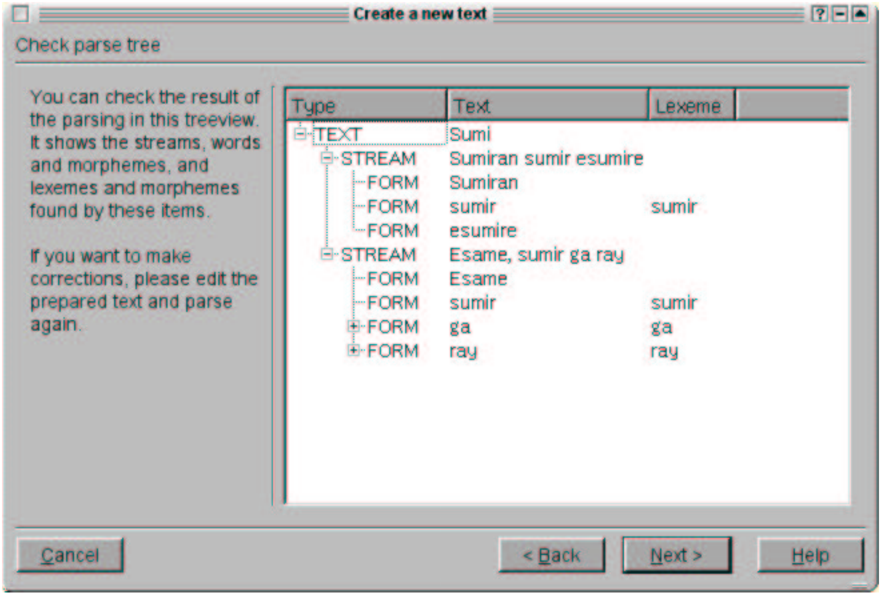
Enter the prepared text

Either paste a text (plain text in utf-8 Unicode encoding), or open a text with the Load from File button.

Sumiran sumir esumire.
Esame, sumir ga ray

A very short text.

If you press Next, the text will be parsed and shown in a tree:



Check parse tree

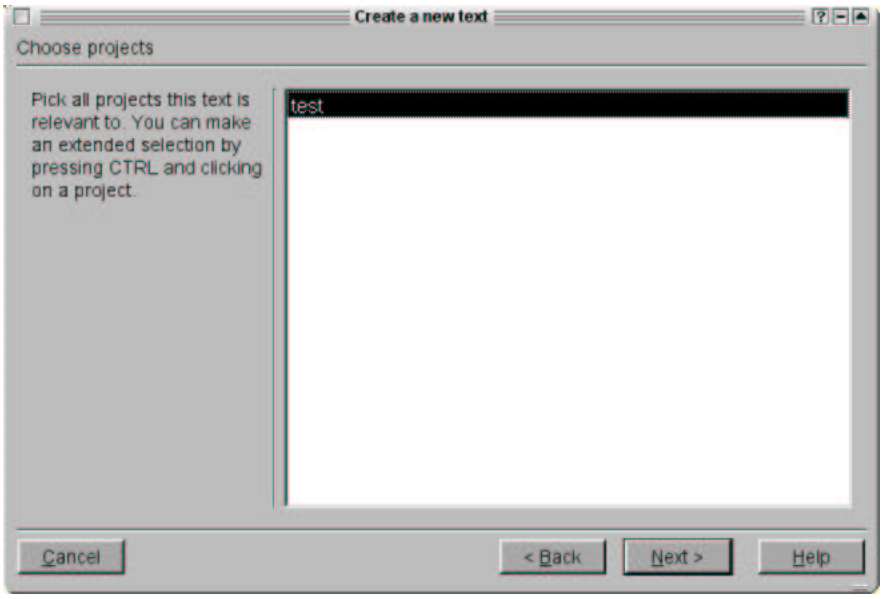
You can check the result of the parsing in this treeview. It shows the streams, words and morphemes, and lexemes and morphemes found by these items.

If you want to make corrections, please edit the prepared text and parse again.

Type	Text	Lexeme
⊟ TEXT	Sumi	
⊟ STREAM	Sumiran sumir esumire	
FORM	Sumiran	
FORM	sumir	sumir
FORM	esumire	
⊟ STREAM	Esame, sumir ga ray	
FORM	Esame	
FORM	sumir	sumir
⊞ FORM	ga	ga
⊞ FORM	ray	ray

A parse tree.

If you think the result is wrong, you can go back and fiddle with the regular expressions to get the right result. If the result is right, press next to pick the projects this text is associated with:



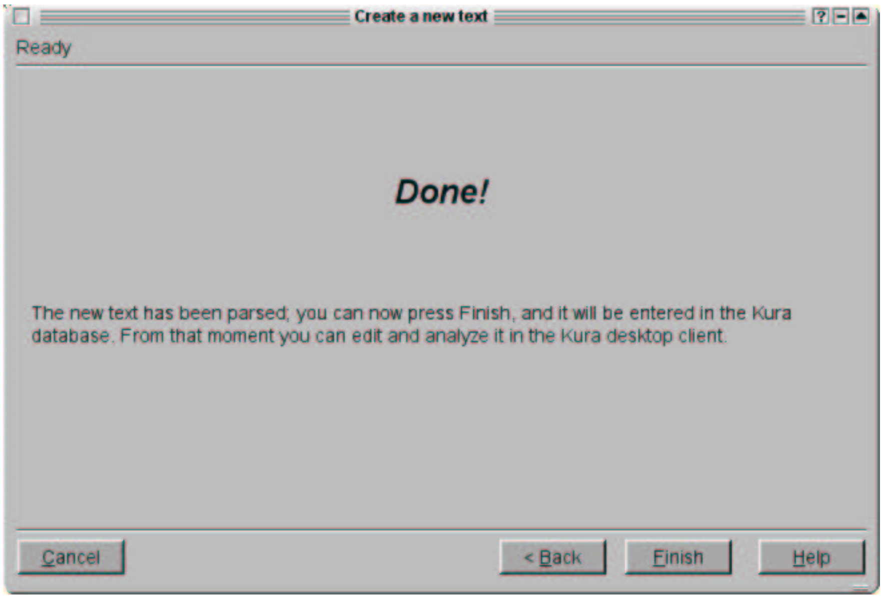
Choose projects

Pick all projects this text is relevant to. You can make an extended selection by pressing CTRL and clicking on a project.

test

Picking the project.

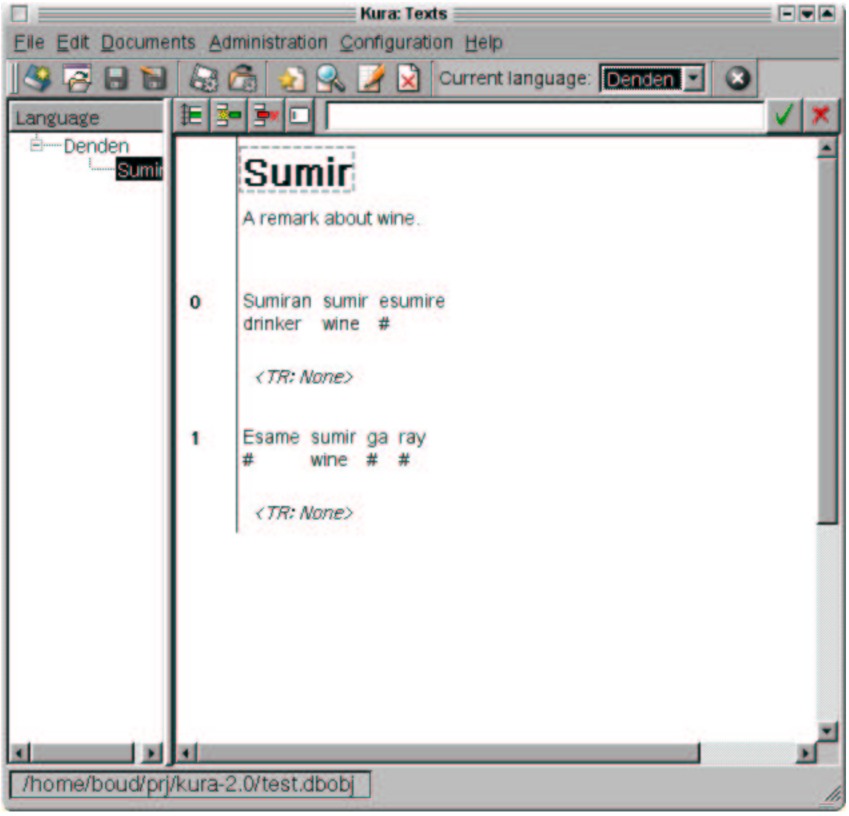
And that was the last step before the finish:



Done!

The new text has been parsed; you can now press Finish, and it will be entered in the Kura database. From that moment you can edit and analyze it in the Kura desktop client.

If you select the new text in the text tree, you will see the interlinear text you've just created, with some glosses already:



Sumir

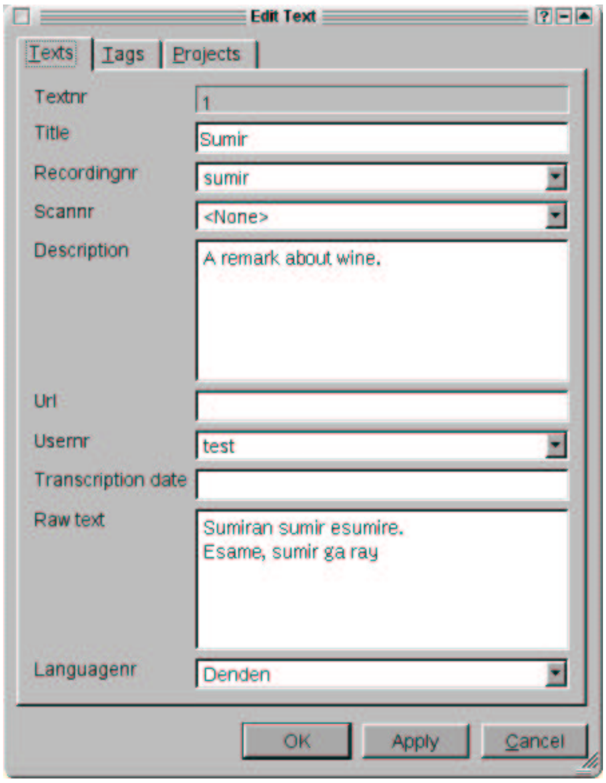
A remark about wine.

```
0    Sumiran  sumir  esumire
     drinker  wine   #

      <TR: None>

1    Esame  sumir  ga  ray
     #      wine   #   #

      <TR: None>
```

An interlinearly glossed text.

Working with interlinear texts is quite simple and intuitive. The currently selected element is enclosed in a gray box. The cursor keys move the selection. When you move the cursor, the text in the edit box in the interlinear toolbar is updated. If you press enter or click in the edit field or click on the edit button, the edit field get focus and you can change the contents. If you then press the button with the green OK sign, or if you press Enter, the element will be updated.

Double clicking on an element opens a dialog window where you can enter more detailed data. For instance, if you click on the title of the text, the dialog window that allows you to annotate the text appears:

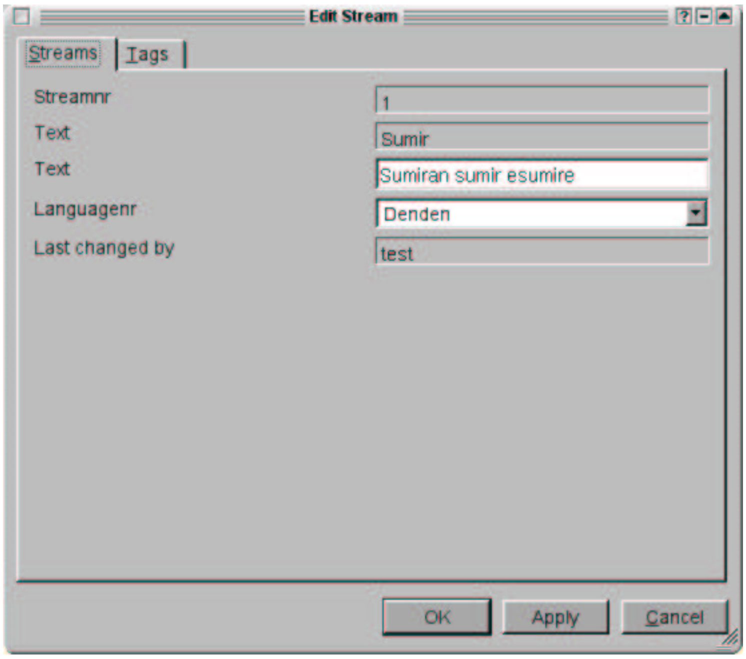


Textnr: 1
Title: Sumir
Recordingnr: sumir
Scannr: <None>
Description: A remark about wine.
Url:
Usernr: test
Transcription date:
Raw text: Sumiran sumir esumire.
Esame, sumir ga ray
Languagenr: Denden

Text dialog.

Here, you can add text-specific tags, or review the associations of the text with the projects. Below the title, there's an element that represents the description you entered in the text dialog. Clicking on this, too, opens the text dialog.

The phrases, or streams in Kura terminology are neatly numbered. Base-zero, I'm afraid. I only noticed just now... Put it down to job deformation. Anyway, clicking on this number opens the dialog window that allows you to enter all kinds of interesting information related to streams:



Streamnr 1

Text Sumir

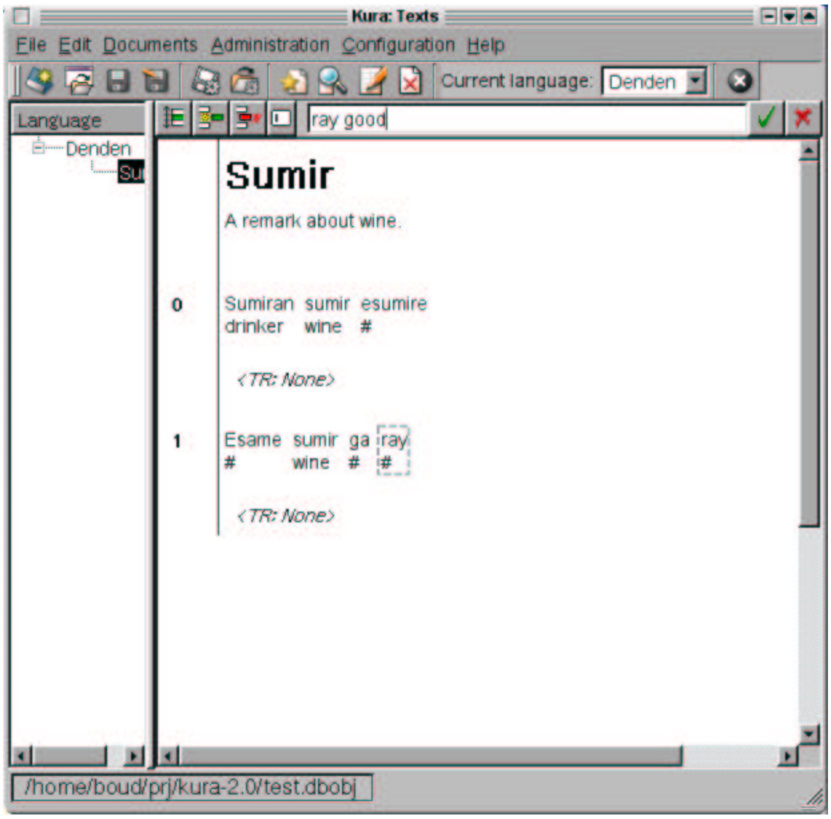
Text Sumiran sumir esumire

Languagenr Denden

Last changed by test

Stream dialog.

The words that make up a stream are one element, together with their glosse. Actually, the gloss is a tag record associated with the element, but that’s not that important. If you use the edit field above the interlinear display to edit the word and its gloss, you need to be aware that Kura splits at the first space: the text up to the first space is the form of the word, everything after that is considered a glosse.



ray good

Sumir

A remark about wine.

0 Sumiran sumir esumire
drinker wine #

<TR: None>

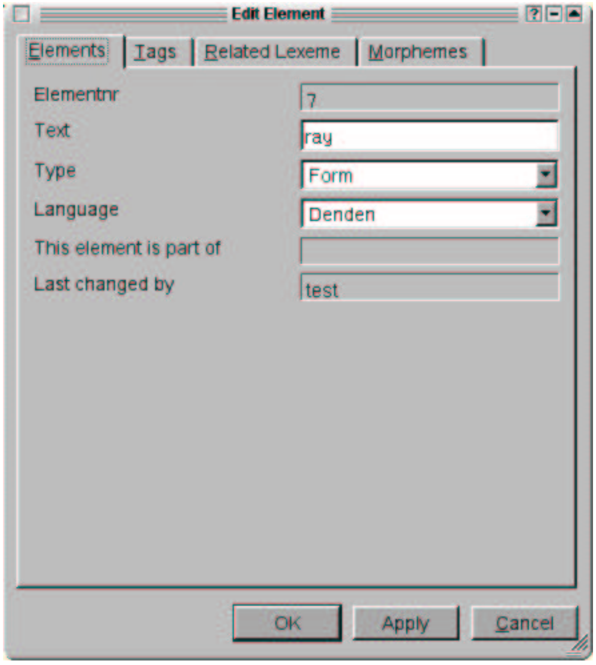
1 Esame sumir ga ray
# wine # #

<TR: None>

/home/boud/prj/kura-2.0/test.dbobj

Using the edit field of the interlinear text editor.

