Documentation and examples of sectioner.py 1.0
boxerer.py 1.0 and miarticle.cls

Manuel Gutierrez Algaba
irmina@ctv.es
http://www.ctv.es/USERS/irmina/texpython.htm

January 1999

1 Copyright issues

boxerer.py, sectioner.py, miarticle.cls and its documentation (this text and the
sources of tex drawings included in it) are copyrighted by Manuel Gutierrez
Algaba 1999 and you are free to use, modify, copy and distribute it under the
condition that you include this notice ad in it.

2 Introduction

This document explains all the details for the user of boxerer.py, sectioner.py
and miarticle.cls

2.1 boxerer.py

When I wrote boxerer.py there wasn’t any automated utility for doing input-
output-function boxes. Of course, you can write them directly in \LaTeX. But
boxerer.py has two major advantages, it’s easier to use and it’s faster to ‘write’.
Besides it can be used as an interface by CASE programs. The kinds of available
drawings are good for:

• Expressing the internal structure of large pieces of code, resembling their
inner and overall structure.

• Expressing the interfaces of modules or functions, that is, expressing mod-
ules and functions as black boxes.

And this is an utility written in python.

http://www.python.org

I imagine that it could be written in \TeX but it’s 3 times easier to use
python. And what’s more important \TeX programmers have a model, if they
want to do the translation.

Another point, the python code could be improved . . .

And, it generates tex code. So if you want it in postscript, gif or whatever,
use the programs dvi, gs or grab directly from a window!
2.2 sectioner.py and miarticle.cls

When I wrote sectioner.py there wasn’t any automated utility for handling relative sectioning in \LaTeX/\TeX, or I didn’t know anything about it. Of course, you can write traditional sections in \LaTeX. But sectioner.py has several major advantages:

- It’s easier to use because you don’t need to know if you are in a section, subsection, ..., you just go up or go down a level.
- It’s more abstract. You don’t need to take any decision about the class of the document (article, report, letter, miarticle, ...), this is done when you filter the tex source through sectioner.py. This, somehow, follows the true spirit of \TeX:

  To pay attention to logical design.

- It allows a greater structuration and modularity of \LaTeX\textcode, see details below 3.2.
- You are still compatible with \LaTeX, because you can flatten your sources of sectioner-\LaTeX\textcode into pure \LaTeX(using sectioner.py), and nobody would never know you used it.

And this is an utility written in python.

http://www.python.org

I imagine that it could be written in \TeX but it’s 8 times easier to use python. And what’s more important \TeX\text Código\textcode programmers have a model, if they want to do the translation.

Another point, the python code could be improved ...

3 How to use it

3.1 Using boxerer.py

Always use a “from boxerer import *” at the beginning of your python code.

3.1.1 Using the class ‘included’

This piece of python code illustrates the full capabilities of included, basically, you can put lot of structured stuff in it, this is very good, when you program has got very rich in terms of modularity and information. This code

\begin{verbatim}
r0 = ("men\_prin.py",None,"Here it’s the main menu is defined")
r1 = ("Graphic representation", [("tkinter.py",None,
    "Main calling module"),r0],
    "Graphic interfaces with the user, mainly")
r2 = ("Structures of the languages(grammar)", None)
r3 = ("Configuration", [("Database of words",None),
    ("Labels",None)])
i = included(("Lritaunas Peki",[r1,r2,r3],
\end{verbatim}
This program aims to teach vocabulary and grammar of different languages.

```
example2=inputoutputfunction('taxman',
  "This is your best friend, who helps you when you earn too much")
example2.do_inputs(['Your income','Your properties','The Law'])
example2.do_outputs(['Your taxes','Your fines',
  'Historical information'])
example2.do_functions(['Compute your taxes','Check your lies',
  'Send you to prison','Bribery'])
example2.generate_latex_code("result2.tex",200)
```

generates fig 1

When you don’t want to put its outputs (for example) then you got a figure as fig 2

Besides, if you put

```
example2=inputoutputfunction('taxman',
  "This is your best friend, who helps you when you earn too much",
  'es')
```
**Figure 1: Input-output-function figure**

**taxman**
This is your best friend, who helps you when you earn too much

**Inputs**
* Your income  
* Your properties  
* The Law

**Functions**
* Compute your taxes  
* Check your lies  
* Send you to prison  
* Bribery

**Outputs**
* Your taxes  
* Your fines  
* Historical information

---

**Figure 2: Input-output-function figure, without outputs**

**doctor**
This is your best friend, who helps you when you are tired of life

**Inputs**
* Your income  
* Your properties  
* Your confidence

**Functions**
* Kill you  
* Heal you

**Outputs**
None
then it generates the labels in Spanish.

### 3.2 Using sectioner.py

This program is a small revolution in the \LaTeX-world, I think. Let’s take a look at a sectioner-\LaTeX code:
\begin{verbatim}
\documentclass[10pt]{miarticle}
\begin{document}

\section*{Lritaunas Peki Documentation}
Lritaunas Peki is a programm for learning languages.

