Typesetting Interlinear Text
An Interlinear text to RTF converter

Martin Hosken,
SIL Non-Roman Script Initiative (NRSI)

Introduction

The laying out and typesetting of interlinear text is one of the more problematic aspects of linguistics. It has been one that I have been addressing, on and off, for quite a few years, and have only really seen any success now. This program is designed to aid with that task by automating the process of converting interlinear text, held in Shoebox, into RTF for inclusion in a Word document. There are actually two programs as part of this system. The first converts the interlinear text into an intermediate form of standard format and the second is an enhanced Shoebox Standard Format to RTF converter.

This paper will discuss the installation and use of the programs and then go on to a discussion of the complexity of the interlinear typesetting problem and the approach taken in this system. This may be of use for those pushing the extremes of interlinearising.

One pre-requisite is that users have a program for handling `.tar.gz` files on their system. WinZip is amply sufficient for the task, as are many of the modern zip handling type programs. You will also need a 32-bit Windows environment with support for long filenames.

Installation

The software is written in Perl, an interpretted language designed with text processing and system interaction in mind. This section will lead you through installing Perl, pmake and then the interlinear processing system.

Installing Perl

The company ActiveState have helped the Perl user community greatly by producing a very nice installation kit for Perl. It is free, and it is called: ActivePerl. It is available from their website: http://www.activestate.com/ActivePerl. Download the installation kit and run it to install Perl. Feel free to follow the defaults, unless you have other plans.

A basic Perl installation takes up about 30Mb, with most of it being support libraries, and help files.

Installing the Interlinear system

Installing the interlinear system is not difficult. You extract everything from a .tar.gz file, run `perl Makefile.PL`, run `pmake install`. The instructions are in the readme.txt file, but we will include them here:

- Open the installation file using WinZip, or the like.
- Extract the files into a temporary directory and open a DOS box (Command prompt) and change to that directory
- Follow the instructions in the readme file, adjusting `make` to `pmake`:
    - Type: `perl Makefile.PL`
    - Type: `pmake install`
    - Type: `pmake realclean`

This process will add various files to your Perl support libraries and also add two batch files to your Perl binary directory (thus making them available in your PATH).

Notice that this whole installation process works across platforms. In the case of a Mac you should be able to install by installing the CPAN module first and then dropping the .tar.gz file on the install_me droplet. On Unix, you install as per above, but can use `make` instead of `pmake`.

Useage

In this section we examine how to use this typesetting system and give an example from the Shoebox sample files.

The programs use knowledge of the database structure to work out what to do with each marker in a database. For this reason, every program must be told where the appropriate Shoebox settings directory is for this database. This is the directory in which the project file (`.prj`) is held which you use when working with your interlinear database.

Processing an interlinear file to convert it to an intermediate `.sfm` format is straightforward. But the conversion of that `.sfm` file to RTF needs a modicum of care. The RTF conversion process uses the information in the database type file (`.typ`) to indicate how various markers should be converted to styles in the final output. There are three things that need to be right about this information, otherwise you may get some very odd looking output.

- Add a marker: `_RTF` with a marker name of `_RTFONLY_` (case is important), which is used by the system to pass pure RTF from the interlinear converter through to the output.
- Add a marker: `_INT` with a marker name of `Interlinear Block`. This should be a paragraph marker. The marker name (and so style name) is unimportant and indicates the name of the paragraph style used for interlinear text blocks.
- All the markers used in the interlinear process (e.g. `\t`, `\m`, `\ps`, etc.) should be set to be *character* styles rather than paragraph. This is important to ensure that you don't end up with paragraph marks in the middle of the interlinear blocks.

Nearly all problems with garbage output are due to not following these instructions in the database properties before converting to RTF.

Once the database type file is correct, we can start to process some data. There are two programs used:

`shintr -s` *`directory inputfile outputfile`*

which converts the *inputfile* interlinear database into a temporary *outputfile* ready for RTF conversion. *`directory`* indicates where the Shoebox settings may be found (and so the database type file). A quick summary help may be found by typing: `shintr` on its own.

The next process takes the temporary file from `shintr` and creates an RTF file from it:

`sh_rtf -s` *`directory inputfile outputfile`*

The command line is of the same structure as the other program. This takes the *inputfile* and converts it to an *outputfile* in RTF format (so it would help for it to have a `.rtf` extension). The data need not be output from `shintr`, but output from `shintr` must be processed using `sh_rtf` rather than the normal RTF output from Shoebox, due to the enhancements provided in `sh_rtf`.

