LANGUAGE DOCUMENTATION & CONSERVATION

Vol. 1, No. 1 (June 2007), pp. 44–57
http://nflrc.hawaii.edu/ldc/

Managing Fieldwork Data with Toolbox and the Natural Language Toolkit

Stuart Robinson,[*] Greg Aumann[†], Steven Bird[+]

[*]*Powerset Inc.*
[†]*SIL International*
[+]*University of Melbourne*

This paper shows how fieldwork data can be managed using the program Toolbox together with the Natural Language Toolkit (NLTK) for the Python programming language. It provides background information about Toolbox and describes how it can be downloaded and installed. The basic functionality of the program for lexicons and texts is described, and its strengths and weaknesses are reviewed. Its underlying data format is briefly discussed, and Toolbox processing capabilities of NLTK are introduced, showing ways in which it can be used to extend the functionality of Toolbox. This is illustrated with a few simple scripts that demonstrate basic data management tasks relevant to language documentation, such as printing out the contents of a lexicon as HTML.

1. BACKGROUND. One of the oldest and best known software tools for field linguistics is Shoebox, a program produced by SIL International (formerly the Summer Institute of Linguistics) that provides linguists with the ability to maintain a lexicon and use it to interlinear gloss texts within an integrated environment. The description of the program provided on the Shoebox homepage (http://www.sil.org/computing/shoebox/) explains its purpose and the motivation for its name:

> Shoebox is a computer program that helps field linguists and anthropologists integrate various kinds of text data: lexical, cultural, grammatical, etc. It has flexible options for sorting, selecting, and displaying data. It is especially useful for helping researchers build a dictionary as they use it to analyze and interlinearize text. The name Shoebox recalls the use of shoe boxes to hold note cards on which definitions of words were written in the days before researchers could use computers in the field.

In 2003 the program was re-released with a new name, Toolbox. Toolbox is fundamentally the same as Shoebox, to the extent that it uses the same basic data format and user interface, but Toolbox differs from its predecessor Shoebox in two important respects. First, Toolbox continues to be developed, although it is not officially supported. (Development of Shoebox has stopped, as has support for it, and users of the program should upgrade to Toolbox.) Second, Toolbox provides a number of useful new features, such as the ability to export data in XML format and support for Unicode data storage.

2. DOWNLOADING AND INSTALLING TOOLBOX. Toolbox is freeware and can be downloaded from the Toolbox homepage (http://www.sil.org/computing/toolbox/). As noted there, the program runs only on Windows. For users of other operating systems, the only option is to run the program using a Windows emulator. This option works fairly well,

E-ISSN 1934-5275

thanks to the advanced state of many Windows emulation programs, and more information about installing Toolbox on Linux and MacOSX can be found on the Toolbox download homepage.[1]

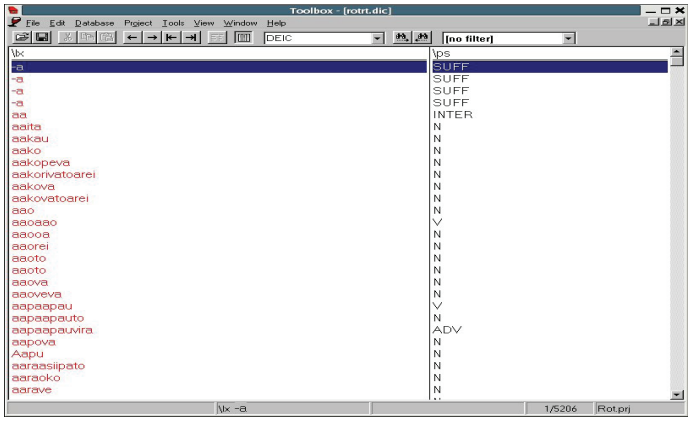
The installation of Toolbox is fairly straightforward, particularly for Windows. The biggest challenge is the initial setup of a project. Although it is possible to do this from scratch by carefully following the documentation, a better approach is to start with a sample project and modify it for one's own purposes. Changes can then be introduced gradually and incrementally. It is a good idea to back up the current version of a project before modifying it, since this will ensure that there is a working version to fall back on in the event that the changes introduce problems that cannot readily be fixed.

For a sample project, there are two options: the ones included with the Toolbox installation package or those from the MDF project (http://www.sil.org/computing/toolbox/ToolboxMDFFields.zip). The benefit of using the latter is that it simplifies data export as well as interactions with other SIL software, such as LexiquePro (http://www.lexiquepro.com/). It will also make it easier to upgrade to the next generation of SIL linguistic software.