\section*{Structure of the Code}

\section*{Graphic user interfaces}

\section*{Ideas}

Basically, you must think about it as a nested layers of metawidgets. So we can reuse the code.

\section*{Python mega widgets}

This is a nice library of megawidgets, that supports scrollbars, multiple entries, dialogs...

\section*{My meta widgets.}

Trying to isolate GUI from tk and from Pmw(python megawidgets) I wrote a higher set of widgets.

\section*{Specialized metawidgets}

This widgets are those really used in the GUI, they’re composites of my meta widgets, and they make calls to the structures of the core code``\ref{lookupindictionaries}``

\end{verbatim}

Well, as you notice, \texttt{sections} have been replaced by \texttt{\textbackslash n,\+ or \textbackslash -. The line below a \texttt{\textbackslash n,\+ or \textbackslash -} is the title of the (sub)section or whatever. There’s no

5
binding with the style of the document, if you replace \texttt{miarticle} by \texttt{report}, then
the sectioning of \texttt{report} style will be used. The only thing you mustn’t forget
is that the title of it \texttt{CAN BE ONLY IN THE NEXT LINE TO THE \n,} or \texttt{\-}.

The \texttt{\+2} or \texttt{\-3} things say you go two levels up or down. And the \texttt{\n} stays
at the same level.

And the last thing is \texttt{/minput{strucincode.tex}}, this includes more code
in \texttt{sectioner-\LaTeX} style. As a matter of fact, all this story about \texttt{sectioner} is
that it’s better for modularity, look at this:
\verbatim
\n
Structures used:

\+ Dictionaries
\+ Functions
\+ Lookups
\texttt{\label{lookupindictionaries}}
This is probably one of the most used flavors.. blah, blah

\verbatim
\-3

As you may notice, \texttt{\-3} makes that all that module is encapsulated, in terms of
sectioning. Depending where you include it, you got sections, subsections,
or paragraphs, or whatever. And you can use labels, because, \texttt{sectioner} is just
a filter, when it finishes its work, you got pure \LaTeX, code. Of course you can
nest to the level you want (if your style allows it).

\subsection{How to call to \texttt{sectioner.py}}
\verbatim
python sectioner.py method inputfile outputfile

method can be: manolo (for \texttt{miarticle.cls}), article or report. \texttt{inputfile} would be
\texttt{highnest.tex} in this case, and \texttt{outputfile} whatever you want. As a example lets
take a look at the output (I wrote \texttt{o.tex} as \texttt{outputfile}):
Lritaunas Peki is a program for learning languages.

\section{Structure of the Code}

\subsection{Graphic user interfaces}
\subsubsection{Ideas}
Basically, you must think about it as a nested layers of metawidgets. So we can reuse the code.
\subsubsection{Python mega widgets}
This is a nice library of megawidgets, that supports scrollbars, multiple entries, dialogs...
\subsection{My meta widgets.}
Trying to isolate GUI from tk and from Pmw(python megawidgets) I wrote a higher set of widgets.
\subsection{Specialized metawidgets}
This widgets are those really used in the GUI, they're composites of my meta widgets, and they make calls to the structures of the core code\ref{lookupindictionaries}
\subsubsection{What you really see in your screen}
Blah, blah

\section{Structures used}
\subsection{Dictionaries}
\subsubsection{Functions}
\subsection{Lookups}
\label{lookupindictionaries}
This is probably one of the most used flavors.. blah, blah

\section{And yeah more blah}
Blah , and blah

3.2.2 About miarticle.cls

It's a good style when you require more than 7 levels of nesting. It has a small bug, it says some warning when compiling. Press Return twice and forget it.

3.3 Sources of help

Well, the best you can do is to take a look at the end of boxerer.py and sectioner.py where you can see how boxerer generates the draws and how sectioner filter. Secondly, you should take a look at the source of this document, that is: less textoolspro.tex
The figures are included using a simple `\input` command.

4 Caveats and bugs

Boxerer has no bugs. Sectioner has no bugs, miarticle has that nasty warning about that number (forget it). Even so, there’ll be some bugs.

5 Bye bye

I hope this documentation helps you to use these utilities. It’s not difficult and greatly profitable. And if you want to get similar drawings or improve some of them just take a look at the code, once you get accustomed to it, you’ll find it quite logical.