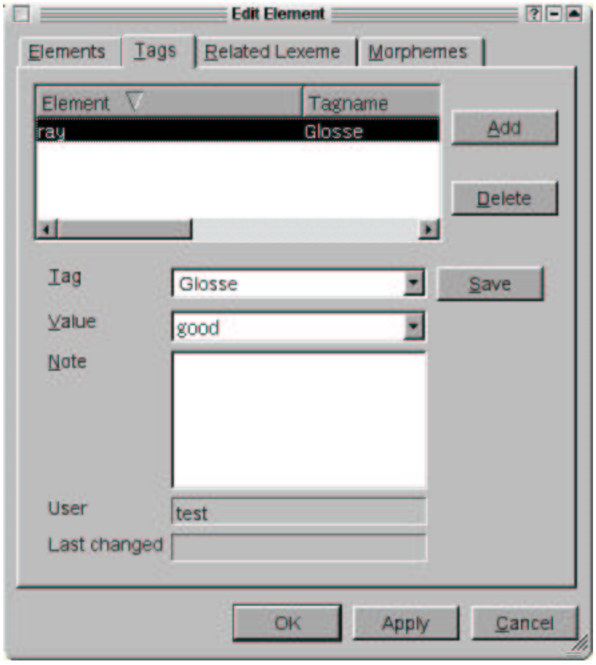
Let’s use the edit field to gloss the word *ray* in the second stream (the one with number one). Go there with the cursor, or click on that word. Press enter. You see that the word *ray* is glossed with a hash sign (or pound sign, your preference). Replace the hash sign with the glosse ’good’. Press Enter. You see that the gloss in the interlinear text has been changed to ’good’. Now double-click on that element. You get the following dialog window:



Elementnr: 7
Text: ray
Type: Form
Language: Denden
This element is part of:
Last changed by: test

Element dialog.

If you now go to the Tags tab, you see that the Glosse tag has been entered:

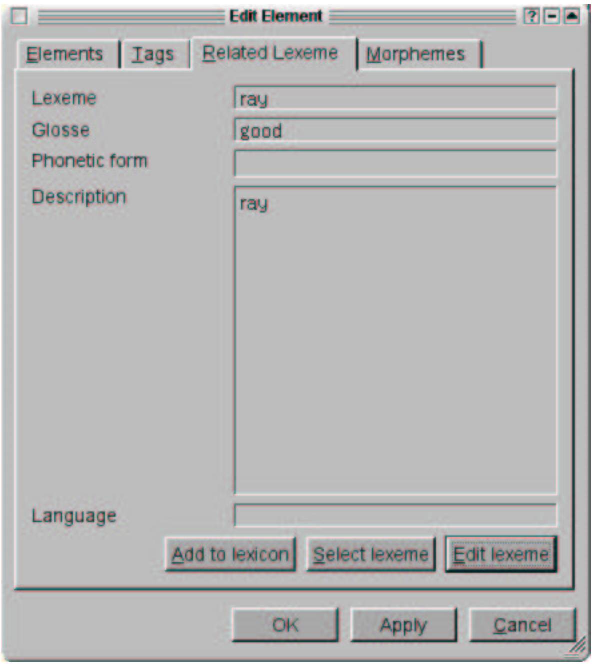


Element	Tagname
ray	Glosse

Tag: Glosse
Value: good
Note:
User: test
Last changed:

Element dialog.

It’s easy to create a lexeme from an element in a text; it’s just as easy to associate an existing lexeme with an element in a text. Go to the Related Lexeme tab. It’s empty now, but if you press the Add to lexicon button, Kura asks you whether you really want to add this word to the lexicon. And if you say yes, it’s done:

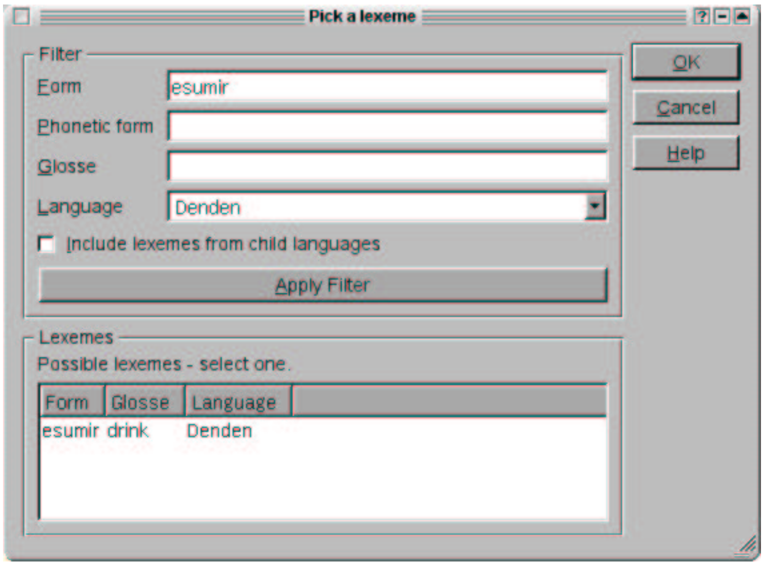


Lexeme: ray
Glosse: good
Phonetic form:
Description: ray
Language:

Adding a lexeme to the dictionary.

If you press Edit lexeme, an entirely familiar dialog pops up, namely, the Edit Lexical Item dialog we've seen before.

Press OK, and now double-click on the element *esumire*. We're going to associate this element with a lexeme, and then split the element into its component morphemes. First go to the Related Lexeme tab again. Press Select Lexeme. A dialog that looks a lot like the one we used to relate lexemes to each other pops up, with some data already filled in:



Form: esumir
Phonetic form:
Glosse:
Language: Denden
□ Include lexemes from child languages

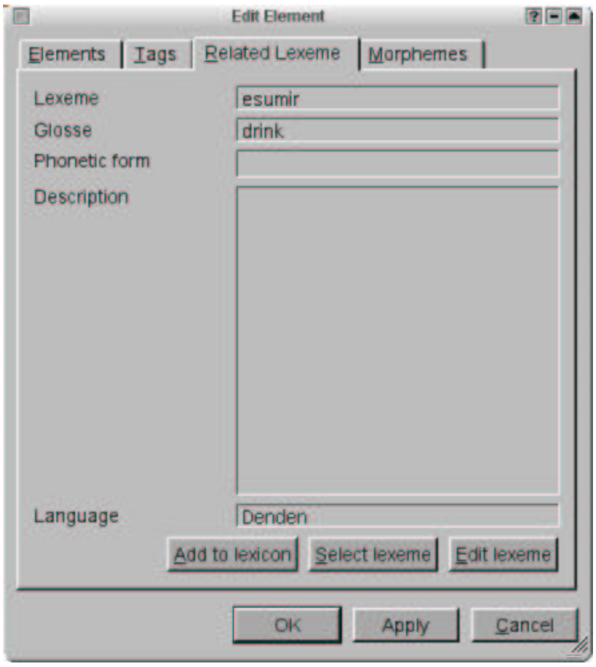
Possible lexemes - select one.

Form	Glosse	Language
esumir	drink	Denden

Picking a lexeme from the dictionary.

Remove the final 'e' from the Form (that's the perfective suffix), and press Apply

Filter. The Lexemes list is now filled with lexemes that conform to the search requirements. Select one. (In this case, there's only one, obviously.)



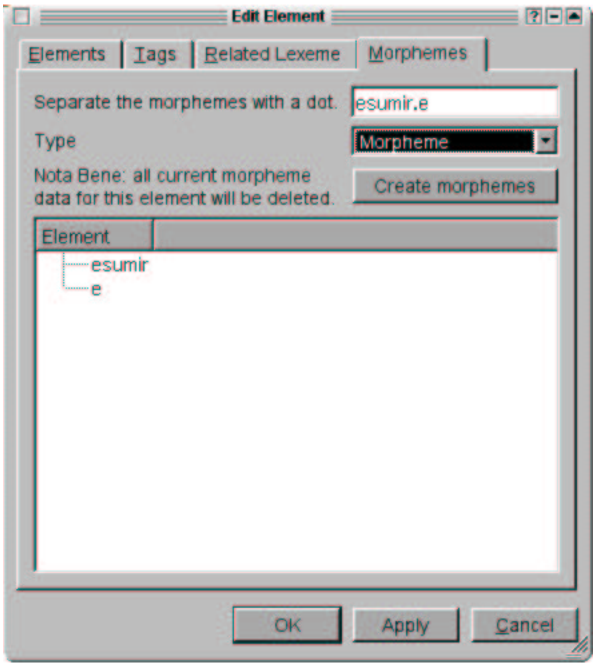
Lexeme: esumir
Glosse: drink
Phonetic form:
Description:
Language: Denden

A lexeme from the dictionary has been associated with this element.

If you press OK after selecting a lexeme, the link has been made. What about the lexemes Kura already glossed for us? Are they linked to the relevant lexemes, too? The answer is, alas, not really. When glossing a text, for every element in a stream that is not associacated with a glosse or a lexeme, Kura looks in the lexicon of that language for any forms that are exactly the same as the form of the element, and picks one. It could be entirely wrong, but that's not likely in our test database. But the process is not nearly accurate enough to allow it to chisel the results in stone.

The form *esumire* is complex, and consists of a root and a suffix. Let's use the Morphemes tab to enter that data.

You separate the morphemes in a word with a dot, select the type of sub-elements you want to create (most like morpheme), and then press Create Morphemes. Now kura will fill the tree in the bottom half of the dialog window with the created morphemes.



Separate the morphemes with a dot. esumir.e

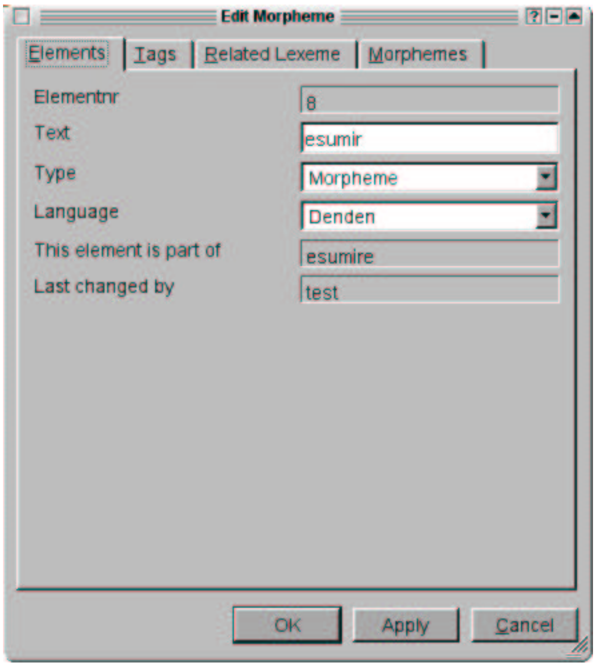
Type Morpheme

Nota Bene: all current morpheme data for this element will be deleted.

Element
- esumir
- e

A word split into two morphemes.

Double-clicking on a morpheme will open a new dialog, similiar to the Edit Element, but titled Edit Morpheme. Now you can repeat the preceeding steps at libitum: add tags, associate with lexemes or split the morpheme into phonemes.



Elementnr 8

Text esumir

Type Morpheme

Language Denden

This element is part of esumire

Last changed by test

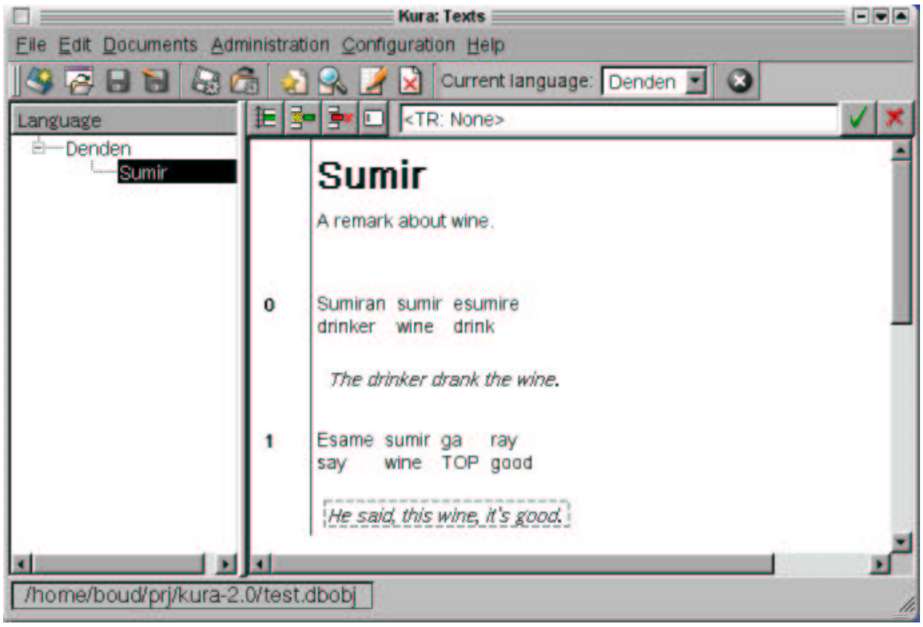
Editing a morpheme.

Now use Add Lexeme to add the root of *Esame,* which is *esam* to the lexicon with the glosse ’to say’, and the particle *ga* with the glosse ’TOPIC’, and the relevant abbrevation and part-of-speech particle. (In order to be able to tag the abbreviation

TOP to *ga*, you need to add it to the list of tag domains.)

Finally, we're going to add translation tags to the two streams. Simply click on the fields that say *TR:None*, and enter for the first line 'The drink drank the wine' and for the second line, 'He said, this wine, it's good'.

Now you should have a neatly glossed text (if some glosses don't appear, click on the text again, there might still be a few bugs left in the redisplay code).



Sumir

A remark about wine.

```
0   Sumiran  sumir  esumire
    drinker  wine   drink

     The drinker drank the wine.

1   Esame  sumir  ga   ray
    say    wine   TOP  good

     He said, this wine, it's good.
```

/home/boud/prj/kura-2.0/test.dbobj

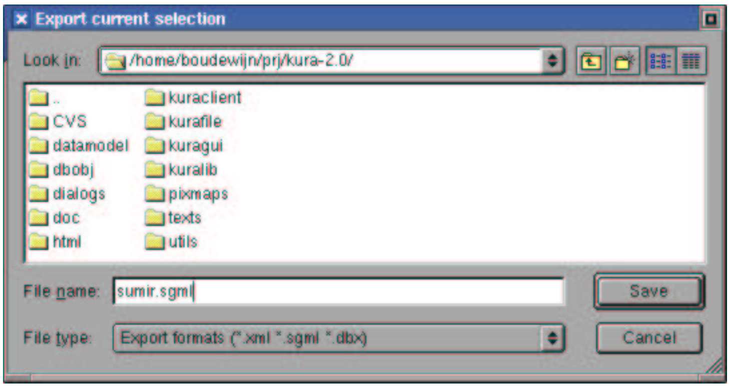
Finished glossing a text.

Chapter 8. Creating a document.

The goal of the whole exercise is, of course, to create publications of the analyzed data. In this part of the tutorial we will create a documentat that contains an interlinearized text that can be published to html or PDF.

Open the Text view and select our text. Then choose File/Export As. A file dialog opens. You can see from the list of supported filetypes that Kura can export data in sgml, xml and dbx format. Those formats are all docbook: if you want the data in another format, you should consult the chapter on hacking Kura.

Type a filename, say, `sumir.sgml`, and press Enter.



Export current selection

Look in: /home/boudewijn/prj/kura-2.0/

File name: sumir.sgml

File type: Export formats (*.xml *.sgml *.dbx)

Exporting a text.

Now Kura has created an sgml docbook fragment that contains your text. The interlinear format is simple: just lines spaced out with spaces in a programlist docbook tag.