3. TOOLBOX BASICS. A full overview of what Toolbox has to offer can be found in the documentation that comes with the program. Here we will cover the highlights using screenshots from the authors' own Toolbox projects. We will start by looking at how lexicons are managed in Toolbox.

In Toolbox, a lexicon consists of a collection of entries, each with a varying number of user-defined data fields. The main lexicon window shows the entries of a lexicon in alphabetical order. The following screenshot shows the browsing window for a lexicon of the Central dialect of Rotokas, a Papuan language spoken in Bougainville (Firchow 1987, Robinson 2006):

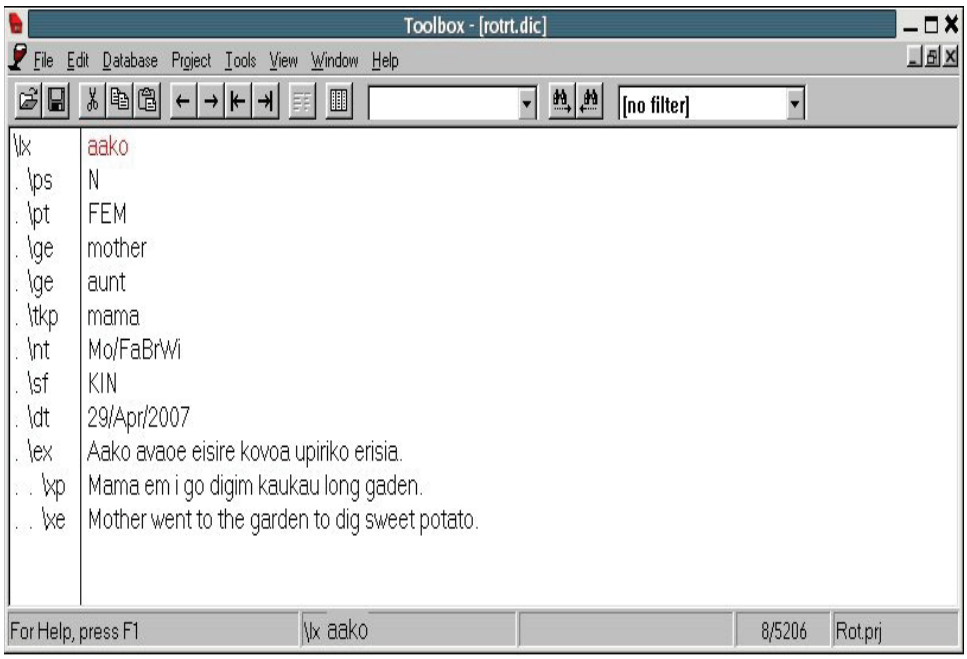
FIGURE 1: Browsing a Toolbox lexicon



1 The first author runs Toolbox on Red Hat Linux (http://www.redhat.com/) using WINE (http://www.winehq.com/), the second on Ubuntu Linux (http://www.ubuntu.com/) using VMware (http://www.vmware.com/products/player/), and the third on Mac OSX (http://www.apple.com/macosx/) using Parallels (http://www.parallels.com/).

By default, only the main lexeme field is displayed, but others can be added to the display, as in the screenshot, where each lexeme is displayed with its part-of-speech field. By double-clicking on a single entry, its full contents can be displayed, as shown below for the entry *aako* (a noun that means "mother").

FIGURE 2: Browsing the entries in a Toolbox lexicon



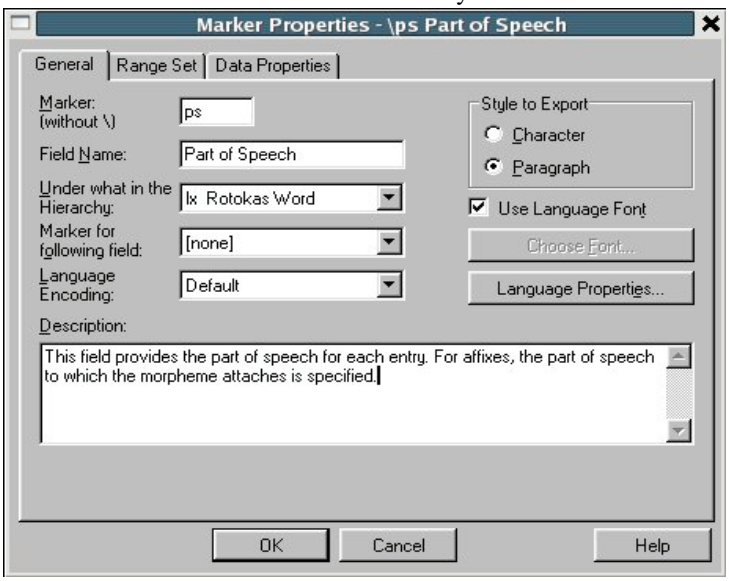
The entry consists of a number of different fields of information: the lexeme for the entry, *aako*; its part of speech, "N"; its part of speech subtype, "FEM" (Feminine); its English gloss (used for interlinear glossing), "mother" or "aunt"; its gloss in Tok Pisin (the lingua franca of Papua New Guinea), "mama"; notes concerning the entry, "Mo/FaBrWi" (the kinship relations covered by the term—that is, "Mother" and "Father's Brother's Wife"); its semantic field, "KIN"; the date last edited, "29/Apr/2007", and an example sentence consisting of three fields: the original Rotokas, the English translation, and the Tok Pisin translation.