Walkthrough

The following is a walkthrough example based on the directory `...\Shoebox\Samples\Adapt\Adapt2b` which contains an example database which we can play with for output to RTF.

- Launch the `.prj` file (`EngAdapt.prj`) which will open the various databases involved in this example.
- Go to **P**rojects/**D**atabase Types... (from the Projects menu), and select the Interlinear database type and press **M**odify...
- Click **A**dd... and add a new marker: `_INT` with a field name of `Interlinear Block`. You may leave everything else about the marker as you find it. Click **O**k.
- Click **A**dd... and add a new marker: `_RTF` with a field name of `_RTFONLY_` (one word, all in capitals and the underscore letter). Again, you may leave everything else as you find it. Click **O**k.
- For each of the markers: `e`, `e1`, `e2`, `mb`, `ps`, `t` repeat the following process:
    - Click on the marker
    - Press modify
    - Press the `Character` radio button under Style to Export
    - Press **O**k.
- Press **O**k to exit the database type properties dialog, and click **C**lose to exit the database types dialog.
- Exit Shoebox (**F**ile/**E**xit)

Now that we have set up the database type file for this project, we can convert interlinear data as often as we want without having to go through that process again.

- Open a DOS box (Command Prompt) and change to the directory we are working in (`.../Shoebox/Samples/Adapt/Adapt2b`)
- Type: `shintr -s . mark14b.txt temp.sfm`
- Type: `sh_rtf -s . temp.sfm mark14b.rtf`
- Type: `start mark14b.rtf` to start Word and open the RTF file.

Assuming everything has gone to plan, you should see various blocks of interlinear text interspersed with comments about the various rearrangement and generation rules used in the example.

Approach Used

There are three main approaches used to rendering interlinear text in RTF. They are:

- Use monospaced fonts and use the monospacing to line things up
- Use tables with each interlinear element being a cell
- Use equation fields which allow the stacking of elements, with each interlinear block appearing as one letter.

We use the latter approach in this system for the following reasons:

- The interlinear blocks automatically wrap without having to measure any widths.
- They are much more manipulable when laying out text.
- They can be used within tables (as opposed to being tables in their own right)

Therefore, if you were to click in the example, on any interlinear element, the whole block would select, but just the one block. Also, if you were to change the viewing options to show field codes, you would see a horrific mess, which is the underlying work that the interlinear converter has done for you. Perhaps it would be better to switch back to normal view fairly smartly!

Problems

Due to a bug in Word[1], whereby it is not possible to store commas or parentheses in an EQ field (despite help's instructions to the contrary), `shintr` deletes any commas and parentheses it finds. You may indicate that it should not do this by using the `-c` option, whereupon, if you are not using Word 2000, all the commas and parentheses should reappear, hopefully not messing up the layout.

If you look carefully at the example output, for example under the word **de** in the first paragraph, you will see that a whole phrase has been rendered under one word rather than across the whole phrase. While this is not ideal, it is an artifact of the way that EQ fields work and the complexity of phrase level interlinear conversion, a subject we will address in a later section.

Advanced Useage

In this section we look at the various ways in which the tools we have discussed may be configured.

shintr

`shintr` has various command line options to control such things as inter-column spacing between interlinear blocks and inter-line spacing between the elements in a block. Both of these values are

[1] I have only seen this bug in Word 2000.

specified as an integer value of points (1/72 of an inch). The defaults are given, but may be overridden using the command line options `-h` and `-l` respectively. Thus, the command line:

```
shintr -h6 -l 8 -s. inputfile outputfile
```

does nothing beyond the normal. Notice that command line options can have space between the option and the value, or not. It is unimportant.

`shintr` also allows for various markers to be ignored (so that you only display the interlinear lines of interest). This is done using the `-x` option which is followed by a list of markers separated by any non-word forming characters (commas will do!)

sh_rtf

`sh_rtf` is a powerful program for converting Shoebox data into RTF. It emulates the Shoebox output process and enhances it in a couple of areas.

The primary enhancement, apart from the magic style `_RTFONLY_`, is to provide some rudimentary support for tables. There are two ways in which this is done. The first is that if there is a style called `Free Translation` as well as `Interlinear Block` then if you use the `-c` option you can specify the width of the free translation column, which will appear as column two, and the width of column 1 for the interlinear block will be set such that the total table width is 7".