You can insert this fragment in any docbook document and then use your operating system tools, like OpenJade to generate documents.

```
db2pdf sumir.sgml
```

And the result will be:

```
      sectiontitleDenden/title
        section id="text_1"titleSumir/title
          informaltable frame="none" colsep="0" rowsep="0"tgroup cols="2"t
                row
                  entryemphasis role="strong"Recording/emphasis/entry
                  entrylink linkend="recording_1"sumir/link/entry
                /row
              /tbody/tgroup/informaltable
          paraA remark about wine./para
          para id="stream_1"
```

```
            programlistingSumiran sumir esumire
drinker wine  drink   /programlisting
The drinker drank the wine.
          /para
          para id="stream_2"
            programlistingEsame sumir ga  ray
say   wine  TOP good /programlisting
He said, this wine, it’s good.
          /para
        /section
      /section
```

You can do the same with the lexicon. More complex formatting or inclusion of texts and lexical items in grammars or articles is offered by the grammar template project, which is a combination of scripts that handle the Kura database and docbook sources. Notice that handling unicode characters out of the box (like the IPA we used) is not OpenJade’s forte(http:/www.valdyas.org/linguistics/printing_unicode.html). Generating to html is no problem, and the grammar project uses Fop to create unicode documents.

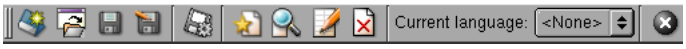
You can read all about docbook at: docbook.org(http://www.docbook.org), where you can also find a complete book on docbook.

III. Functionality

Table of Contents

The functionality of the Kura gui client is divided over several menus. This part discusses the contents of those menus.

Chapter 9. Toolbar



Kura currently sports a very simple toolbar. From left to right, the buttons represent: New Datastore, Open Datastore, Save Datastore, Save Datastore As, Export, New Item, Find Item, Edit Item, Delete Item.

The Current Language Combobox is used to quickly set the language the user is currently working with. This value is used in searches, if the Use Default Values in Search preference is set.

Note that there are currently unresolved issues with this feature: selecting a new language after changing between datastores or between datastores and databases causes Kura to crash.

The final button can be used to quit Kura.

Chapter 10. File Menu

The contents of the File Menu are related to accessing databases or datastore files, creating external files, opening new windows and setting the preferences that are stored in the `$HOME/.qt/kurarc` file.

10.1. New Window

Opens a new Kura window on the *same* database or datastore file as the first window. That way, you can compare different queries or add, say, recordings in one window while adding texts in another.

10.2. New

Shortcut:CTRL-N. Creates a new datastore file. Datastore files should have the extension `.dbobj` and are created based on the template file in `/usr/local/share/kura/template.dbobj`.

10.3. Open

Shortcut:CTRL-O. Opens an existing datastore file. The file should have the extension `.dbobj`. Kura remembers the last opened datastore file and will open that file automatically when starting the GUI client. It is not (yet) possible to start kura with a filename as a command-line argument.

Please be aware that file-based backed is strictly single user, while the the MySQl database backend is multi-user. If more than one instance of Kura accesses a file, it *will* be corruped. I have *not* implemented a locking mechanism. If you want it, use the database.

10.4. Save

Shortcut:CTRL-O. Saves the currently open datastore file.

10.5. Save As

Saves the currently open datastore in a new file with a new name. Closes the currently open datastore file and makes the new file the default.

10.6. Export As

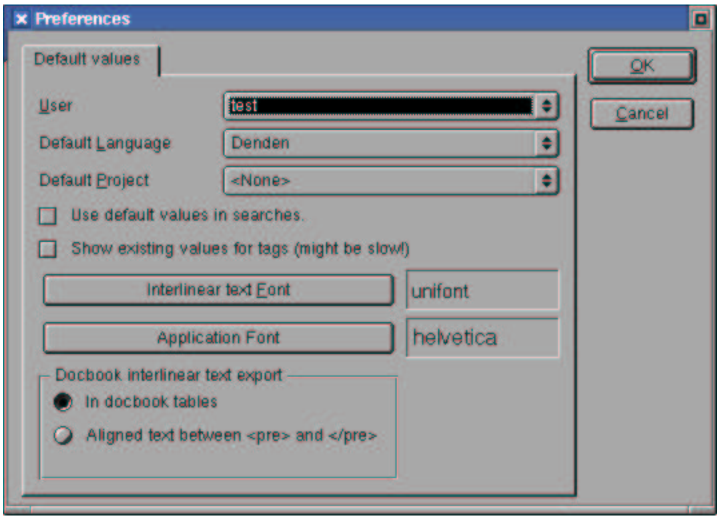
Saves the currently active text, or the current selected set of lexemes, recordings, scans or bibliographic references in docbook sgml or docbook xml format.

10.7. Connect

Connects to a MySQL database. If a datastore file is open, it is saved and closed. The connection to the database is made the default and will be restored the next time Kura is started. Please be aware that the MySQl database backend is multi-user, while the file-based backed is strictly single user. If more than one instance of Kura accesses a file, it *will* be corruped. I have *not* implemented a locking mechanism. If you want it, use the database.

10.8. Preferences

Opens the preferences dialog window:



User test

Default Language: Denden

Default Project <None>

□ Use default values in searches.

□ Show existing values for tags (might be slow!)

Interlinear text Font: unifont

Application Font: helvetica

Docbook interlinear text export

● In docbook tables

○ Aligned text between <pre> and </pre>

Preferences dialog.

The User combobox sets the default user. This is the username that Kura will use to tag new and changed records with. In a multi-user situation, this will enable a form of academic accountability by making clear which linguists did what.

The default language is the language the user works most with. If the checkbox Use Default Values in Searches is checked, this will be used to, for instance, filter the results of a query on the lexicon.

The default project is the current project the user works with. When creating new items or querying, this value can be taken into account, just like language or user.

The checkbox Use Default Values in Searches controls Kura's query behaviour. If it is checked, Kura will pre-filter the results of queries on tables, restricting the results to those that have the user, language and project chosen above.

Show Existing Values in Tags tells you also that it might be slow. That is true. The case is that for some tags, such as glosse in element tags, you enter short texts. These texts might reoccur often, so if you check this, you can choose from the values you entered before.

The interlinear text font is the font used to create the text in the interlinear text editor.

The application font is the font Kura uses for all the other texts it shows, in menu's, on buttons and in edit fields.

Finally you have the choice between two options related to the generation of sgml export files of texts. You can either choose to generate the interlinear lines aligned with spaces in a preformatted block, or to generate the interlinear lines in the form of Docbook tables. Both are not very satisfactory, I'm afraid. You can always write your own TeX output with the one of the classes LaTeX offers for interlinear examples.

10.9. Exit

Closes all open windows and exits Kura completely. If you’ve changed records in the datastore file, Kura will ask you whether you want to save the changes. Be aware that if you use the MySQL backend all your changes are made immediately, and are undoable.

Chapter 11. Edit Menu

The Edit menu concerns itself with the handling of individual records in the Kura tables.

11.1. New Item

Shortcut: CTRL-ALT-N, INS. New Item is used to create a new entry in the currently active table. If you are currently editing texts, it will start the New Text Wizard

11.2. Find

Shortcut: CTRL-F, F2. The Find option opens a dialog window you can use to retrieve a filtered set of data in the database or the2 datastore.

If you have selected Use Default Values in searches in the Preferences dialog you current project, language and user will already be filled in.

You can search for part of a word using the % wildcard. For instance, ’%are’ in glosse matches all lexemes where the glosse ends in ’are’, ’a%’ matches all lexemes where the glosse starts with an *a*, and ’a%a’ matches all lexemes where the glosse starts and ends with an a. It is possible that the way the match is made differs subtly between the database and the text backend.

11.3. Open Item

Shortcut: CTRL-ALT-O, Enter This action opens the currently selected line (or record) in a dialog box, and allows you to change data.

11.4. Delete

Shortcut: DEL. The delete action removes an item irretrievably from the database.

If there are records dependend upon this item, Kura will ask you whether it should remove those, too. If you say yes, that means that, removing a lexical item that has been associated with an element in a text, will also remove that element from the text, and all associated data. I probably won’t have to point out the danger of this operation to you.

If you use the database backend, the action is carried out immediately, if you use the file datastore backend, you can back out of your changes by not saving the file. In that case you would, of course, lose all your other changes.

Chapter 12. Documents

The Documents menu contains the options that show the real data Kura is all about: lexemes, recordings, manuscript scans and bibliographic references.

12.1. Lexicon

Table: lng_lex

The lexicon view shows a long list with all currently selected lexemes. You can filter the lexemes by creating a different selection using the Find option, or by choosing a language in the language combobox in the toolbar.

Using the Kura lexicon has been described in the tutorial.

12.2. Recording

Table: lng_recording

The table with recordings points, in the form of url's to sound files. Note that the idea of an url is not limited to http addresses: and url can just as well point to a local file, an ftp file or any of dozens other protocol dependent location. So if you make your recordings public in a free format, like ogg or realaudio, you can point your readers to those files from you analysises.

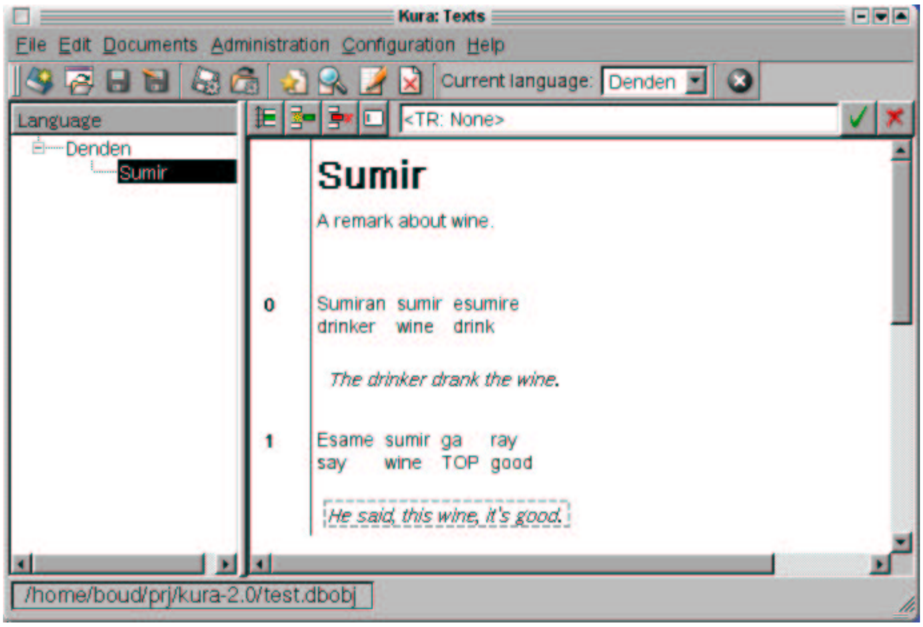
12.3. Scan

Table: lng_scan

The table with scans contains pointers similar to those in the recordings table to images. The idea is to scan in the priceless manuscripts you've transcribed to analyze the texts, and make them available when you publish you analysis.

12.4. Text

Table: lng_text



Sumir

A remark about wine.

```
0  Sumiran  sumir  esumire
   drinker  wine   drink

   The drinker drank the wine.

1  Esame  sumir  ga   ray
   say    wine   TOP  good

   He said, this wine, it's good.
```

/home/boud/prj/kura-2.0/test.dbobj

Text view

The text view is more complex than the others. On the left side, there is a tree of languages and texts. Currently, this list is always filled with all languages and text. You cannot filter here.

The text view toolbar buttons have the following functions: Insert new stream, insert new element, remove currently selected item, edit current item, edit text, apply changes, cancel changes.

In the interlinear text, you can move using the cursor keys, home and end. Moving around in the text or clickin on an item puts the contents of that item in the edit field. Pressing Enter makes the edit field active. Pressing Enter again, or clicking on the apply button applies your changes to the text. Double-clicking on an item opens a detailed dialog window for that item.

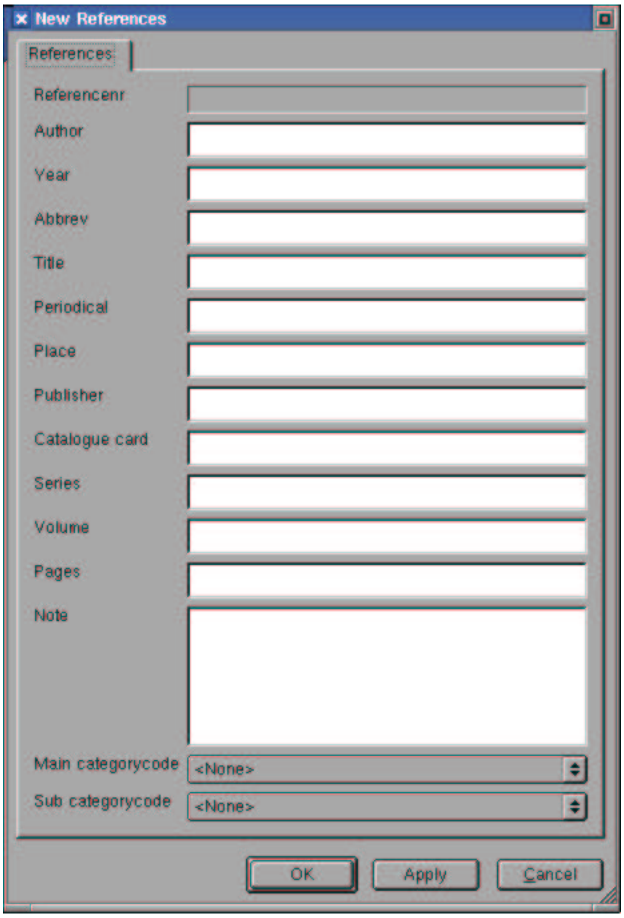
The interlinear text view uses the *TR* tag for the translations of the streams and the *GL* tag for the glosse of the individual elements. The elements shown are of the type *FORM*.

In contrast with Kura 1.2, it is no longer possible to alter these settings except through hacking the code. On the other hand, the interlinear editor now works.

12.5. References

Table: lng_reference

The references table is not a full substitute for a bibtex database, but it can be useful if you want to generate documents using the Kura database that include bibliographic references. References are divided into a main category and subdivided into subcategories. You can create new categories in the Configuration/Reference Categories table.



Main categorycode <None>

Sub categorycode <None>

Text view

You can also link references to texts, streams, elements or lexemes by choosing the REF tag and picking one from this list.

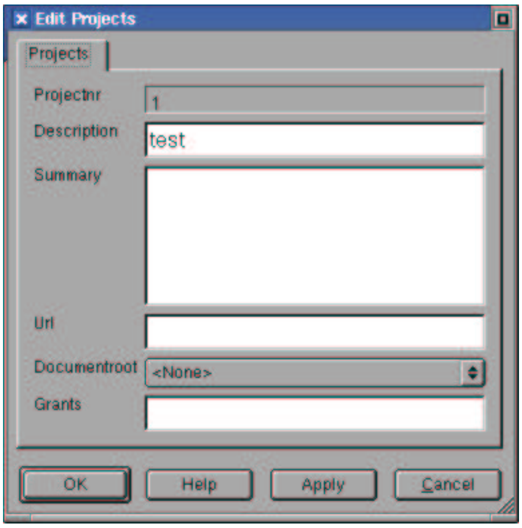
Chapter 13. Adminstration

The Administration menu contains administrative tables that determine the languages, users and projects that Kura knows about. The contents of these tables are not used by the Kura code directly, so changing things here doesn't break much.

13.1. Project

Table: lng_project

Work in Kura is organized by project. Projects can comprise the writing of an article, or the compilation of a complete dictionary. Projects have a description, a summary and an URL for the project webpage. The document the project produces can be chosen here, too. Finally, abbreviations for eventual grants can be entered in the grants field.



Projectnr	1
Description	test
Summary	
Url	
Documentroot	<None>
Grants	

The test project again.

13.2. Language

Table: lng_language

Kura, of course, deals with languages. The registration of meta-data about languages is fairly simplistic in this release of Kura, and not extensible. For instance, preferred script, character set and sort order are not possible.

13.3. User

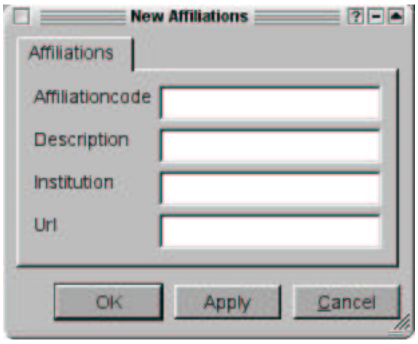
Table: lng_user

Kura uses the user table to keep track of who has last edited which record.

13.4. Affiliations

Table: lng_affiliationcode

The affiliations view stores the affiliations you can associate with certain user. As you can see, it’s the simplicity itself.