In the default display shown above, the program displays the data for an entry in two columns: on the left are the field markers, and on the right are the field values. (Note that the hierarchical relationship between fields is also shown: "xe" and "xp" are children of "xe" and all other fields are children of "lx".) Alternate views of the data are possible. For example, below we find the same data displayed with additional information: the field markers are accompanied by their description on the left, while the field values are displayed in their usual location on the right:

FIGURE 3: An entry in a Toolbox lexicon

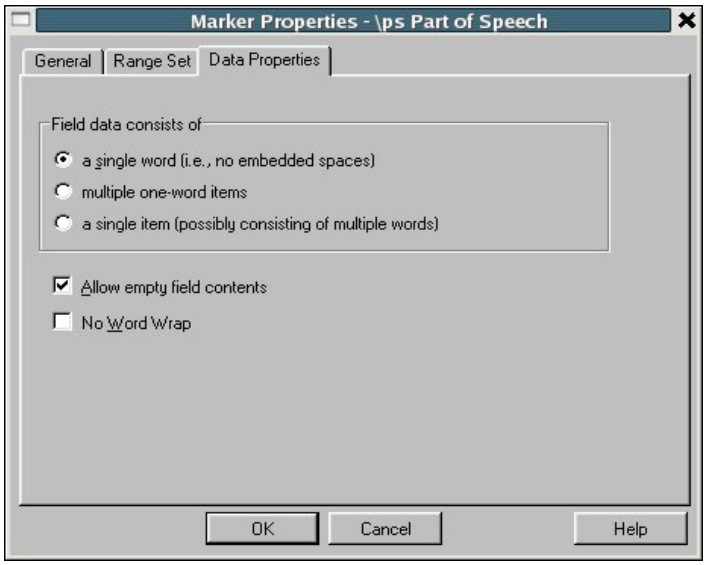
Toolbox provides more fine-grained control over the contents of the fields in an entry, allowing the values of a field to be restricted in various ways. In the "Database" menu, under "Properties", various properties of a field can be defined. Below we see the default tab displayed when the marker properties for the "ps" field is selected:

FIGURE 4: Another view of an entry in a Toolbox lexicon



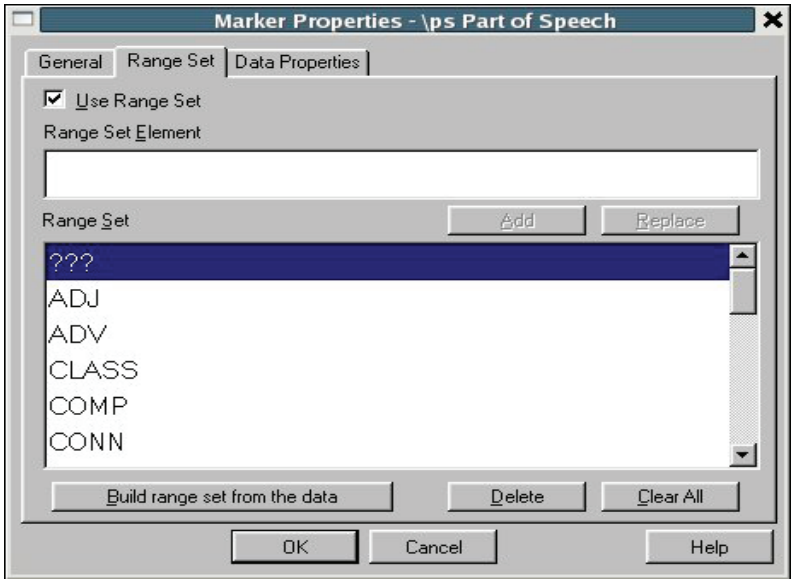
If the "Data Properties" tab is selected, we see that a value for the field is not required (empty field contents are allowed), but is restricted to a single word (no spaces allowed):

