Kika:Text Preproccesor

Pablo Marchant

October 15, 2006

Abstract

This document describes the kika software project first decent release, kika 0.1.

1 Introduction

Kika is meant to simplify tedious typing on various types of text files, by specifying certain rules on an xml file, wich are performed on a text file. It consists (for now at least) on a simple and small python script named *kika.py*. A first release has been made, with some changes on the basic idea that are due to certain problems on the implementation of the original one, and also as attempts to simplify it.

2 Calling a rule

To perform a rule on certain parts of a text file, a escape string must be defined, the string that separates the parameters must be defined and the string that indicates the end of the parameters must be defined. All this definitions are writen on the xml file. So, the general form of calling a rule looks like:

<escape String><Rule name><Parameter Separator><Param1><Parameter Separator>...<Parameter end><escape String>

So, for example, if I define a rule named hello that receives two parameters, specify the escape string to be #, specify parameters to be separated by a white space and the end of parameters specified by an end of line, the calling would look like:

```
#hello param1 param2
Any text can go here...
#
```

The text that goes between the "end of parameters string" and the "escape string" will be proccessed may be proccessed in an iteration, or simply printed out, depending on the form of the rule (How this is specified, is explained next)

3 Normal Rules

I call normal rules to those that receive only one set of parameters (i.e. they do not iterate over datasets). the basic xml for these would have the form:

```
<rule name="hello">
    <rb>
        <print>My name is </print> <param id="1"/><print>\n</print>
    </rb>
    <ra>
        <print>Goodbye</print> <param id="2"/><print>\n</print>
    </ra>
</rule>
```

The rule is divided in an "ra" node, and an "rb" node. Rb stands for "right before", and Ra for "right after". What goes on <rb> will be put before the text that isn't proccessed, and what goes on <ra> will be put after the text that is not proccessed.

A print node, prints what is inside of it literally, while a param node, prints the value of the specified parameter.

So, this rule would transform the text on the call made on the "Calling a rule" section for:

```
My name is param1
Any text can go here...
Goodbye param2
```

4 Iterarion Rules

An iteration rule is very similar to a normal one, except for the fact that it does not contain a section of unproccessed text. It's meant to iterate over datasets. For example, I could call a rule named "iter" with the text:

```
#iter param1 param2
1 2
12 32
1 43
#
```

The basic xml would be very similar, but the rule includes an "iterate" node:

```
<rule name="iter" itsep=" ">
    <rb>
        <print>My name is </print> <param id="1"/><print>\n</print>
    </rb>
    <iterate>
        <itparam id="1"> <symbol name="ampersand"/> <itparam id="2" /> <print>\n</print>
    </iterate>
    <ra>
        <print>Goodbye</print> <param id="2"/><print>\n</print>
    </ra>
</rule>
```

When a rule includes an iterate node, the itsep attribute specifies the string that separates the parameters on the dataset.

The "rb" and "ra" nodes have the same use than in normal rules, except that here, they enclose the result of the iteration. The "print" and "param" do exactly the same thing than before.

The "iterate" node specifies what to do in each iteration, and the "itparam" nodes refer to the parameters of each iteration.

The symbol node is used to print out certain symbols (specified by the "name") that cause conflict with minidom (the xml parser used to read the file). For the moment, the only symbol accepted is the ampersand (&).

The general parameters, those that are referred by the "param" nodes, are the ones that are given first after the name of the iteration rule. Every text that is between "end of parameters strings" is an iteration. So, the result of performing this iteration rule over the text would be:

```
My name is param1
1&2
12&32
1&43
Goodbye param2
```

5 Xml file

In the xml file, all rules and iteration rules are nested on a <rules> node. This node defines the strings that identify the rules. A complete file of xml rules would look like:

```
<rules es="#" ps=" " pe="\n">
<rule name="hello">
    <rb>
        <print>My name is </print> <param id="1"/><print>\n</print>
    </rb>
    <ra>
        <print>Goodbye</print> <param id="2"/><print>\n</print>
    </ra>
</rule>
<rule name="iter" itsep=" ">
    <rb>
        <print>My name is </print> <param id="1"/><print>\n</print>
    </rb>
    <iterate>
        <itparam id="1"> <symbol name="ampersand"/> <itparam id="2" /> <print>\n</print>
    </iterate>
    <ra>
        <print>Goodbye</print> <param id="2"/><print>\n</print>
    </ra>
</rule>
</rules>
```

6 Running kika

Kika can be run from the command line by calling it with python. It receives 2 arguments, the first being the name of the file that contains the xml rules, and the second, the text file to be proccessed. For example:

```
python kika.py xml2.rule ejemplo.tex > result.tex
```

This proccesses the file *ejemplo.tex* according to the rules defined in the file *xml2.rule*, and directs the output to the file *result.tex*

7 Conclusion

This is the description of the functionality of kika 0.1. Two test files have also been published on the projects home page (kika.sourceforge.net). These use kika to reduce the size of a latex file by defining rules that create the beggining of the document, figures, and tables.The script *kika.py* can be downloaded from the project page on sourceforge (sourceforge.net/projects/kika/)