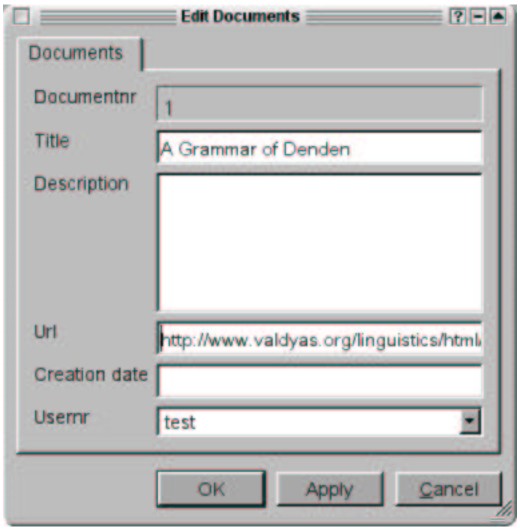


The Affiliations dialog.

13.5. Document

Table: lng_document

This is a list of documents you can choose from when you associate a root document with a project.



Documentnr: 1
Title: A Grammar of Denden
Description:
Url: http://www.valdyas.org/linguistics/html
Creation date:
Usernr: test

The documents dialog.

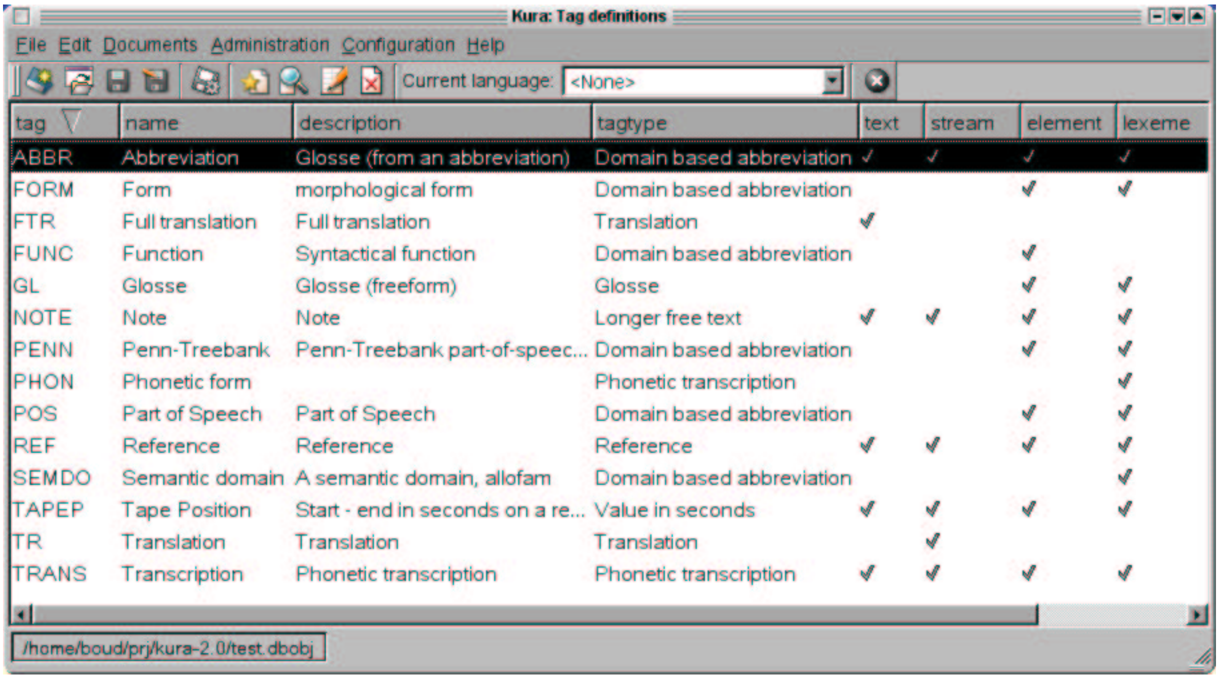
Chapter 14. Configuration

The configuration tables offer the possibility of expanding Kura without changing code. You are free in what you add. However, you need to be aware that *deleting* records can create big problems since Kura uses some values in the code itself. And if you first add an item, use it in your project, and then try to delete it, you might delete all dependent data, too, in one fell swoop. Take care, plan ahead, and have fun.

14.1. Tags

Table: lng_tag

Tags are the definitions of the kinds of annotations you can add to texts, streams, elements and lexemes. Tags themselves have a certain type, that determines whether the value of the tag must be chosen from a limited set, whether the tag allows for a longer free-form text, or a shorter text or whether the tag should present a picklist of defined bibliographic references.



Current language: <None>

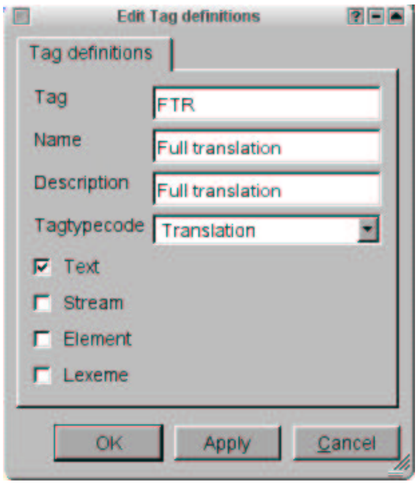
tag	name	description	tagtype	text	stream	element	lexeme
ABBR	Abbreviation	Glosse (from an abbreviation)	Domain based abbreviation	✓	✓	✓	✓
FORM	Form	morphological form	Domain based abbreviation			✓	✓
FTR	Full translation	Full translation	Translation	✓			
FUNC	Function	Syntactical function	Domain based abbreviation			✓	
GL	Glosse	Glosse (freeform)	Glosse			✓	✓
NOTE	Note	Note	Longer free text	✓	✓	✓	✓
PENN	Penn-Treebank	Penn-Treebank part-of-speec...	Domain based abbreviation			✓	✓
PHON	Phonetic form		Phonetic transcription				✓
POS	Part of Speech	Part of Speech	Domain based abbreviation			✓	✓
REF	Reference	Reference	Reference	✓	✓	✓	✓
SEMDO	Semantic domain	A semantic domain, allofam	Domain based abbreviation				✓
TAPEP	Tape Position	Start - end in seconds on a re...	Value in seconds	✓	✓	✓	✓
TR	Translation	Translation	Translation		✓		
TRANS	Transcription	Phonetic transcription	Phonetic transcription	✓	✓	✓	✓

/home/boud/prj/kura-2.0/test.dbobj

Default tags.

Tags have an abbreviation, a full name, a description, a type, and can be selected for texts, streams, elements and lexemes. Some kinds of tags, like Penn-Treebank parts of speech are inappropriate for texts, for instance, others, like reference works are applicable to everything.

Please, take care. Do not, never, remove the GL, ABBR or TR tags. Kura will cease functioning.



Tag: FTR

Name: Full translation

Description: Full translation

Tagtypecode: Translation

☑ Text

□ Stream

□ Element

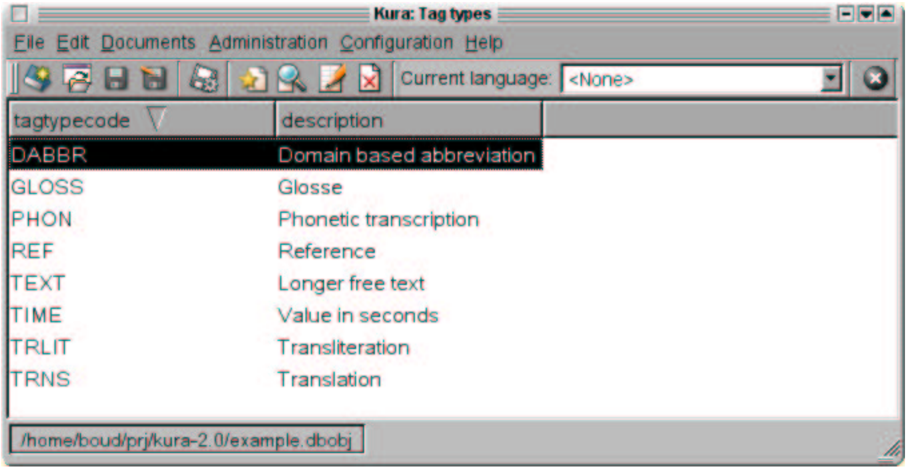
□ Lexeme

Default tags.

14.2. Tag Categories

Table: lng_tagtypecode

As said in the previous section, tags are defined to have a certain type. These types are defined in this table. The abbreviation and description are not really necessary, but it is nice to be able to distinguish tags in a more meaningful way.



Current language: <None>

tagtypecode	description
DABBR	Domain based abbreviation
GLOSS	Glosse
PHON	Phonetic transcription
REF	Reference
TEXT	Longer free text
TIME	Value in seconds
TRLIT	Transliteration
TRNS	Translation

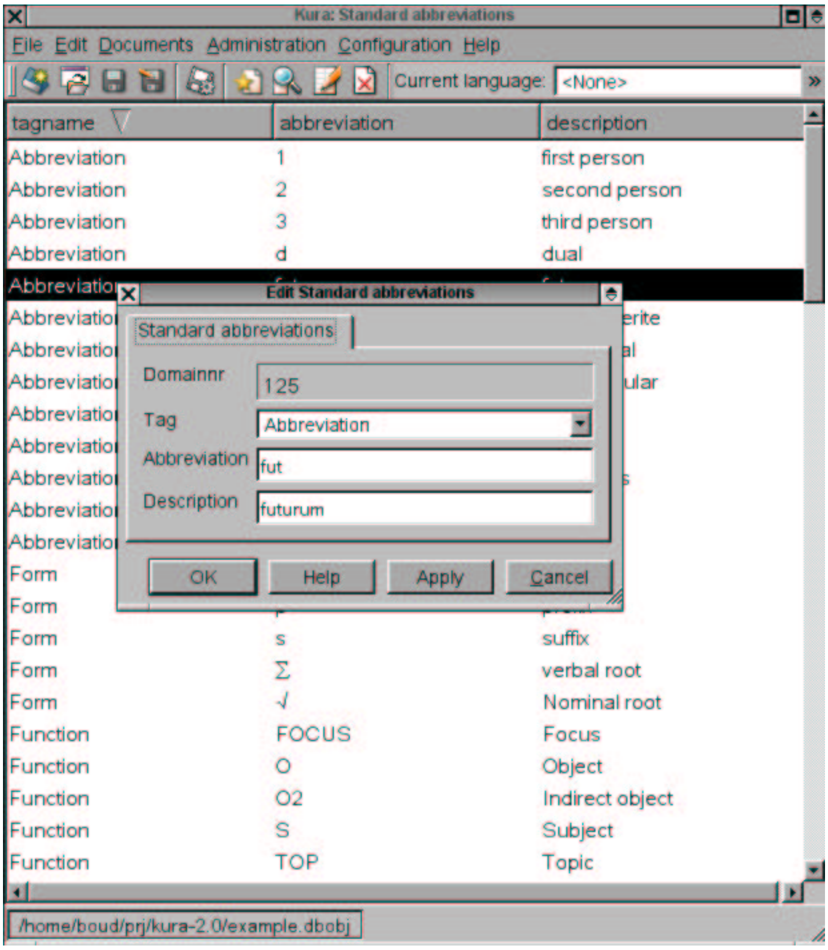
/home/boud/prj/kura-2.0/example.dbobj

Tag types.

14.3. Tag Domains

Table: lng_tagdomain

Certain tags can be defined to be of the type ’DOMAIN’. That means that there will be a fixed list of choices when you want to add such a tag to a record. This table is where those fixed choices are defined. Examples are abbreviations or syntactical functions that you want to be the same everywhere.


Kura: Standard abbreviations
File Edit Documents Administration Configuration Help
Edit Standard abbreviations
Standard abbreviations
OK
Help
Apply
Cancel

Current language: <None>

tagname	abbreviation	description
Abbreviation	1	first person
Abbreviation	2	second person
Abbreviation	3	third person
Abbreviation	d	dual
Abbreviatio		
Abbreviatio		erite
Abbreviatio		al
Abbreviatio		ular
Abbreviatio		
Abbreviatio		
Abbreviatio		s
Abbreviatio		
Abbreviatio		
Form		
Form		
Form	s	suffix
Form	Σ	verbal root
Form	√	Nominal root
Function	FOCUS	Focus
Function	O	Object
Function	O2	Indirect object
Function	S	Subject
Function	TOP	Topic

Domainnr: 125

Tag: Abbreviation

Abbreviation: fut

Description: futurum

/home/boud/prj/kura-2.0/example.dbobj

Tag domains.

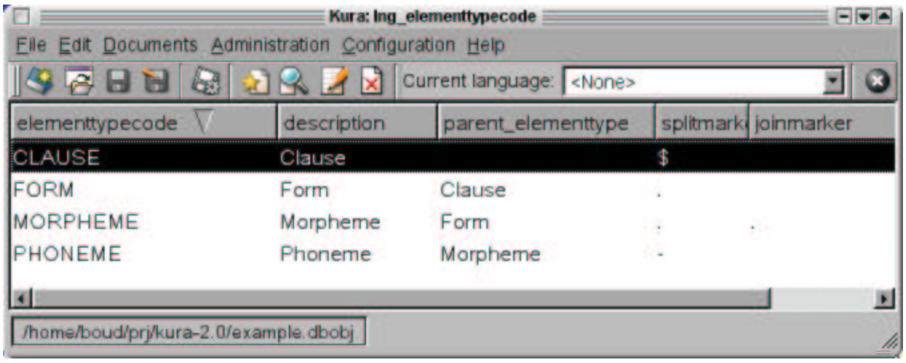
14.4. Element types

Table: lng_elementtypecode

Element types are in principle a closed category. Kura uses the codes in this table to determine what to show you in interlinear text.

You can add to this category, but the only of elements that are directly shown are ’FORM’ elements. You can subdivide forms into ’MORPHEME’ and

’MORPHEME’ elements into ’PHONEME’ elements. The ’CLAUSE’ type elements are intended to divide sentences, which Kura calls ’streams’. However, use of this element is not fully supported in Kura 2.0.


Kura: lng_elementtypecode
File Edit Documents Administration Configuration Help
Current language: <None>

elementtypecode	description	parent_elementtype	splitmark	joinmarker
CLAUSE	Clause		$	
FORM	Form	Clause	.	
MORPHEME	Morpheme	Form	.	.
PHONEME	Phoneme	Morpheme	-	

/home/boud/prj/kura-2.0/example.dbobj

Element types.

The fields splitmarker and joinmarker are used to let Kura know where it can seperate elements of this type, and how it should join those elements for presentation purposes.

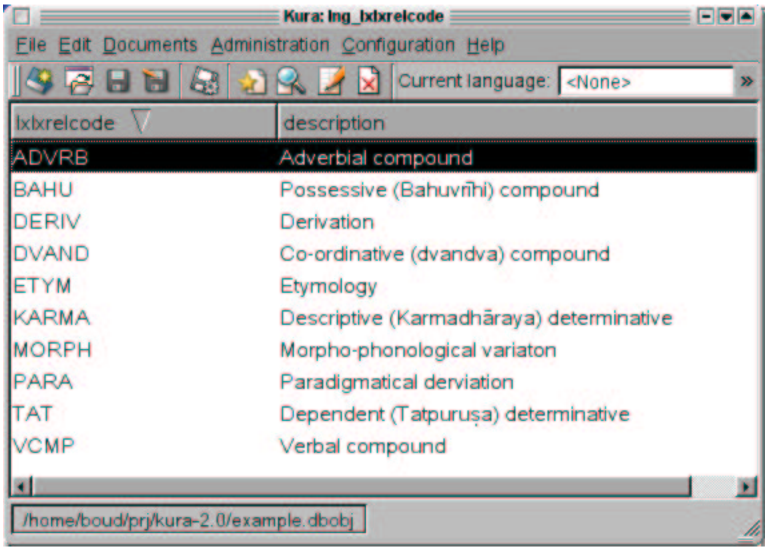
For instance, if you want to separate a form into morphemes, you would type dots at the morpheme boundaries, and then ask Kura to perform a split.

14.5. Lexical Relations

Table: lng_lxlxrelcode

Kura can store relations between lexemes in the database, and use those relations to produce cross-linked dictionaries, for instance in HTML. Using the menu opion ConfigurationLexical Relation, you can review the default types of lexical relations Kura offers.