FIGURE 5: Data properties for a field in a Toolbox lexicon



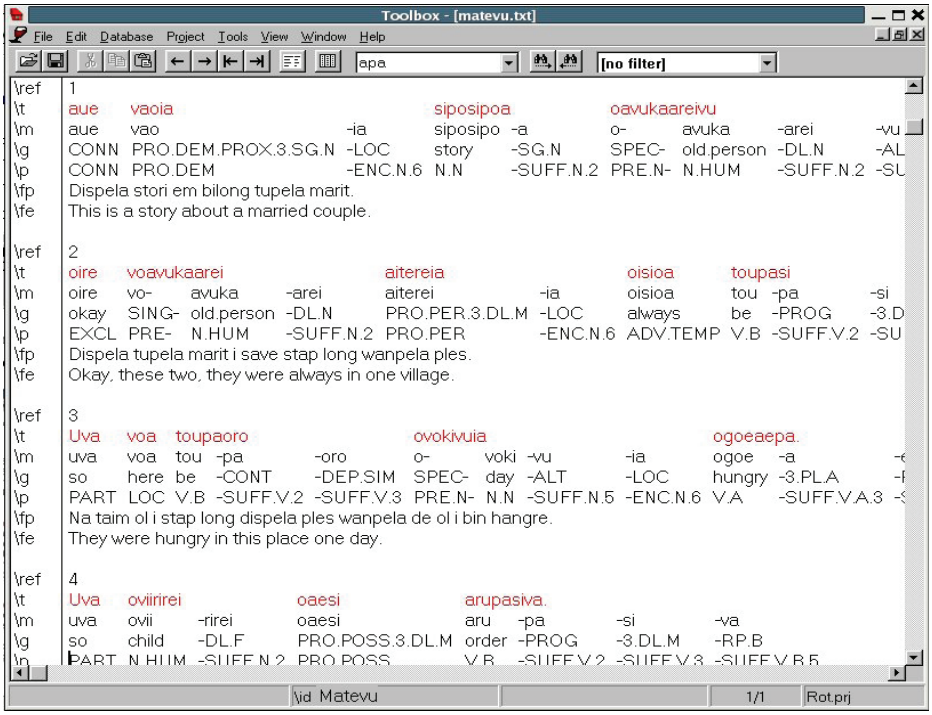
In the "Range Set" tab of the same "Marker Properties" window, we see that the values for the "ps" field are restricted to a fixed set of options:

FIGURE 6: The range set for a field in a Toolbox lexicon



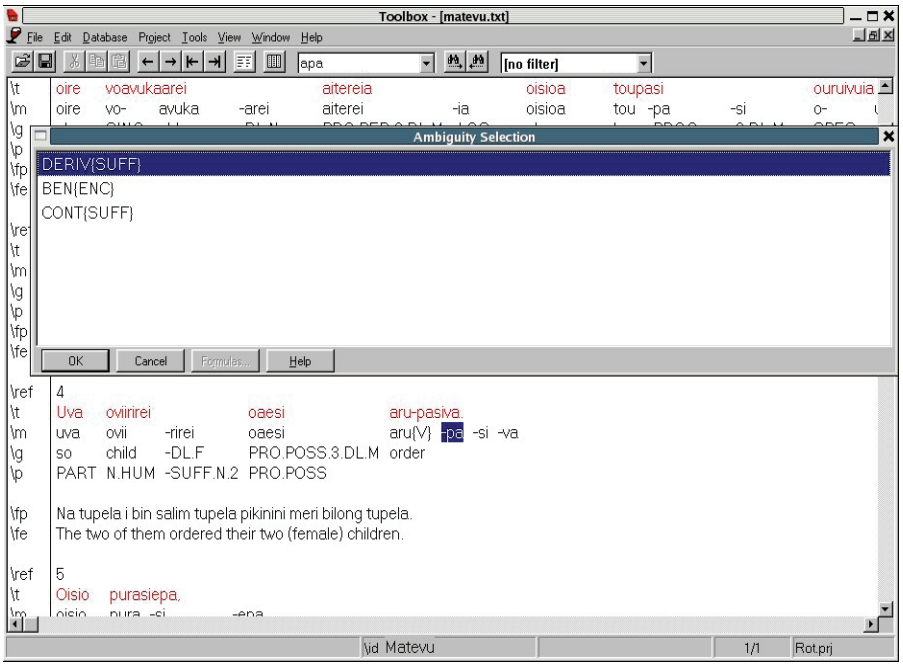
One of the advantages of storing a lexicon in Toolbox is that it can also be used to create interlinear glossed texts, which break down each line of a text into numerous tiers of analysis (see Bow, Hughes, and Bird 2003 for a general model of interlinear glossed text), as illustrated for a Rotokas folk tale recorded by Irwin Firchow (a missionary linguist who first described the language) (Firchow 1974):