The second approach is to embed extra styling information into the marker specification. Since Shoebox merrily ignores and removes any markers it does not understand in a database type file, we embed the information in the marker description. Embedded information is stored in the description in the form of XML empty tags. For example, to indicate which column of a table a marker's paragraph should be inserted into, add the following to the description:

```
<col num="number"/>
```

2 other tags are also required to help in the specification:

```
<totalcols num="number"/>
<colwidths num="number, number, ..."/>
```

`totalcols` indicates the number of columns in the table. This is necessary so that for any particular column, the converter can work out what to do at the end of the cell, whether this is the end of a row, and whether and how many blank cells to insert if intervening cells are not filled.

`colwidths` is also needed for every marker in a table. It specifies the width (in floating point inches) of all the columns. Again this is needed in case the column we are on has to start a new table. Since the styles are not linked, all the information is needed for every column: that is, assuming you cannot guarantee perfectly structured data.

Using this approach it should be possible to typeset interlinear text with diglot free translation and intervening notes.

Other Options

Other options for `sh_rtf` are: `-d` allows the user to specify a document template that this document should attach to and update styles from on loading into Word. If you need to use SDF, then the `-p` option is used to specify the directory to find the `render.dll` in. This is an experimental feature and you will need the `Win32::API` module for anything to hope to work!

Interlinear Text

This section is a long and somewhat involved explanation of why `shintr` works the way it does. It is of only ancillary interest and may be skipped.

Interlinear text is fun stuff. The traditional approach to interlinearising is that the root of an interlinear block is a root word in the source language. This is then transformed into each of the forms on each line. The relationship between the transformations is such that the transformation path back to the root is such that a tree may be formed with no overlapping branches. If we compare the hierarchy of transformations along with the lines on which they occur, we find that we have a clean tree. There is the need for an agreement between the order in which interlinear lines are rendered and the transformational hierarchy of the interlineariser.

It is this relationshipe between hierarchies that a tool such as `shintr` has to address and resolve. Technologies such as EQ fields in Word, or XML, require data to be in a hierarchy. And if it is also necessary to keep line ordering, then the two hierarchies need to be resolved in some way. And for the most part, for normal interlinearising, rather than analysis and generation, such a model is perfectly sufficient (especially if care is taken over the order of the lines in the interlinear block).

Shoebox uses a different model. Rather than seeing each interlinear element as a node in a transformation tree, it sees each transformation as its own independent transformation from one line to another, without any relationship to any other transformation. This is a highly flexible approach allowing lines to be in any order.

Apart from the line ordering problem, the two models were easy to integrate for Shoebox versions 1-3 where only single word analysis was possible. With the advent of Shoebox 4, things became much more complicated due to the ability to do string to string mapping in addition to word to word mapping. The result is that we can no longer say that there is any root upon which a hierarchy may be built. The relationship between how the word in a line is generated becomes divorced from the generation of a word on another line from that word. For example, a string which is used as a single node in generating a subsequent line, may itself be made up of multiple nodes in its creation.

The tying of branches together, in this way, is another way in which the hierarchy is broken. Thus we have two ways in which the hierarchy can be broken and which such a tool as `shintr` must resolve. The first (overlapping branches) is resolved by, effectively, making a lower branch a child of the lowest common ancestor branch such that there is no overlap. This is not ideal, and some line-up errors may occur, but at least the data is there, and with some careful re-ordering of lines to be output, the problem can be resolved. The second problem (conflating branches by string replacement) is resolved by making the string replacement relate to the first node in the string in the hierarchy. Due to a technique of node conflation (nodes with no children and a common parent are conflated into a previous node with the same parent with children), the results are not as bad as might first be expected. But there are occasions, and the sample text is one of them, where the problem is visually evident.

These problems could be resolved if true tables were used where cell widths can vary and even cells be merged. The problem with a table based output approach is that it is necessary to measure the width of strings in order to find out how wide the table is for each block and to split it appropriately for line wrapping. On the other hand, EQ fields have the problem that Microsoft really don’t want to have to support them, and so they will probably get more buggy as time goes on. I’m sure the debate will run and run. In the meantime, I offer this humble tool to perhaps help a few people along the way.