This list is extensible, so if you need to add a new kind of compound, or a more specific kind of etymological relation, you can add them here.



lxlxrelcode	description
ADVRB	Adverbial compound
BAHU	Possessive (Bahuvrīhi) compound
DERIV	Derivation
DVAND	Co-ordinative (dvandva) compound
ETYM	Etymology
KARMA	Descriptive (Karmadhāraya) determinative
MORPH	Morpho-phonological variaton
PARA	Paradigmatical derviation
TAT	Dependent (Tatpuruṣa) determinative
VCMP	Verbal compound

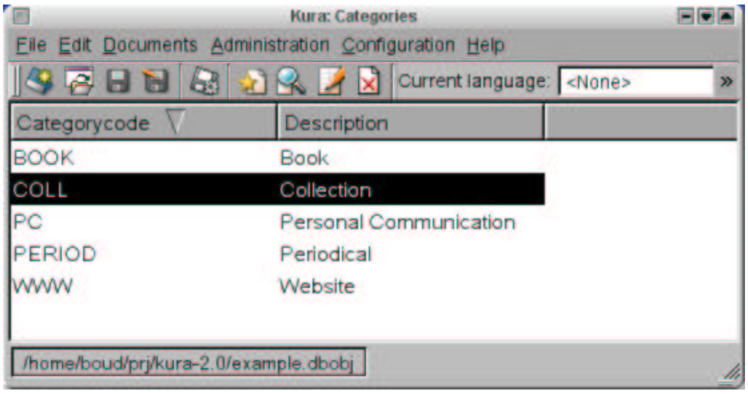
/home/boud/prj/kura-2.0/example.dbobj

Lexical relation types.

14.6. Reference categories

Table: lng_categorycode

Reference categories are simply kinds of bibliographic references. If you use these to define your references, Kura's presentation modules will know how to output the reference.



Categorycode	Description
BOOK	Book
COLL	Collection
PC	Personal Communication
PERIOD	Periodical
WWW	Website

/home/boud/prj/kura-2.0/example.dbobj

Lexical relation types.

IV. Extending Kura

Table of Contents

Kura is meant to be extensible. It is also meant to be used by field linguists, i.e. by people who think nothing of traipsing through sky-high mountains and enduring unspeakable discomforts in the pursuit of knowledge. So, I guess I can count on having determined, bloody-minded users of fair intelligence. So this isn't a 'for dummies' chapter. To get the most out of Kura you should start reading the Python tutorial at www.python.org(http:/www.python.org/doc/current/tut/tut.html).

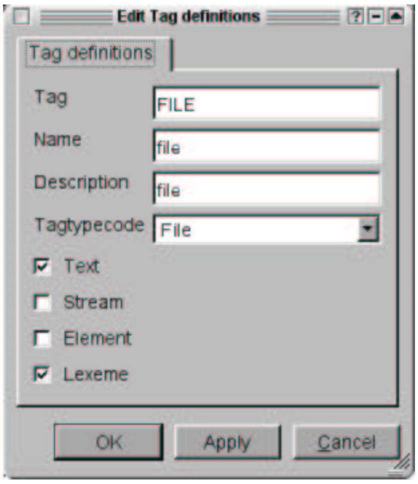
Warning

If you use the file datastore backend you should *NOT* keep Kura open while using the scripts. Your data *WILL* become corrupted. If you want to access your data from more than one program at the same time, use the MySQL database.

Chapter 15. Creating new tags

Adding a new tag is easy. Open the Configuration/>Tags screen, press INS and type away. But most likely you will find the default selection quite sufficient.

Anyway, in a short time we're going to import data into Kura from a file, and it would be nice to store the source of those records in a tag. Go, add a tag:



Tag: FILE
Name: file
Description: file
Tagtypecode: File
☑ Text
☐ Stream
☐ Element
☑ Lexeme

Adding a tag.

It will be useful in the next chapters.

It's more likely that you would want to add entries to the tag domains table: the lists that determines what choices a fixed-choice tag offers you. See the section on the tag domains table for that.

Chapter 16. Creating scripts that use the Kura database

It is very easy to script Kura. You use the Python(http:/www.python.org) language for that purpose. Python is quite easy to learn; there are many tutorials available, but most people who have at least some experience with programming will be able to grok the examples in this chapter without problems.

You will find the example scripts used in this section in the doc subdirectory of your Kura installation directory.

16.1. Opening the database and retrieving data

The first operation generally is to open the relevant database or datastore. Kura uses a central application object to access the data. That object, the KuraApp class can use different backends depending on whether you want to use a MySQL database or a file.

First you have to import the application object, then select a backend and then connect to the database or the file. You can either use the same default values as the Kura gui uses, or hack your own values in the script. Here, I use the gui defaults (guiConf) to determine whether to open a file or a database.

Example 16-1. Connecting to the database (script1.py)

```
import os.path, sys                                         ❶
from kuralib import kuraapp                                 ❷
from kuragui.guiconfig import guiConf                       ❸
from kuragui import guiconfig                               ❹

if guiConf.backend == guiconfig.FILE:                       ❺
    kuraapp.initApp(guiConf.backend,
                    dbfile = os.path.join(guiConf.filepath, ❻
                                          guiConf.datastore))
elif guiConf.backend == guiconfig.SQL:                      ❼
    if guiConf.username != "":
        try:
            kuraapp.initApp(guiConf.backend,
                            username = str(guiConf.username),
                            database = str(guiConf.database),
                             password = str(guiConf.password),
                             hostname = str(guiConf.hostname))

        except Exception, e:
            print "Error connecting to database: %s" % e
            sys.exit(1)

kuraapp.initCurrentEnvironment(guiConf.usernr,              ❽
                                   guiConf.languagenr,
                                   guiConf.projectnr)
```

❶ These are standard Python modules. `os.path` is used to join directory names and filenames in a platform independent way. `sys` is used here to provide for a way to exit the script when we couldn't connect to a database

❷ The central application object comes from the `kuralib` module. If the script complains that it cannot find this class, add the directory where Kura is installed to the PYTHONPATH environment variable. For instance:
`export PYTHONPATH=/usr/local/share/kura`

❸ This line imports the configuration settings of the Kura gui client. Using these settings, you know that your script will use the same database or datastore that Kura uses.

❹ The `guiconfig` module also contains some definitions we need to determine whether to use the file or the database, so we import it, too.

❺ This line determines whether Kura uses the file backend.

❻ If that is true, the script proceeds to create the right backend using the `kuraapp.initApp()` function. The first argument is the backend which should be used, the second argument the file to be opened. You could easily open another file by saying:
`kuraapp.initApp(guiconfig.FILE, "~/projects/myfile.dbobj")`

❼ If the backend didn't happen to be a file, this line will check if it is a database. If that's so, Kura proceeds to connect to the database. It is quite possible for that operation to fail, for instance if you forgot to start the MySQL server. In that case, an exception will be thrown, and the script will exit with the error code 1.

❽ Finally, some default values need to be set. The `kuraapp.KuraApp` class expects to know the default language, user and project you are working with. Whether that is actually important depends on the rest of the script.

Now that we have a working application object, we can use it to select a list of, say lexemes, and do something useful with it. In this example script, we will print a simple list of lexeme-glosse pairs.

```
rows = kuraapp.app.getObjects("lng_lex",
                              languagenr = 1)
for row in rows:
    print row.form, row.glosse
```

As you can see, once you’ve set up date application life suddenly gets *very* simple. The application object, which we access via the `kuraapp` module (because it’s a module level variable we are sure we use the same object everywhere), has a number of handy methods, like the one used here.

This method, `getObjects`takes one string parameter, the name of the table we want to retrieve data from (see the data model), and an unspecified number of named arguments.

These arguments are the fields you want to filter the result with; these can only be the fields that the table ’owns’, not the descriptions retrieved from related tables.

So, you *can* select on **languagenr = 1** but not on **language = "%den%"**. You *can* use the % wildcard.

The `getObjects` function returns a Python list containing the result of your query. Now you can use a simple for loop to loop over the records

These records have a rich set of functions, too, but at the most basic level, you can use use the dot notation to set and get values. (But there’s also `getFieldValue` and `setFieldValue` which you can use if you want to loop over all fields in a record.)

Now run this script as follows:

```
export KURADIR=/usr/local/share/kura
expor PYTHONPATH=/usr/local/share/kura
/usr/bin/python script1.py
```

And after a while you will get the following output (if you’ve done the tutorial, that is):

```
Initializing repository
Opening datastore:  /home/boudewijn/prj/kura-2.0/test.dbobj
Loading database: 0.0884840488434 seconds
sumir wine
esumir drink
```

```
sumiran drinker
sumirsumiran wine-bibber
tan RTV
yudir woman
ray good
esam say
ga TOP
boudewijn
```

16.2. Parents and children

Kura is based on a relation datamodel. That means (as I've explained in the introduction that things like lexemes do not stand on their own, but are part of a web of references to other things, like elements in a sentence. The next version, `script2.py`, will show you how to find display the sentences a lexeme occurs in:

The first part of the script is the same as with the previous example: I won't repeat it here.

Example 16-2. Example sentences with words from the dictionary (script2.py)

```
lexemes = kuraapp.app.getObjects("lng_lex",
                              languagenr = 1)
for lexeme in lexemes:
    elements = kuraapp.app.getObjects("lng_element",
                                      lexnr = lexeme.lexnr)
    if elements:
        print lexeme.form, lexeme.glosse
        print

        examples = {}
        for element in elements:
            if not examples.has_key(element.streamnr):
                stream = kuraapp.app.getObject("lng_stream",
                                               streamnr = element.streamnr
                examples[element.streamnr] = stream

        for streamnr, stream in examples.items():

            print "\t", stream.text
            print "\t", stream.translation()
            print
        print
        print
```

In the script above, we first loop over all lexemes in our lexicon. Then we retrieve the elements from the texts that have been associated with lexemes.

If we get any elements (that's not a certainty, of course), the form and the glosse fields of the lexeme are printed. Then we loop trough all elements, and for each element retrieve the stream the element belongs to.

These streams we store in a temporary Python datastructure named a *dictionary*, named 'examples'. The dictionary has the advantage that it is indexed by key, and that those keys are unique. This means that if our lexeme is associated with more than one element in a stream and thus ultimately more than once with the same stream, we will only print the stream once.

Otherwise we'd have a big chance of giving the same example sentence twice for the same lexeme. And that would look silly.

So, when the set of examples is established for this lexeme, we loop through the example streams, and print the text and the translation:

```
sumir wine

        Sumiran sumir esumire
        The drinker drank the wine.

        Esame, sumir ga ray
        He said, this wine, it’s good.



esumir drink

        Sumiran sumir esumire
        The drinker drank the wine.



esam say

        Esame, sumir ga ray
        He said, this wine, it’s good.



ga TOP

        Esame, sumir ga ray
        He said, this wine, it’s good.
```

Of course, we can also output a nicely interlinearized text (script3.py):

Example 16-3. Interlinear texts

```
for text in kuraapp.app.getObjects("lng_text"):
    print
    print text.title
    print
    print text.description
    print
    for stream in text.getStreams():
        print
        print stream.getInterlinearLines(stream.getElements()
        print stream.translation()
```

Again, this uses the first part of the first script demonstrated in this chapter. The output is as follows:

```
boudewijn@ejb:~/prj/kura-2.0> /usr/bin/python doc/script3.py
          Initializing repository
Opening datastore:  /home/boudewijn/prj/kura-2.0/test.dbobj
Loading database: 0.0685980319977 seconds

Sumir

A remark about wine.


Sumiran sumir esumire
drinker wine  drink
The drinker drank the wine.

Esame sumir ga  ray
say   wine  TOP good
He said, this wine, it’s good.
```

As you can see, it’s really easy to hack up short scripts that enable you to work with the data in your database. A more real-world example would be a more complex script that creates LaTeX output from your database, for instance a complete, marked-up dictionary.

16.3. Performing calculations on your database

You also might want to apply calculations or analysises on your database. This is not much different from the examples in the previous section, since it also entails retrieving data from the database, looping through it, and performing some action.

Let’s create a script that shows you the word order of the example sentences. This script depends on their being the right tags, so make sure you’ve completely filled the database.

Example 16-4. Calculating the word order

```
False = 0
True = 1

import os.path, sys
from kuralib import kuraapp
from kuragui.guiconfig import guiConf
from kuragui import guiconfig

if guiConf.backend == guiconfig.FILE:
    kuraapp.initApp(guiConf.backend,
                    dbfile = os.path.join(guiConf.filepath,
                                          guiConf.datastore))
elif guiConf.backend == guiconfig.SQL:
    if guiConf.username != "":
        try:
            kuraapp.initApp(guiConf.backend,
                            username = str(guiConf.username),
                            database = str(guiConf.database),
                            password = str(guiConf.password),
                            hostname = str(guiConf.hostname))

        except Exception, e:
            print "Error connecting to database: %s" % e
            sys.exit(1)

kuraapp.initCurrentEnvironment(guiConf.usernr,
                               guiConf.languagenr,
                               guiConf.projectnr)


orders = {}

for stream in kuraapp.app.getObjects("lng_stream"):
    order = []
    for element in stream.getElements():
        tag = element.getTag(tag = "POS")
        if tag.element_tagnr:
            order.append(tag.getDescription(False))
            continue
        elif element.lexnr:
            lexeme = kuraapp.app.getObject("lng_lex",
                                           lexnr = element.lexnr)
            tag = lexeme.getTag(tag = "POS")
            if tag.lex_tagnr:
                order.append(tag.getDescription(False))
                continue
        order.append("#")

    s = " ".join(order)
    if orders.has_key(s):
        orders[s] += 1
    else:
        orders[s] = 1

for k, v in orders.items():
    print k, v
```

This script is a little more the previous scripts. We are again building a dictionary, with the name ’orders’. We will use the actual word order as the key to this dictionary. The value is the number of times that order occurs.

We select all the streams in our database (a measly two, unless you’ve been adding texts since the tutorial). For each stream we determine the order of the words. In this case, we have decided that the tag POS (for Part Of Speech) carries the necessary information: noun, verb etcetera.

The parts of speech will be stored in order in a list: that list is called ’order’.

If the element itself has not been taggend, that is, when we use the function element.getTag(tag = "POS") and get back an emtpy tag record, we try again, with the lexical item associated with the element. If that doesn’t give us a POS tag, we give in and put a hash sign in the order.

When all elements for that stream have been analyzed, the list of elements (’order’) is complete. We now convert the list to a string, because you cannot use lists as keys to a dictionary in Python. It’s a pity, but there it is -- if you want to know more, consult the Python tutorial. The conversion is done by joining the parts with a space: **" ".join(order)** is the command.

Then, when we’ve got our word order token in the string ’s’, we determine whether we have encountered it before. That is, when it already occurs in the dictionary, we simply bump up the count by one. If it’s a new token, we create an entry and set the count to one.

Finally, the contents of the dictionary are printed:

```
boud@calcifer:~/prj/kura-2.0> python doc/script4.py
Initializing repository
```

```
Opening datastore:  /home/boud/prj/kura-2.0/test.dbobj
Loading database: 0.0657420158386 seconds
N N V 1
V N PART STATV 1
```

It’s not much at the moment, but imagine! With thousands of texts all neatly marked up... Get typing already.

16.4. Adding and modifying data

The scripts in this section are intrinsically more dangerous than the scripts in the previous section. After all, if you add a whole load of garbage to your carefully filled database, you will probably not be able to undo the changes.

So, it’s a good idea to make a backup of your data, possbile as part of your script. If you’re on Unix, you could probably use code like this to create a backup:

Example 16-5. Backing up with a script

```
import os.path
from kuragui.guiconfig import guiConf
from kuragui import guiconfig

if guiConf.backend == guiconfig.FILE:
    f = os.path.join(guiConf.filepath, guiConf.datastore)
    os.system("cp %s %s.bak" % (f, f))
elif guiConf.backend == guiconfig.SQL:
    os.system("mysqldump %s -p=%s -u=%s -h=%s > backup.sql"
              % (str(guiConf.database),
                 str(guiConf.password),
                 str(guiConf.username),
                 str(guiConf.hostname))
```

So, now that you know how to backup, you can safely mangle your data. Let’s create a little script that adds a load of lexical items to your database. You will probably already have set of files that contain lexical data. You first task is to massage those files into a semblance of parseable unity.

For the sake of this example, I’ve assumed a simple comma separated format, such as KSpread, Gnumeric or Windows apps like Access all know how to produce.

Parsing csv files is harder than you’d think, and this is an excellent opportunity to learn about the wonderful Vaults of Parnassus(http://www.vex.net/parnassus/), where you can find lots of extensions for Python, most of them free.

The query http://py.vaults.ca/apyllo.py?find=csv(http://py.vaults.ca/apyllo.py?find=csv) gives us a plethora of csv parsing packages, but let's take a simple one, one that I can package in this tutorial: splitcvs from Colotstudy(http://www.colorstudy.com/software/webware/).

First, we're going to write code that reads the file with lexemes (`lexicon.csv`) and splits it. We print the result of the splitting, so we can be sure our code is correct. Please consult the complete script (`script5.py`) for the splitCSVLine function and the way the initalisation of the kuraapp.app object has moved to a `init()` function.

Example 16-6. Importing lexical data (script5.py)

```
...

def main(args):
    if len(args) < 2:
        print "Usage: python script5.py f1...fn"
        sys.exit(1)
    init()
    for line in codecs.open(sys.argv[1], "r", "UTF-8"):
        print splitCSVLine(line)

if __name__ == "__main__":
    main(sys.argv)
```

As you can see, we've moved the body of the script code into a function, called `main(args)`. This is useful when your scripts start to get bigger. When that's the case, you will likely have created several useful functions you might like to use from other scripts.