FIGURE 7: A Toolbox interlinear glossed text



Each line in the text consists of multiple user-defined fields (sometimes called "tiers"): the raw line, a morpheme-by-morpheme breakdown of the line, the gloss for each morpheme, the part of speech for each morpheme, the Tok Pisin gloss, and the English gloss. Toolbox has a built-in morphological parser that automatically interlinearizes a line of text by parsing words into morphemes and looking them up in the lexicon to populate each field in the interlinear gloss. For example, in the Rotokas text shown above, the "m" field in the text comes from the "lx" field of the lexicon and the "g" tier comes from the "ge" field of the lexicon. When there is ambiguity in the morphemic analysis, the program prompts the user to disambiguate, as illustrated below, where various options for the analysis of the morpheme *-pa* from the fourth line of the Rotokas text are presented:

FIGURE 8: Morpheme disambiguation during the parsing of a word in a Toolbox interlinear glossed text



4. THE STRENGTHS AND WEAKNESSES OF TOOLBOX. Toolbox is a mature program that is very full-featured. It handles many of the tasks performed by the everyday working linguist, such as searching for entries, constraining how they can be edited, tracking when changes are made by automatically updating a timestamp field, analyzing texts by breaking down the words into morphemes and looking them up in the lexicon, etc. It has a fairly large user base, including a mailing list for the discussion of Toolbox-related issues (http://groups.google.com/group/ShoeboxToolbox-Field-Linguists-Toolbox), where questions about the program can be posted.

Two major strengths of the program are that:

- Its data format is very flexible. It is possible to use Toolbox to store all kinds of data that the developers never envisioned. It is therefore easy to incorporate idiosyncratic features of any language into a lexicon.
- All data and metadata are stored in text files that use a relatively simple-to-understand format. This means that it is easy to write external software that reads and even modifies these files to extend the built-in functionality of Toolbox.

Despite the above-mentioned strengths of the program and its general popularity, Toolbox still has a few limitations that are worth pointing out, such as the following:

- The initial set-up of a project is not straightforward, and many linguists find it

daunting. This is particularly true with sorting and interlinearization. There is a need for a set-up wizard that would guide a user through the creation of a new project.

- One of the most commonly voiced complaints about Toolbox concerns the integration of the lexicon and interlinear glossed texts. When the lexicon is revised, those changes do not propagate back to the previously glossed texts, and the mechanism for updating an outdated text from a revised lexicon is not foolproof.
- Support for the enforcement of data consistency is weak. For example, the range sets discussed above are only applied to individual entries when they are saved (i.e., when they are created or edited). If changes are made to the range set, they are not automatically applied to all entries.
- There is no macro language to automate common operations.

Although the functionality of Toolbox will be increased over time, and future versions may provide additional features that current versions lack, there will always be tasks that were not envisioned by its designers or that are too specific to a particular audience to warrant the investment of time required to add them to the program. For this reason, it is useful to be able to write custom programs that manipulate the contents of a Toolbox lexicon or text.

It is relatively easy to parse a Toolbox data file and extract its contents using a scripting language with good string-processing capabilities, such as Perl (http://www.perl.com/), Python (http://www.python.org/), or Ruby (http://www.ruby-lang.org/). Although one could write custom code to manipulate Shoebox data, it is much easier to use an existing code library that provides that ability. Libraries of this sort can be found for at least two popular scripting languages, Perl and Python. For Perl, Sean M. Burke has developed a module for processing Toolbox data, which is described in *Making Dictionaries with Perl* (http://www.perl.com/pub/a/2004/03/25/dictionaries.html), and Martin Hosken has written a number of Shoebox utilities (http://scripts.sil.org/SHUtils-manual).

For Python, the authors have developed a module for processing Toolbox data, which is distributed as part of a larger suite of tools known as the Natural Language Toolkit (NLTK). In the following section, we will describe NLTK and provide a brief introduction to its Toolbox processing capabilities.

5. PROCESSING TOOLBOX DATA WITH THE NATURAL LANGUAGE TOOLKIT. The Natural Language Toolkit (NLTK) (http://nltk.org) is a comprehensive set of tools for natural language processing. It is written in the programming language Python and includes tools for a variety of tasks, including tagging, chunking, and parsing (Loper and Bird 2002, 2004). Here we will concentrate on only one component, the module nltk.corpus.toolbox, which provides tools for processing Toolbox data. (More information about what NLTK has to offer can be found on the hompage. Since the project is pedagogically-oriented, the documentation includes numerous tutorials and exercises.)

On the installation page (accessible from the homepage), there is information about how to install and run NLTK on various operating systems (Windows, MacOS, Linux).