Now if you import a Python script file, python executes every statement from the first line to the last. That's not what you want when you import a script: in that case you merely want access to the functions. So, if you move your script code into a `main(args)` function, only the `def` statements will be executed: that means that Python defines the functions, but doesn't execute them yet. When the Python interpreter arrives at the

```
if __name__ ==
            "__main__":
```

line, it will see whether the name of the function is executing is `__main__`, and if that's the case, it will execute the `main()` function. And, if you run a script directly from the commandline, it *will* define a `__name__` to be "__main__", but not when you import the a file.

The `sys` module provides a handy variable, `argv` that contains the command-line arguments to the script. If you start the script with: **`python script5.py lexicons.cvs`**, the arguments will be:

```
['doc/script5.py',
            'doc/lexicon.csv']
```

, which is easily demonstrated by adding a

```
print sys.argvc
```

line in the `main(args)` function.

The `main(args)` function first checks whether it has enough arguments, and if there aren't enough, it complains and exits with an error. Otherwise, it will take the second argument (with number 1 -- programming is base zero) to refer to a filename.

Python has extensive support for Unicode (otherwise I wouldn't have tried to write Kura in it). You need to use the codecs module to open a file in another encoding than your platform encoding (which is most likely ASCII). The first argument to `codecs.open` is the filename, the second the mode ("r" means read-only) and the third the encoding. This can be anything from utf-8 to SJIS. But the strings Python creates when reading lines from the file will be Unicode objects.

So you can see that we open the file, read its lines and process each line with the `splitCSVLine()`. This function returns a list for each line:

```
oud@calcifer:~/prj/kura-2.0> python doc/script5.py doc/lexicon.csv
Initializing repository
Opening datastore:  /home/boud/prj/kura-2.0/test.dbobj
Loading database: 0.0796259641647 seconds
['form', 'glosse', 'POS', 'phonetic form', ”]
['hedrad', 'to force', 'V', 'he.drad', ”]
['jedor', 'winehouse', 'N', 'je.dor', ”]
['kinad', 'miracle', 'V', 'ki.nad', ”]
```

As you can see, the conversion is not perfect. This function adds a spurious empty element at the end of each line. We can ignore that, but it shows how import it is to check the format of the data your script generates at every step. (If you want to, you can also had the `splitCSVLine(line)` to behave itself better.

And now for the real fun. We're going to create lexemes and insert them in the database, and then saving the database.

```
def main(args):
    if len(args) < 2:
        print "Usage: python script5.py f1...fn"
        sys.exit(1)
    init()
    for line in codecs.open(args[1], "r", "UTF-8"):
        line = splitCSVLine(line)
        print "Inserting %s" % line[0]
        lexeme = kuraapp.app.createObject("lng_lex", fields={},
                                          form = line[0],
                                          glosse = line[1],
                                          languagenr = guiConf.languagenr,
                                          phonetic_form = line[3],
                                          usernr = guiConf.usernr)
        lexeme.insert()
        tag = kuraapp.app.createObject("lng_lex_tag", fields={},
                                       lexnr = lexeme.lexnr,
                                       tag = "POS",
                                       value = line[2],
                                       usernr = guiConf.usernr)
        tag.insert()
        tag = kuraapp.app.createObject("lng_lex_tag",
                                       lexnr = lexeme.lexnr,
                                       tag = "FILE",
                                       value = args[1],
                                       usernr = guiConf.usernr)
        tag.insert()
    kuraapp.app.saveFile()

if __name__ == "__main__":
    main(sys.argv)
```

If you haven’t added a tag FILE before, you will get the following exception when you run the script:

```
boud@calcifer:~/prj/kura-2.0> python doc/script5.py doc/lexicon.csv
Initializing repository
Opening datastore:  /home/boud/prj/kura-2.0/test.dbobj
Loading database: 0.075140953064 seconds
Traceback (most recent call last):
  File "doc/script5.py", line 99, in ?
    main(sys.argv)
  File "doc/script5.py", line 97, in main
    tag.insert()
  File "/home/boud/prj/kura-2.0/dbobj/dbobj.py", line 167, in insert
    if  self.__verify() :
  File "/home/boud/prj/kura-2.0/dbobj/dbobj.py", line 155, in __verify
    self.__checkParents()
  File "/home/boud/prj/kura-2.0/dbobj/dbobj.py", line 123, in __checkParer
    if self.app.getObjects(relation.rtable, fields={key.foreign:keyval})==
  File "/home/boud/prj/kura-2.0/dbobj/appobj.py", line 305, in getObjects
    tbl.select(rec, orderBy = orderBy)
  File "/home/boud/prj/kura-2.0/dbobj/dbobj.py", line 426, in select
    orderBy = orderBy)
  File "/home/boud/prj/kura-2.0/dbobj/textdb/textquery.py", line 57, in se
    resultSet = self.database[table].select(queryRec)
  File "/home/boud/prj/kura-2.0/dbobj/textdb/table.py", line 245, in selec
    raise "No record in table %s with primary key %s " % (self.__name, str
No record in table lng_tag with primary key FILE
```

On the other hand, if you have done so, then the script will first create a new lexeme, and insert it. Upon insertion, the field `lexnr` is calculated, and then you use it to create the relation between lexeme and tag.

Note that you need to *save* the datastore explicitly with `kuraapp.app.saveFile()`. The `createObject()` function takes any number of parameters. The first is the name of the table you want to create an object of. The second parameter a named one: `fields` and needs a dictionary with field-value pairs. Or an empty dictionary if you go for adding fieldsnames as parameters. That works, too.

And when you run the script, you will see the following output:

```
boud@calcifer:~/prj/kura-2.0> python doc/script5.py doc/lexicon.csv
Initializing repository
Opening datastore:  /home/boud/prj/kura-2.0/test.dbobj
Loading database: 0.0587170124054 seconds
Inserting hedrad
Inserting jedor
Inserting kinad
Saving database: 0.0702489614487 seconds
```

Armed with this knowledge, you can create scripts as complicated as you want. Consult the section on the datamodel on which tables are available and fields they sport. Consult the section on the objectmodel to learn the API of Kura’s library objects, and get coding!

Chapter 17. Hacking Kura

This chapter gives you some pointers in case you want to change Kura. I hope that if you do code some useful additions to Kura you will share them with me!

17.1. Retrieving the code

The distribution of Kura does not contain everything. For instance, the source for this manual isn't part of the distribution. If you really want to start hacking Kura, you need to get a CVS checkout. CVS is part of every Linux distribution. The commandline client is easiest in daily use, but you might want to try the Cervisia KDE CVS gui(http://cervisia.sf.net).

Execute the following command to get the latest Kura sources:

```
cvs -d :pserver:anonymous@rempt.xs4all.nl:/data/pub/cvsroot co kura-
2.0
```

From now on, you can simple enter:

```
cvs up
```

To sync your sources with mine. If you give:

```
cvs diff -u > ~/kura-diffs
```

Then you will get a file called kura-diffs in your home directory that contains all your changes in unified diff format. Ready for you to send me. Never send me patches that are not in unified diff format!

17.2. Layout of the code

The Kura sourcetree has the following parts:

- dbobj: the generic data objects code.
- dbobj/textdb: the file-based backend
- datamodel: sql scripts to create and fill MySQL
- dialogs: Qt Designer sources for the special dialogs, wizards and widgets

- doc: the manual documentation
- kuragui: specialized gui elements, such as data-aware listviews and dialog boxes.
- kuraclient: the code for the gui client.
- kuralib: specialiazed data objects
- pixmaps: toolbar buttons
- utils: useful scripts

In the root directory there are scripts to create the distributions, a makefile that is useful to clean up and create documentation or dialogs and the `k` script which is used to start the developmet version of Kura.

17.2.1. Adding a field to a table

If you want to add a field to a table, first consider whether you need the field, whether you cannot use the tags tables. If you still want to add a field, first write an `upgrade.sql` script that adds the field to the database.

Then update the repository information in `kuralib/lngapp.py`. Take a good look at the way other fields are defined. In particular, look at the convenience functions at the beginning of `lngapp.py`.

In some cases, you are done now. Most tables use a generic dialog that is dynamically created based on the contents of the repostory.

For other tables, there is a specialized dialog. Check whether that is true in `dialogs`. If you see one, update it with Qt Designer. Then generate the forms with **make dialogs**. Now the autogenerated dialog will contain your new field. Then update the corresponding proxy dialog from `kuraclient`: it will have the same name as the generated dialog, but starts with `dlg` instead of `frm`. Update that, too.

If there isn't a Designer form, there might still be a specialized dialog in `kuraclient`. Check that and update it. Now test your new field.

17.2.2. Adding a preference option

You need to hack `kuragui/guiconfig.py` to add the option, with defaults and perhaps code to read and write the option.

Your next call is to edit the `frmpreferences.ui` with Qt Designer and then to generate the form with **make dialogs**. The corresponding dialog, `kuraclient/dlgpreferences.py` must be update, too. Take a good look at how options are read and saved.

What now remains is to use your preference in the right places... For instance, if you’ve created a new preference to change the color of the cursor in the interlinear text editor, you would have to hack `kuraclient/kurailcanvas.py`.

17.2.3. Adding a table

To add a table, you need to create an SQL script that creates the table, and add the table to `lngapp.py`. In `kuraclient/kurawindow.py` you will find code to add a menu option. Carefully look at the existing code, and add your own.

You should be done now, for simple tables. More complex tables might need a more complex dialog window. Take `dlgtagtypecode.py` as an example, or perhaps `dlglexeme.py`.

17.2.4. LaTeX output

Take a good, hard look at the grammar project. Look at the `export()` function in, for instance, `kuralib/lng_text.py`. Look at `kuralib/docbook.py` for the supporting functions. Now duplicate that for LaTeX. Of fix the docbook2latex converters that do exist.

17.2.5. Adding a parser for texts

This is an interesting hack. First, you need to locate the place where I parse the texts. This is, against any sane design decision, not in `kuralib/lng_elmt.py` or `kuralib/lng_text.py` but in `kuraclient/dlgnewtext.py`.

Cut the parser from that module, and make it use a parser that using a config option (perhaps an extra field in lng_language) can be dynamically loaded. Check the current API of the parser, and change it, if necessary in the code that calls the parser.

Write the parser. Submit the diff!

I. The datamodel

Please consult the chart of the Kura datamodel(http://www.valdyas.org/linguistics/data000.jpg) for an overview of how the Kura's tables link. Most of the relations in that image still quite correct, except for the preferences table, which has disappeared.

lng_affiliationcode

Name

`lng_affiliationcode`—

Attributes

Table type	Code table with a textual primary key
Table alias	affc
Primary key field	affiliationcode
Sequence field	None
Hint	
Table described by	description
Fieldorder	affiliationcode, description, institution, url
Indexes	
Unique Indexes	

Fields

Fieldname	Field definition
url	String, 255 (null allowed)
description	None
affiliationcode	None
institution	String, 255 (null allowed)

Lookup tables (parents)

Childtable	Local key	Foreign key
lng_user	affiliationcode	affiliationcode

lng_categorycode

Name

`lng_categorycode`—

Attributes

Table type	numeric primary key
Table alias	catc
Primary key field	categorycode
Sequence field	None
Hint	
Table described by	description
Fieldorder	categorycode, description
Indexes	
Unique Indexes	

Fields

Fieldname	Field definition
categorycode	None
description	None

Lookup tables (parents)

Childtable	Local key	Foreign key
lng_reference	categorycode	main_categorycode
lng_reference	categorycode	sub_categorycode

lng_doc_doc

Name

`lng_doc_doc`—

Attributes

Table type	numeric primary key
Table alias	dcdc
Primary key field	docdocnr
Sequence field	None
Hint	
Table described by	description
Fieldorder	docdocnr, documentnr_1, title_1, documentnr_2, title_2, linkcode, link, description
Indexes	
Unique Indexes	

Fields

Fieldname	**Field definition**
docdocnr	Integer, 10 (pk)
documentnr_2	None
documentnr_1	None
link	String, 255 (null allowed)
description	String, 255 (null allowed)
title_1	String, 255 (null allowed)
title_2	String, 255 (null allowed)
linkcode	None

Lookup tables (parents)

Name	**Keypair**	**Descriptors**	**Related table**	**Related alias**

Name	Keypair	Descriptors	Related table	Related alias
dcdc_doc1	documentnr_1, documentnr	title_1, title	lng_document	doc1
dcdc_doc2	documentnr_2, documentnr	title_2, title	lng_document	doc2
dcdc_lnkc	linkcode, linkcode	link, description	lng_linkcode	lnkc

lng_document

Name

`lng_document—`

Attributes

Table type	numeric primary key
Table alias	doc
Primary key field	documentnr
Sequence field	None
Hint	
Table described by	description
Fieldorder	documentnr, title, description, url, creation_date, usernr, user
Indexes	
Unique Indexes	

Fields

Fieldname	Field definition
usernr	Integer, 10 (null allowed)
description	Text, 255 (null allowed)
title	None

Fieldname	Field definition
url	String, 255 (null allowed)
user	String, 255 (null allowed)
documentnr	Integer, 10 (pk)
creation_date	String, 255 (null allowed)

Lookup tables (parents)

Childtable	Local key	Foreign key
lng_doc_doc	documentnr	documentnr_1
lng_doc_doc	documentnr	documentnr_1

Lookup tables (parents)

Name	Keypair	Descriptors	Related table	Related alias
doc_user	usernr, usernr	user, name	lng_user	user

lng_element

Name

`lng_element—`

Attributes

Table type	Numeric primary key and a sequential counter
Table alias	elmt
Primary key field	elementnr
Sequence field	streamnr
Hint	
Table described by	text

Fieldorder	elementnr, text, elementtypecode, elementtype, streamnr, stream, seqnr, languagenr, language, parent_elementnr, parent_element, lexnr, lexeme, usernr, user, datestamp
Indexes	streamnr, languagenr, seqnr, parent_elementnr, lexnr
Unique Indexes	

Fields

Fieldname	**Field definition**
parent_elementnr	Integer, 10 (null allowed)
usernr	Integer, 10 (null allowed)
stream	String, 255 (null allowed)
language	String, 255 (null allowed)
text	String, 255 (null allowed)
lexnr	Integer, 10 (null allowed)
parent_element	String, 255 (null allowed)
streamnr	None
lexeme	String, 255 (null allowed)
datestamp	Date/time, 255 (null allowed)
seqnr	Integer, 10 (seq)
user	String, 255 (null allowed)
elementnr	Integer, 10 (pk)
elementtypecode	None
languagenr	None
elementtype	String, 255 (null allowed)

Lookup tables (parents)

Childtable	**Local key**	**Foreign key**
lng_element_tag	elementnr	elementnr
lng_element	elementnr	parent_elementnr

Lookup tables (parents)

Name	Keypair	Descriptors	Related table	Related alias
elmt_lex	lexnr, lexnr	lexeme, glosse	lng_lex	lex
elmt_strm	streamnr, streamnr	stream, text	lng_stream	strm
elmt_elmt	parent_elementnr, elementnr	parent_element, text	lng_element	elmt2
elmt_user	usernr, usernr	user, name	lng_user	user
elmt_lngg	languagenr, languagenr	language, language	lng_language	lngg
elmt_eltc	elementtypecode, elementtypecode	elementtype, description	lng_elementtypecode	eltc

lng_element_tag

Name

`lng_element_tag`—

Attributes

Table type	numeric primary key
Table alias	eltg
Primary key field	element_tagnr
Sequence field	None
Hint	
Table described by	tagname
Fieldorder	element_tagnr, elementnr, tag, element, tagname, value, description, note, usernr, user, datestamp
Indexes	elementnr, tag
Unique Indexes	

Fields

Fieldname	Field definition
usernr	Integer, 10 (null allowed)
description	String, 200 (null allowed)
value	String, 255 (null allowed)
element	String, 255 (null allowed)
note	Text, 255 (null allowed)
element_tagnr	Integer, 10 (pk)
tag	None
user	String, 255 (null allowed)
tagname	String, 255 (null allowed)
datestamp	Date/time, 255 (null allowed)
elementnr	None

Lookup tables (parents)

Name	Keypair	Descriptors	Related table	Related alias
eltg_elmt	elementnr, elementnr	element, text	lng_element	elmt
eltg_tag	tag, tag	tagname, name	lng_tag	tag
eltg_user	usernr, usernr	user, name	lng_user	user

lng_elementtypecode

Name

`lng_elementtypecode`—

Attributes

Table type	Code table with a textual primary key
Table alias	eltc

Primary key field	elementtypecode
Sequence field	None
Hint	
Table described by	description
Fieldorder	elementtypecode, description, parent_elementtypecode, parent_elementtype, splitmarker, joinmarker
Indexes	
Unique Indexes	