Once NLTK has been installed, it can be tested using Python's interactive mode (which allows Python code to be typed directly and the results obtained immediately, in the style of a calculator). This is illustrated below in a Unix-style command-line window, where the main code library is imported for use (if NLTK is not installed, an error will result):

```
$ python
Python 2.4.3 (#1, Oct 23 2006, 14:19:47)
[GCC 4.1.1 20060525 (Red Hat 4.1.1-1)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from nltk.corpus import toolbox
>>>
```

In order to manipulate Toolbox data effectively using NLTK, it is necessary to know at least the basics of Python programming. A good starting point is python.org (http://www.python.org/), the official web site for the programming language. There are also a number of useful books in print (e.g., Lutz and Ascher 2004, Mertz 2003). Here we will illustrate the workings of NLTK using fairly basic Python constructs so that even readers unfamiliar with the language can follow the discussion.

5.1 THE TOOLBOX DATA FORMAT. Although we have looked mainly at Toolbox data through the graphical user interface, it is also possible to look at this data in its raw form using a word processor (e.g., Notepad, MS Word, etc.) or text editor (e.g., vi, emacs, etc.). The data format used by Toolbox is called "standard format marker" (SFM), a markup format that predates Shoebox. Although standard format is easy to read and interpret, it is not as easily parsed as modern mark-up schemes, such as XML, which is easily parsed and validated using a wide variety of freely available tools. Here is what the sample entry from the Rotokas lexicon that was illustrated above looks like in its raw form:

```
\lx aako
\ps N
\pt FEM
\ge mother
\ge aunt
\tkp mama
\nt Mo/FaBrWi
\sf KIN
\dt 29/Apr/2007
\ex Aako avaoe eisire kovoa upiriko erisia.
\xp Mama em i go digim kaukau long gaden.
\xe Mother went to the garden to dig sweet potato.
```

The entry consists of a number of fields. Each field has two parts: a field marker and a field value. The field marker is preceded by a backslash and is separated from the field value by white space. For example, the first field consists of the field marker "lx" with the field value *aako*. The same basic markup scheme is used for lexicons and texts. The raw data that makes up the third line from the text shown above is illustrated below:

```
\ref 3
\t Uva  voa  toupaoro               ovokivuia
ogoeaepa.
\m uva  voa  tou -pa       -oro     o-     voki -vu       -ia
ogoe   -a          -epa
\g so   here be  -CONT     -DEP.SIM  SPEC-  day  -ALT     -LOC
hungry -3.PL.A     -RP.A
\p PART LOC  V.B -SUFF.V.2 -SUFF.V.3 PRE.N- N.N  -SUFF.N.5 -ENC.N.6
V.A    -SUFF.V.A.3 -SUFF.V.A.5
```

```
\fp Na taim ol i stap long dispela ples wanpela de ol i bin hangre.
\fe They were hungry in this place one day.
```

The information about the nature of the various fields in a lexicon or text (see earlier screenshots) is not found in the data file itself. Instead, it is found in separate "settings" files that provide information about the contents of the lexicon and text. (These are sometimes called metadata files since they contain data about data.) Toolbox uses these files to interpret the contents of a lexicon. For example, here is the metadata for the part of speech field:

```
\+mkr ps
\nam Part of Speech
\lng Default
\rngset ??? ADJ ADV CLASS COMP CONN DEM ENC EXCL FFP INFIX INTER LOC N
NUM PART POST PPRO PRE PRO RPRO SUFF V
\SingleWord
\mkrOverThis lx
\-mkr
```

Note that it describes the range set for the field, which restricts the number of possible values the field can take. It is therefore unwise to modify them by hand in the absence of expert information about their contents. A detailed discussion of the format and contents of these files goes beyond the scope of this introduction to Toolbox, but fortunately there is a good deal that can be done with the raw lexicon file without having to worry about metadata.

5.2 PARSING AND MANIPULATING TOOLBOX DATA. NLTK provides tools for parsing and manipulating Toolbox lexicons and texts. Here we will concentrate on a method of parsing a lexicon into a tree structure using the ElementTree module, which has become the standard Python interface for reading, writing, and manipulating XML (eXtensible Markup Language) and was added to Python's standard library as of version 2.5. Since standard format is a less sophisticated markup language, a standard XML interface for it makes handling it significantly easier. It also greatly facilitates conversion between standard format and XML, which is an important consideration, given that a great deal of recently developed linguistics software uses XML formatted data. (Note that NLTK includes a copy of the ElementTree code to simplify installation for users of Python versions prior to 2.5.)