Fields

Fieldname	Field definition
joinmarker	String, 255 (null allowed)
description	None
elementtypecode	None
parent_elementtypecode	String, 255 (null allowed)
parent_elementtype	String, 255 (null allowed)
splitmarker	String, 255 (null allowed)

Lookup tables (parents)

Childtable	Local key	Foreign key
lng_elementtypecode	elementtypecode	parent_elementtypecode
lng_element	elementtypecode	elementtypecode

Lookup tables (parents)

Name	Keypair	Descriptors	Related table	Related alias
eltc_eltc	parent_elementtypecode, elementtype-code	parent_elementtype, description	lng_elementtypecode	eltc2

lng_language

Name

`lng_language`—

Attributes

Table type	Recursive table with a numeric primary key
Table alias	lngg
Primary key field	languagenr
Sequence field	None
Hint	
Table described by	language
Fieldorder	languagenr, language, description, parent_languagenr, parent_language, documentroot, title
Indexes	parent_languagenr
Unique Indexes	

Fields

Fieldname	Field definition
parent_languagenr	Integer, 10 (null allowed)
description	Text, 255 (null allowed)
language	None
documentroot	Integer, 10 (null allowed)
languagenr	Integer, 10 (pk)
title	String, 255 (null allowed)
parent_language	String, 255 (null allowed)

Lookup tables (parents)

Childtable	Local key	Foreign key
lng_lex	languagenr	languagenr
lng_recording	languagenr	languagenr
lng_element	languagenr	languagenr
lng_stream	languagenr	languagenr
lng_text	languagenr	languagenr
lng_proj_lngg	languagenr	languagenr
lng_scan	languagenr	languagenr
lng_language	languagenr	parent_languagenr

Lookup tables (parents)

Name	Keypair	Descriptors	Related table	Related alias
lngg_lngg	parent_languagenr, languagenr	parent_language, language	lng_language	lngg2
lngg_doc	documentroot, documentnr	title, title	lng_document	doc

lng_lex

Name

`lng_lex`—

Attributes

Table type	numeric primary key
Table alias	lex
Primary key field	lexnr
Sequence field	None
Hint	

Table described by	form
Fieldorder	lexnr, form, glosse, description, phonetic_form, alternative_form, languagenr, language, isdone, usernr, user, datestamp
Indexes	languagenr, glosse, form
Unique Indexes	

Fields

Fieldname	**Field definition**
usernr	Integer, 10 (null allowed)
description	Text, 255 (null allowed)
language	String, 255 (null allowed)
phonetic_form	String, 100 (null allowed)
glosse	None
lexnr	Integer, 10 (pk)
languagenr	None
user	String, 255 (null allowed)
form	None
datestamp	Date/time, 255 (null allowed)
alternative_form	String, 50 (null allowed)
isdone	Boolean, 255 (null allowed)

Lookup tables (parents)

Childtable	**Local key**	**Foreign key**
lng_lex_tag	lexnr	lexnr
lng_lex_lex	lexnr	lexnr_1
lng_element	lexnr	lexnr

Lookup tables (parents)

Name	**Keypair**	**Descriptors**	**Related table**	**Related alias**

Name	**Keypair**	**Descriptors**	**Related table**	**Related alias**
lex_lngg	languagenr, languagenr	language, language	lng_language	lngg
lex_user	usernr, usernr	user, name	lng_user	user

lng_lex_lex

Name

`lng_lex_lex`—

Attributes

Table type	numeric primary key
Table alias	lxlx
Primary key field	lxlxnr
Sequence field	None
Hint	
Table described by	form_1, form_2, relation
Fieldorder	lxlxnr, lexnr_1, form_1, lexnr_2, form_2, lxlxrelcode, relation, usernr, note, user, datestamp
Indexes	lexnr_1, lexnr_2
Unique Indexes	

Fields

Fieldname	**Field definition**
lexnr_1	None
lxlxrelcode	None
lexnr_2	None
lxlxnr	Integer, 10 (pk)
note	Text, 255 (null allowed)

Fieldname	Field definition
usernr	Integer, 10 (null allowed)
relation	String, 255 (null allowed)
user	String, 255 (null allowed)
datestamp	Date/time, 255 (null allowed)
form_1	String, 255 (null allowed)
form_2	String, 255 (null allowed)

Lookup tables (parents)

Name	Keypair	Descriptors	Related table	Related alias
lxlx_user	usernr, usernr	user, name	lng_user	user
lxlx_lex1	lexnr_1, lexnr	form_1, form	lng_lex	lex1
lxlx_lex2	lexnr_2, lexnr	form_2, form	lng_lex	lex2
lxlx_lxrl	lxlxrelcode, lxlxrelcode	relation, description	lng_lxlxrelcode	lxrl

lng_lex_tag

Name

`lng_lex_tag`—

Attributes

Table type	numeric primary key
Table alias	lxtg
Primary key field	lex_tagnr
Sequence field	None
Hint	
Table described by	tagname

Fieldorder	lex_tagnr, lexnr, tag, lexeme, tagname, value, description, note, usernr, datestamp
Indexes	lexnr, tag
Unique Indexes	

Fields

Fieldname	**Field definition**
note	Text, 255 (null allowed)
usernr	Integer, 10 (null allowed)
description	String, 200 (null allowed)
lexnr	None
value	String, 255 (null allowed)
lexeme	String, 255 (null allowed)
tag	None
user	String, 255 (null allowed)
tagname	String, 255 (null allowed)
datestamp	Date/time, 255 (null allowed)
lex_tagnr	Integer, 10 (pk)

Lookup tables (parents)

Name	**Keypair**	**Descriptors**	**Related table**	**Related alias**
lxtg_lex	lexnr, lexnr	lexeme, form	lng_lex	lex
lxtg_tag	tag, tag	tagname, name	lng_tag	tag
lxtg_user	usernr, usernr	user, name	lng_user	user

lng_linkcode

Name

`lng_linkcode`—

Attributes

Table type	Code table with a textual primary key
Table alias	lnkc
Primary key field	linkcode
Sequence field	None
Hint	
Table described by	description
Fieldorder	linkcode, description
Indexes	
Unique Indexes	

Fields

Fieldname	**Field definition**
description	None
linkcode	None

Lookup tables (parents)

Childtable	**Local key**	**Foreign key**
lng_doc_doc	linkcode	linkcode

lng_lxlxrelcode

Name

`lng_lxlxrelcode—`

Attributes

Table type	Code table with a textual primary key
Table alias	lxrl
Primary key field	lxlxrelcode
Sequence field	None
Hint	
Table described by	description
Fieldorder	lxlxrelcode, description
Indexes	
Unique Indexes	

Fields

Fieldname	Field definition
lxlxrelcode	None
description	String, 255 (null allowed)

Lookup tables (parents)

Childtable	Local key	Foreign key
lng_lex_lex	lxlxreclode	lxlxrelcode

lng_proj_lngg

Name

`lng_proj_lngg`—

Attributes

Table type	numeric primary key
Table alias	prln
Primary key field	prlnnr
Sequence field	None
Hint	
Table described by	project, language
Fieldorder	project, projectnr, languagenr, language, prlnnr
Indexes	languagenr
Unique Indexes	

Fields

Fieldname	**Field definition**
project	String, 255 (null allowed)
projectnr	None
languagenr	None
language	String, 255 (null allowed)
prlnnr	Integer, 10 (pk)

Lookup tables (parents)

Name	**Keypair**	**Descriptors**	**Related table**	**Related alias**
prln_project	projectnr, projectnr	project, description	lng_project	proj
prln_lngg	languagenr, languagenr	language, language	lng_language	lngg

lng_proj_text

Name

`lng_proj_text—`

Attributes

Table type	numeric primary key
Table alias	prtx
Primary key field	prtxnr
Sequence field	None
Hint	
Table described by	project, text
Fieldorder	prtxnr, projectnr, project, textnr, text
Indexes	projectnr, textnr
Unique Indexes	

Fields

Fieldname	**Field definition**
prtxnr	Integer, 10 (pk)
project	String, 255 (null allowed)
projectnr	None
text	String, 255 (null allowed)
textnr	None

Lookup tables (parents)

Name	**Keypair**	**Descriptors**	**Related table**	**Related alias**
prtx_proj	projectnr, projectnr	project, description	lng_project	proj

Name	Keypair	Descriptors	Related table	Related alias
prtx_text	textnr, textnr	text, title	lng_text	text

lng_proj_user

Name

`lng_proj_user`—

Attributes

Table type	numeric primary key
Table alias	prus
Primary key field	prusnr
Sequence field	None
Hint	
Table described by	project, user
Fieldorder	prusnr, projectnr, project, usernr, user
Indexes	
Unique Indexes	

Fields

Fieldname	Field definition
project	String, 255 (null allowed)
projectnr	None
usernr	None
prusnr	Integer, 10 (pk)
user	String, 255 (null allowed)

Lookup tables (parents)

Name	Keypair	Descriptors	Related table	Related alias
prus_proj	projectnr, projectnr	project, description	lng_project	proj
prus_user	usernr, usernr	user, name	lng_user	user

lng_project

Name

`lng_project—`

Attributes

Table type	numeric primary key
Table alias	proj
Primary key field	projectnr
Sequence field	None
Hint	Projects A project is a basic unit of research in Kura.
Table described by	description
Fieldorder	projectnr, description, summary, url, documentroot, title, grants
Indexes	
Unique Indexes	

Fields

Fieldname	Field definition
grants	String, 255 (null allowed)
description	None
title	String, 255 (null allowed)

Fieldname	Field definition
url	String, 255 (null allowed)
projectnr	Integer, 10 (pk)
documentroot	Integer, 10 (null allowed)
summary	Text, 255 (null allowed)

Lookup tables (parents)

Childtable	Local key	Foreign key
lng_recording	projectnr	projectnr
lng_proj_lngg	projectnr	projectnr
lng_proj_text	projectnr	projectnr
lng_proj_user	projectnr	projectnr
lng_scan	projectnr	projectnr

Lookup tables (parents)

Name	Keypair	Descriptors	Related table	Related alias
proj_doc	documentroot, documentnr	title, title	lng_document	doc

lng_recording

Name

`lng_recording`—

Attributes

Table type	numeric primary key
Table alias	recd
Primary key field	recordingnr

Sequence field	None
Hint	
Table described by	description
Fieldorder	recordingnr, title, url, source, tapenr, tape_location, informant, duration, recording_date, languagenr, language, projectnr, project, description, usernr, user
Indexes	languagenr, projectnr
Unique Indexes	

Fields

Fieldname	**Field definition**
language	String, 255 (null allowed)
usernr	Integer, 10 (null allowed)
description	None
recording_date	Date/time, 255 (null allowed)
title	None
url	None
tapenr	String, 255 (null allowed)
project	String, 255 (null allowed)
source	String, 255 (null allowed)
languagenr	None
user	String, 255 (null allowed)
recordingnr	Integer, 10 (pk)
tape_location	String, 255 (null allowed)
duration	String, 255 (null allowed)
projectnr	Integer, 10 (null allowed)
informant	String, 255 (null allowed)

Lookup tables (parents)

Childtable	**Local key**	**Foreign key**
lng_text	recordingnr	recordingnr

Lookup tables (parents)

Name	Keypair	Descriptors	Related table	Related alias
recd_user	usernr, usernr	user, name	lng_user	user
recd_lngg	languagenr, languagenr	language, language	lng_language	lngg
recd_proj	projectnr, projectnr	project, description	lng_project	proj

lng_reference

Name

`lng_reference`—

Attributes

Table type	numeric primary key
Table alias	refs
Primary key field	referencenr
Sequence field	None
Hint	
Table described by	title, author
Fieldorder	referencenr, author, year, abbrev, title, periodical, place, publisher, catalogue_card, series, volume, pages, note, main_categorycode, main_category, sub_categorycode, sub_category
Indexes	author, abbrev, title
Unique Indexes	

Fields

Fieldname	**Field definition**
volume	String, 255 (null allowed)
note	Text, 255 (null allowed)
sub_category	String, 255 (null allowed)
sub_categorycode	String, 255 (null allowed)
author	None
main_categorycode	String, 255 (null allowed)
series	String, 255 (null allowed)
title	None
publisher	String, 255 (null allowed)
referencenr	Integer, 10 (pk)
main_category	String, 255 (null allowed)
abbrev	String, 25 (null allowed)
place	String, 255 (null allowed)
periodical	String, 255 (null allowed)
year	None
pages	String, 255 (null allowed)
catalogue_card	String, 255 (null allowed)

Lookup tables (parents)

Name	**Keypair**	**Descriptors**	**Related table**	**Related alias**
refs_catc1	main_categorycode, categorycode	main_category, description	lng_categorycode	catc1
refs_catc2	sub_categorycode, categorycode	sub_category, description	lng_categorycode	catc2

lng_scan

Name

`lng_scan`—

Attributes

Table type	numeric primary key
Table alias	scan
Primary key field	scannr
Sequence field	None
Hint	
Table described by	title, scan_date
Fieldorder	scannr, title, url, manuscript_location, page, scan_date, size, description, languagenr, language, projectnr, project, usernr, user
Indexes	languagenr, projectnr
Unique Indexes	

Fields

Fieldname	**Field definition**
scannr	Integer, 10 (pk)
usernr	Integer, 10 (null allowed)
description	None
language	String, 255 (null allowed)
title	None
url	None
scan_date	Date/time, 255 (null allowed)
manuscript_location	String, 255 (null allowed)
project	String, 255 (null allowed)
languagenr	None
user	String, 255 (null allowed)
projectnr	None
page	String, 255 (null allowed)
size	String, 255 (null allowed)

Lookup tables (parents)

Childtable	**Local key**	**Foreign key**
lng_text	scannr	scannr

Lookup tables (parents)

Name	Keypair	Descriptors	Related table	Related alias
scan_user	usernr, usernr	user, name	lng_user	user
scan_lngg	languagenr, languagenr	language, language	lng_language	lngg
scan_proj	projectnr, projectnr	project, description	lng_project	proj

lng_stream

Name

`lng_stream`—

Attributes

Table type	Numeric primary key and a sequential counter
Table alias	strm
Primary key field	streamnr
Sequence field	textnr
Hint	
Table described by	text
Fieldorder	streamnr, textnr, title, text, seqnr, languagenr, language, usernr, user, datestamp
Indexes	textnr, languagenr, seqnr
Unique Indexes	

Fields

Fieldname	Field definition

Fieldname	Field definition
streamnr	Integer, 10 (pk)
usernr	Integer, 10 (null allowed)
languagenr	None
user	String, 255 (null allowed)
language	String, 255 (null allowed)
title	String, 255 (null allowed)
datestamp	Date/time, 255 (null allowed)
text	String, 255 (null allowed)
seqnr	Integer, 10 (seq)
textnr	None

Lookup tables (parents)

Childtable	Local key	Foreign key
lng_stream_tag	streamnr	streamnr
lng_element	streamnr	streamnr

Lookup tables (parents)

Name	Keypair	Descriptors	Related table	Related alias
strm_text	textnr, textnr	title, title	lng_text	text
strm_user	usernr, usernr	user, name	lng_user	user
strm_lngg	languagenr, languagenr	language, language	lng_language	lngg

lng_stream_tag

Name

`lng_stream_tag—`

Attributes

Table type	numeric primary key
Table alias	sttg
Primary key field	stream_tagnr
Sequence field	None
Hint	
Table described by	tagname
Fieldorder	stream_tagnr, streamnr, tag, stream, tagname, value, description, note, usernr, user, datestamp
Indexes	streamnr, tag
Unique Indexes	

Fields

Fieldname	**Field definition**
stream_tagnr	Integer, 10 (pk)
note	Text, 255 (null allowed)
usernr	Integer, 10 (null allowed)
stream	String, 255 (null allowed)
value	String, 255 (null allowed)
streamnr	None
tag	None
user	String, 255 (null allowed)
tagname	String, 255 (null allowed)
datestamp	Date/time, 255 (null allowed)
description	String, 200 (null allowed)

Lookup tables (parents)

Name	**Keypair**	**Descriptors**	**Related table**	**Related alias**
sttg_strrm	streamnr, streamnr	stream, text	lng_stream	strm
sttg_tag	tag, tag	tagname, name	lng_tag	tag
sttg_user	usernr, usernr	user, name	lng_user	user

lng_tag

Name

`lng_tag`—

Attributes

Table type	Code table with a textual primary key
Table alias	tag
Primary key field	tag
Sequence field	None
Hint	
Table described by	name
Fieldorder	tag, name, description, tagtypecode, tagtype, text, stream, element, lexeme
Indexes	
Unique Indexes	