The first step in processing a Toolbox lexicon with NLTK is to parse the lexicon file. To simplify the discussion, we will use the Rotokas lexicon shown above. An excerpt from that lexicon is available in NLTK as a sample data file called "rotokas.dic". We can use the *toolbox.parse_corpus()* method to access "rotokas.dic" and load it into an ElementTree object, which is saved as the variable *lexicon*:

```
>>> from nltk.corpus import toolbox
>>> lexicon = toolbox.parse_corpus('rotokas.dic')
```

The contents of the lexicon object can be accessed by indices or by paths. Indices essentially treat a lexicon as a list of entries, each of which consists of a list of fields. Individual items in the list can be accessed by number (where counting begins with zero). Continuing the session from above, the fourth entry from the lexicon can be accessed

with *lexicon[3]* and saved as the variable *entry*. The first field from the entry is accessed with *entry[0]* and saved as the variable *field*. The field has two attributes: "tag" (the field's marker) and "text" (the field's value).

```
>>> entry = lexicon[3]
>>> field = entry[0]
>>> field.tag
'lx'
>>> field.text
'kaa'
```

Alternatively, the syntax for nested lists can be used: *lexicon[3][0]* accesses the first field of the fourth entry, as illustrated below:

```
>>> lexicon[3][0]
<Element lx at 77bd28>
>>> lexicon[3][0].tag
'lx'
>>> lexicon[3][0].text
'kaa'
```

We can iterate over all the fields of a given entry to print out the contents of the lexicon:

```
>>> for entry in lexicon :
...   for field in entry :
...     print "\\lx %s %s" % (field.tag, field.text)
```

The second way to access the contents of the lexicon object uses paths. The lexicon is a series of record objects, each containing a series of field objects, such as "lx" and "ps". We can conveniently address all of the lexemes using the path "record/lx". Here we use the *findall()* function to search for any matches to the path "record/lx", and we access the text content of the element:

```
>>> for lex in lexicon.findall('record/lx') :
...   print lex
```

Using some of the techniques discussed above, NLTK provides the ability to write Python scripts that give more fine-grained control over the presentation of data. Although Toolbox has the built-in capacity for the generation of dictionaries, it is still primarily oriented towards print dictionaries. However, electronic publication of materials on the web is increasingly common, and it is therefore useful to be able to take a Toolbox lexicon and format it as HTML for web display. For example, an entry might be formatted as HTML in the style of a dictionary, as follows:

```
<p><b>aako</b> n. "mother, aunt"</p>
```

Producing such HTML code automatically from a Toolbox lexicon is fairly straightforward using NLTK. The following sample script will print out the contents of the Rotokas dictionary in tabular format. That is, the output will be HTML consisting of a table in which each entry is a row with three columns: the lexeme, its part of speech, and its gloss.

```
from nltk.etree.ElementTree import ElementTree, Element, SubElement,
tostring
from nltk.corpus import toolbox

lexicon = toolbox.parse_corpus('rotokas.dic')
```

```
html = Element("table")
html.text = "\n  "
for entry in lexicon[0:10]:
    lx = entry.findtext('lx')
    ps = entry.findtext('ps')
    ge = entry.findtext('ge')
    row = SubElement(html, "tr")
    for ent in (lx, ps, ge):
        ent_elem = SubElement(row, "td")
        ent_elem.text = ent
    row.tail = "\n  "
row.tail = "\n"
print tostring(html)
```

The output of the script is the first 10 entries of the dictionary formatted as HTML (which can be saved into a file and viewed in a web browser):

```
<html><body><table>
  <tr><td>kakae</td><td>???</td><td>small</td></tr>
  <tr><td>kakae</td><td>CLASS</td><td>child</td></tr>
  <tr><td>kakaevira</td><td>ADV</td><td>small-like</td></tr>
  <tr><td>kakapikoa</td><td>???</td><td>small</td></tr>
  <tr><td>kakapikoto</td><td>N</td><td>newborn baby</td></tr>
  <tr><td>kakapu</td><td>V</td><td>place in sling for purpose of
carrying</td></tr>
  <tr><td>kakapua</td><td>N</td><td>sling for lifting</td></tr>
  <tr><td>kakara</td><td>N</td><td>arm band</td></tr>
  <tr><td>Kakarapaia</td><td>N</td><td>village name</td></tr>
  <tr><td>kakarau</td><td>N</td><td>frog</td></tr>
</table></body></html>
```