Fields

Fieldname	**Field definition**
lexeme	Boolean, 255 (null allowed)
tag	None
tagtypecode	None
name	None
stream	Boolean, 255 (null allowed)
text	Boolean, 255 (null allowed)
description	String, 255 (null allowed)
element	Boolean, 255 (null allowed)
tagtype	String, 255 (null allowed)

Lookup tables (parents)

Childtable	Local key	Foreign key
lng_text_tag	tag	tag
lng_lex_tag	tag	tag
lng_element_tag	tag	tag
lng_stream_tag	tag	tag

Lookup tables (parents)

Name	Keypair	Descriptors	Related table	Related alias
tag_ttpc	tagtypecode, tagtypecode	tagtype, description	lng_tagtypecode	ttpc

lng_tagdomain

Name

`lng_tagdomain`—

Attributes

Table type	numeric primary key
Table alias	tdmn
Primary key field	domainnr
Sequence field	None

Hint	Tag domains The items you define here form the set of choices you are presented with when you want to add a tag with some value to an item. For instance, if you want to label a word in a phrase with a part of speech, you can define here all possible parts of speech. When labelling the word, you will then be able to choose from the possibilities defined here.
Table described by	description
Fieldorder	domainnr, tag, tagname, abbreviation, description
Indexes	
Unique Indexes	

Fields

Fieldname	Field definition
abbreviation	None
tag	None
domainnr	Integer, 10 (pk)
description	None
tagname	String, 255 (null allowed)

Lookup tables (parents)

Childtable	Local key	Foreign key
lng_text_tag	domainnr	domainnr
lng_lex_tag	domainnr	domainnr
lng_element_tag	domainnr	domainnr
lng_stream_tag	domainnr	domainnr

Lookup tables (parents)

Name	Keypair	Descriptors	Related table	Related alias

Name	**Keypair**	**Descriptors**	**Related table**	**Related alias**
tdmn_tag	tag, tag	tagname, name	lng_tag	tag

lng_tagtypecode

Name

`lng_tagtypecode`—

Attributes

Table type	Code table with a textual primary key
Table alias	ttpc
Primary key field	tagtypecode
Sequence field	None
Hint	
Table described by	description
Fieldorder	tagtypecode, description, isdomain, isvalue, isnote, isreference
Indexes	
Unique Indexes	

Fields

Fieldname	**Field definition**
isdomain	Boolean, 255 (null allowed)
description	None
isvalue	Boolean, 255 (null allowed)
tagtypecode	None
isnote	Boolean, 255 (null allowed)
isreference	Boolean, 255 (null allowed)

Lookup tables (parents)

Childtable	Local key	Foreign key
lng_tag	tagtypecode	tagtypecode

lng_text

Name

`lng_text—`

Attributes

Table type	numeric primary key
Table alias	text
Primary key field	textnr
Sequence field	None
Hint	
Table described by	title
Fieldorder	textnr, title, recordingnr, recording, scannr, scan, description, url, usernr, user, transcription_date, raw_text, languagenr, language
Indexes	languagenr
Unique Indexes	

Fields

Fieldname	Field definition
scannr	Integer, 10 (null allowed)
usernr	Integer, 10 (null allowed)
description	None
language	String, 255 (null allowed)

Fieldname	Field definition
scan	String, 255 (null allowed)
url	String, 255 (null allowed)
title	None
textnr	Integer, 10 (pk)
recording	String, 255 (null allowed)
languagenr	None
transcription_date	Date/time, 255 (null allowed)
recordingnr	Integer, 10 (null allowed)
raw_text	Text, 255 (null allowed)
user	String, 255 (null allowed)

Lookup tables (parents)

Childtable	Local key	Foreign key
lng_proj_text	textnr	textnr
lng_text_tag	textnr	textnr
lng_stream	textnr	textnr

Lookup tables (parents)

Name	Keypair	Descriptors	Related table	Related alias
text_recd	recordingnr, recordingnr	recording, title	lng_recording	recd
text_scan	scannr, scannr	scan, description	lng_scan	scan
text_user	usernr, usernr	user, name	lng_user	user
text_lngg	languagenr, languagenr	language, language	lng_language	lngg

lng_text_tag

Name

`lng_text_tag`—

Attributes

Table type	numeric primary key
Table alias	txtg
Primary key field	text_tagnr
Sequence field	None
Hint	
Table described by	tagname
Fieldorder	text_tagnr, textnr, tag, title, tagname, value, description, note, usernr, user, datestamp
Indexes	textnr, tag
Unique Indexes	

Fields

Fieldname	**Field definition**
usernr	Integer, 10 (null allowed)
description	String, 200 (null allowed)
title	String, 255 (null allowed)
text_tagnr	Integer, 10 (pk)
value	String, 255 (null allowed)
textnr	None
note	Text, 255 (null allowed)
tag	None
user	String, 255 (null allowed)
tagname	String, 255 (null allowed)
datestamp	Date/time, 255 (null allowed)

Lookup tables (parents)

Name	Keypair	Descriptors	Related table	Related alias
txtg_text	textnr, textnr	title, title	lng_text	text
txtx_tag	tag, tag	tagname, name	lng_tag	tag
txtg_user	usernr, usernr	user, name	lng_user	user

lng_user

Name

`lng_user—`

Attributes

Table type	numeric primary key
Table alias	user
Primary key field	usernr
Sequence field	None
Hint	
Table described by	name, email
Fieldorder	usernr, name, affiliationcode, affiliation, email, snailmail, fax, telephone, url
Indexes	
Unique Indexes	

Fields

Fieldname	Field definition
affiliation	String, 255 (null allowed)
fax	String, 255 (null allowed)
usernr	Integer, 10 (pk)
name	None

Fieldname	Field definition
title	String, 255 (null allowed)
telephone	String, 255 (null allowed)
url	String, 255 (null allowed)
email	String, 255 (null allowed)
affiliationcode	String, 5 (null allowed)
snailmail	String, 255 (null allowed)

Lookup tables (parents)

Childtable	Local key	Foreign key
lng_lex	usernr	usernr
lng_text_tag	usernr	usernr
lng_stream_tag	usernr	usernr
lng_proj_user	usernr	usernr
lng_stream	usernr	usernr
lng_scan	usernr	usernr
lng_lex_lex	usernr	usernr
lng_recording	usernr	usernr
lng_lex_tag	usernr	usernr
lng_element_tag	usernr	usernr
lng_document	usernr	usernr
lng_element	usernr	usernr
lng_text	usernr	usernr

Lookup tables (parents)

Name	Keypair	Descriptors	Related table	Related alias
user_affc	affiliationcode, affiliationcode	affiliation, description	lng_affiliationcode	affc

Chapter 18. The objectmodel

This is a simple listing of classes per package. For each class, I list the methods you can count on not to change.

Please consult the actual sourcecode to see what each method does in case it's not completely obvious from the method name. I am in te process of adding pydoc strings, but, well, new features have priority.

18.1. dbobj

Data objects.

18.1.1. dbobj

Classes that represent tables and records.

```
class dbRecord:
  def __init__(self, app, table, fields={}):

  def updateUser(self):

  def next(self):

  def insert(self, checkIntegrity=TRUE):

  def deleteChildren(self, childtable):

  def delete(self, delChildren=FALSE):

  def update(self):

  def getChildren(self, childtable, orderBy = None):

  def hasChildren(self, childTable):

  def createChild(self, childtable, relation):

  def picklist(self, fieldName):

  def getFields(self):

  def getOwnerFields(self):
```

```
  def getFieldValue(self, field):

  def getFieldValueAsString(self, field):

  def getFieldDefinition(self, fieldName):

  def setFieldValue(self, field, value):

  def getPrimaryKey(self):

  def setPrimaryKey(self, pk):

  def getDescriptorColumnName(self, field):

  def getForeignDescriptorColumnName(self, field):

  def getForeignKeyColumnName(self, field):

  def getDescription(self):

  def getLink(self):


class dbTable:

  def __init__(self, app, table, recObj=dbRecord ):

  def select(self, queryRec, orderBy = None):

  def export(self, queryRec, format):
```

18.1.2. appobj

Classes that represent definitions for applications, tables and fields. appobj.dbAppDef is the basis for kuraapp.KuraApp.

```
class dbPair:

    def __init__(self, local, foreign):


class dbChildDef:

    def __init__(self, childTable, keys):
```

```
class dbRelationDef:

    def __init__(self, name, keys, descriptors, rtable, ralias):

class dbFieldDef:

    def __init__( self
                , length = 255
                , pk=FALSE
                , datatype = VARCHAR
                , nullable=TRUE
                , sequence=FALSE
                , owner=TRUE
                , auto=FALSE
                , default=None
                , autoincrement = FALSE
                , relation = None
                , label = None
                , dialog=TRUE
                , inList=TRUE
                , url=FALSE
                , name=""
                , comment=""
                , hint=""
                , readonly=FALSE
                ):

class dbTableDef:

    def __init__( self, tabletype, alias,
                primarykey = None,
                fields={},
                descriptors=["description"],
                childtables={},
                lookuptables=[],
                fieldOrder=None,
                unique_indexes=[],
                indexes=[],
                name="",
                comment="",
                hint="",
                sequencebase=None):

    def orderedFieldList(self):

    def getChildDef(self, childtable):
```

```
class dbAppDef:

    def __init__(self, sql):

    def reconnect(self, host, user, database, password):

    def getLabel(self, tableName):

    def getTableDef(self, tableName):

    def getDefaultValueFor(self, key):

    def createDefaultObject(self, tableName, fields={}, **args):

    def createObject(self, tableName, fields={}, **args):

    def createTableObject(self, tableName):

    def getObject(self, tableName, fields=None, **args):

    def getObjects(self, tableName, fields={}, orderBy = None, **args):

    def getObjectsByRec(self, rec, orderBy = None):

    def getTableLable(self, tableName):

    def addDef(self, **args):

    def toXML(self, outfile):

    def toSQL(self, outfile=sys.stdout):

    def XMLinit(self, infile):


class dbAppObj:
    def __init__(self, recObj, tblObj, label):
```

18.1.3. dbexceptions

Exceptions that can be thrown by the database.

```
class dbError(Exception):
  def __init__(self, errorMessage):
```

```
class dbRecordNotFoundException(dbError):
  def __init__(self, tableName, queryRec ):

class dbTooManyRowsException(dbError):
  def __init__(self, tableName, queryRec ):

class dbRepositoryError(dbError):
  def __init__(self, error, tableName):

class dbModuleError(dbError):
  def __init__(self, error):
```

18.2. kuralib

The lng_XXX classes are based on dbobj.dbRecord and dbobj.dbTable. Only the lng_XXX class are listed here that offer extra functionality above and beyond that offered by `dbobj.dbobj`.

18.2.1. lngapp

The central repository that defines the Kura datamodel.

```
def PK(relation=None): # primary key

def SQ(label=None, dialog=TRUE, inList=TRUE, readonly=TRUE):

def FK(relation,label=None, dialog=TRUE, inList=FALSE,
readonly=FALSE):

def OK(relation,label=None, dialog=TRUE, inList=FALSE,
       readonly=FALSE, default=None):

def FC(relation, length=255,label=None, dialog=TRUE,
       inList=FALSE, readonly=FALSE):

def OC(relation, length=255,label=None, dialog=TRUE,
       inList=FALSE, readonly=FALSE)

def NN(datatype=VARCHAR, length=255,label=None, dialog=TRUE,
       inList=TRUE, readonly=FALSE):

def NO(label=None, inList=TRUE, dialog=FALSE, readonly=TRUE):
```

```
def DS():

def FL( length = 255
      , pk=FALSE
      , datatype = VARCHAR
      , nullable=TRUE
      , sequence=FALSE
      , owner=TRUE
      , auto=FALSE
      , autoincrement=FALSE
      , relation=None
      , label=None
      , dialog=TRUE
      , inList=TRUE
      , default=None
      , readonly=FALSE
      ):

def setRepository(self):
```

18.2.2. lngobj

A mapping from table names to specific table classes.

```
def setObjects(self):
```

18.2.3. docbook

Utility functions to write docbook encoded output.

```
def xmlHeader(doctype):

def periodical_RAW():

def book_RAW():

def collection_RAW():

def website_RAW():

def periodical():
```

```
def book():

def collection():

def website():


def lexemelink():

def textlink():

def glossentry():

def recordingHeader():

def recording(description = False):

def scanHeader():

def scan(description = False):

def element(lexnr, note):

def textHeader(r):

def filter(s):
```

18.2.4. kuraaapp

Central application object for Kura.

```
class KuraApp(dbAppDef):

    def __init__(self,):

    def init(self, backend, **args):

    def isDirty(self):

    def openFile(self, filename):

    def saveFile(self, filename = None):

    def settings(self, **args):
```

```
    def reconnect(self, hostname, username, database, password):

def initApp(backend, **args):

def initCurrentEnvironment(usernr, languagenr, projectnr):
```

18.2.5. lng_abstract_tag

Basis for all classes that represent tags, such as lng_text_tag, lng_stream_tag, lng_element_tag and lng_lex_tag.

```
class AbstractTag(dbRecord):

  def __init__(self, app, table, fields={}):

  def getDescription(self, showifnone=True):

  def getNewDescription(self):
```

18.2.6. lng_elmt

Represents elements in a stream. Note that you can sort elements by sequence in a stream.

```
class lng_element(dbRecord):

  def __init__(self, app, **args):

  def __len__(self):

  def getTags(self):

  def getElements(self):

  def buildElementTree(self):

  def translation(self, cache = True):

  def note(self):

  def getGlosse(self):
```

```
  def setGlosse(self, text):

  def getTag(self, tag):

  def elmtLength(self):

  def asDocbook(self):

  def type(self):

class lng_elements(dbTable):

  def __init__(self, app):

  def select(self, queryRec, orderBy = None):

  def insert(self, streamnr, languagenr, elementTexts=[]):
"""
```

18.2.7. lng_lex

Represents lexemes.

```
class lng_lex(dbRecord):

  def __init__(self, app, **args):

  def getTag(self, tag):

  def getLink(self):

  def asDocbook(self):

class lng_lexemes(dbTable):

  def __init__(self, app):

  def export(self, query, format, *args):
```

18.2.8. lng_lngg

Represents languages.

```
class lng_language(dbRecord):

  def __init__(self, app, **args):

  def getChildLanguages(self):

class lng_languages(dbTable):

  def __init__(self, app):
```

18.2.9. lng_recd

Represents recordings.

```
class lng_recording(dbRecord):

    def __init__(self, app, **args):

    def asDocbook(self):

class lng_recordings(dbTable):

    def __init__(self, app):

    def export(self, query, format, *arg):
```

18.2.10. lng_refs

Represents references.

```
class lng_reference(dbRecord):

    def __init__(self, app, **args):

    def asDocbook(self):

class lng_references(dbTable):
    def __init__(self, app):

    def export(self, query, format, *args):
```

18.2.11. lng_scan

Represents scans

```
class lng_scan(dbRecord):

    def __init__(self, app, **args):

    def asDocbook(self):

class lng_scans(dbTable):

    def __init__(self, app):

    def export(self, query, format, *args):
```

18.2.12. lng_strm

Represents streams. Note that you sort streams by their sequence in a text. The `getElements` function returns only those elements that do not have a parent-element if all == False.

```
class lng_stream(dbRecord):

  def __init__(self, app, **args):

  def translation(self):

  def note(self):

  def getTag(self, tag):

  def getTags(self):

  def getElements(self, all = False):

  def asDocbook(self, asExample = False, simple=True):
```

```
  def getInterlinearLines(self, elements):

  def simpleDocBook(self, asExample):
  def tableDocBook(self, asExample):

class lng_streams(dbTable):

  def __init__(self, app):

  def select(self, queryRec, orderBy = None):

  def insert(self, textnr, languagenr, streamTexts=[]):

  def export(self, query, format, simple = True):
```

18.2.13. lng_tag

Represents tag definitions.

```
class lng_tag(dbRecord):

  def __init__(self, app, **args):

  def getTagTypeCode(self):

  def getDomainTags(self):

class lng_tags(dbTable):

  def __init__(self, app):
```

18.2.14. lng_tdmn

Represents tag domains

```
class lng_tagdomain(dbRecord):

  def __init__( self, app, **args):

  def picklist(self, fieldName):

class lng_tagdomains(dbTable):
  def __init__(self, app):
"""
```

18.2.15. lng_text

Represents texts

```
class lng_text(dbRecord):

    def __init__(self, app, **args):

    def translation(self):

    def getTags(self):

    def getTag(self, tag):

    def removeStreams(self):

    def getStreams(self):

    def asDocbook(self, simple = True):

    def getLink(self):

class lng_texts(dbTable):

    def __init__(self, app):


    def export(self, query, format, simple = True):
```

Bibliography

Steven Bird.

John Nerbonne, *Linguistic Database*, 1997, CSLI, Stanford.

Guido van Rossum.