The advantage of using a Python script to generate HTML code automatically is that the output formatting can be changed without touching the underlying data format. Furthermore, a Python script can be custom tailored to a specific data format, which means that it can be made to work even for lexicons (such as the one for Rotokas) that do not conform to the MDF standard. For example, entries in the Rotokas dictionary have part of speech information in two separate fields: “ps” and “pt”. The script above can be easily modified so that the HTML display produced by it includes both part-of-speech fields, separated by a dot (e.g., ‘N.FEM’):

```
from nltk.etree.ElementTree import ElementTree, Element, SubElement,
tostring
from nltk.corpus import toolbox

lexicon = toolbox.parse_corpus('rotokas.dic')

html = Element("html")
body = SubElement(html, "body")
table = SubElement(body, "table")
table.text = "\n  "
for entry in lexicon[0:10]:
    lx = entry.findtext('lx')
    ps = entry.findtext('ps')
    pt = entry.findtext('pt')
    if pt :
        ps = "%s.%s" % (ps, pt)
    ge = entry.findtext('ge')
    row = SubElement(table, "tr")
    for ent in (lx, ps, ge):
        ent_elem = SubElement(row, "td")
        ent_elem.text = ent
```

```
    row.tail = "\n  "
row.tail = "\n"
print tostring(html)
```

The output of the script is the first 10 entries of the dictionary formatted as HTML with full part-of-speech information:

```
<html><body><table>
  <tr><td>kakae</td><td>???</td><td>small</td></tr>
  <tr><td>kakae</td><td>CLASS</td><td>child</td></tr>
  <tr><td>kakaevira</td><td>ADV.MANNER</td><td>small-like</td></tr>
  <tr><td>kakapikoa</td><td>???</td><td>small</td></tr>
  <tr><td>kakapikoto</td><td>N.HUM</td><td>newborn baby</td></tr>
  <tr><td>kakapu</td><td>V.B</td><td>place in sling for purpose of
carrying</td></tr>
  <tr><td>kakapua</td><td>N.NT</td><td>sling for lifting</td></tr>
  <tr><td>kakara</td><td>N.NT</td><td>arm band</td></tr>
  <tr><td>Kakarapaia</td><td>N.PN</td><td>village name</td></tr>
  <tr><td>kakarau</td><td>N.FEM</td><td>frog</td></tr>
</table></body></html>
```

We have provided only a very brief introduction to the Toolbox processing capabilities of NLTK, concentrating on the processing of lexicons rather than texts. For a more in-depth tutorial, consult the chapter Accessing and Analyzing Linguistic Field Data from the forthcoming book *Natural language processing in Python* (Bird et al. 2007). There is a mailing list for NLTK users, where general questions can be posted: nltk-devel.

6. CONCLUSION. As the publication of primary materials on the web becomes more commonplace, it is necessary for field linguists to organize their materials in such a way that they can be easily processed and shared with other researchers. This is especially important for research on endangered languages, since the materials gathered may represent the only record of a language before it ceases to be spoken. Toolbox is an excellent tool for this purpose. It provides an integrated workspace for the development and maintenance of lexicons and texts. On balance, its strengths outweigh its limitations, especially for researchers who have a basic command of a scripting language such as Python, which together with NLTK can be used to extend its functionality.

References

BIRD, STEVEN, EWAN KLEIN, and EDWARD LOPER. 2007. *Natural language processing in Python*. http://nltk.org/.

BIRD, STEVEN, and GARY SIMONS. 2003. Seven dimensions of portability for language documentation and description. *Language* 79(3):557–82.

BOW, CATHERINE, BADEN HUGHES, and STEVEN BIRD. 2003. Towards a general model of interlinear text. In *Proceedings of the EMELD Conference 2003: Digitizing and annotating texts and field recordings*. http://linguistlist.org/emeld/workshop/2003/proceedings03.html.

FIRCHOW, IRWIN B. 1974. *Rotokas songs*. Ukarumpa: Summer Institute of Linguistics.

FIRCHOW, IRWIN B. 1987. Form and function of Rotokas words. *Language and Linguistics in Melanesia* 15:5–111.

MERTZ, DAVID. 2003. *Text processing in Python*. Reading, MA: Addison-Wesley.

LOPER, EDWARD, and STEVEN BIRD. 2002. NLTK: The Natural Language Toolkit. In *Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*. Philadelphia, July 2002, Association for Computational Linguistics. http://arxiv.org/abs/cs/0205028.

LOPER, EDWARD, and STEVEN BIRD. 2004. NLTK: The Natural Language Toolkit. In *Proceedings of the ACL Demonstration Session*, Barcelona, July 2004. http://www.ldc.upenn.edu/sb/home/papers/nltk.pdf.

LUTZ, MARK, and DAVID ASCHER. 2004. *Learning Python*. Sebastopol, CA: O'Reilly Press.

ROBINSON, STUART. 2006. The phoneme inventory of the Aita dialect of Rotokas. *Oceanic Linguistics* 45(1):206–9.

Stuart Robinson
stuart@zapata.org

Greg Aumann
greg_aumann@sil.org

Steven Bird
sb@csse.unimelb.edu.au