Code for rst2pdf pretty-printed by rst2pdf

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import docutils.core,docutils.nodes,sys,re
import pygments_code_block_directive
import sys
import pprint
from types import StringType
from docutils import __version__, __version_details__, SettingsSpec
from docutils import frontend, io, utils, readers, writers
from docutils.frontend import OptionParser
from docutils.transforms import Transformer
import docutils.readers.doctree

from urlparse import *

from reportlab.platypus import *
#from reportlab.platypus.para import Paragraph,FastPara,Para
from reportlab.pdfbase.pdfmetrics import stringWidth
from reportlab.lib.enums import *
import reportlab.lib.colors as colors
from reportlab.lib.units import *
from reportlab.lib.pagesizes import *
from copy import copy
from cgi import escape

def _log(msg):
 sys.stderr.write('%s\n'%str(msg))
 sys.stderr.flush()

try:
 from wordaxe.rl.paragraph import Paragraph
 from wordaxe.rl.styles import ParagraphStyle, getSampleStyleSheet
except:
 _log("No support for hyphenation, install wordaxe")

from styles import *
styles=None

def styleToFont(style):
 '''Takes a style name, returns a font tag for it, like
  "<font face=helvetica size=14 color=red>". Used for inline
  nodes (custom interpreted roles)'''

 try:
  s=styles[style]
  return '<font face="%s" size="%d" color="%s">'%(s.fontName,s.fontSize,s.textColor)
 except KeyError:
  _log('Unknown class %s'%style)
  return None

try:
 import wordaxe
 from wordaxe.PyHnjHyphenator import PyHnjHyphenator
 try:
  wordaxe.hyphRegistry['EN'] = PyHnjHyphenator('en_US',5)
 except:
  wordaxe.hyphRegistry['EN'] = PyHnjHyphenator('en_US',5,purePython=True)
except:
 _log("No support for hyphenation, install wordaxe")


# This is A4 paper, change it as you wish
pw,ph=ps=A4

# Margins
```

```
tm=2*cm
bm=2*cm
lm=3*cm
rm=1.5*cm

# tw is the text width.
# We need it to calculate header-footer height
# and compress literal blocks.
tw=pw-lm-rm

page_break_level=0

marks="#=-_*^>%&|"
lowerroman=['i','ii','iii','iv','v','vi','vii','viii','ix','x','xi']
loweralpha="abcdefghijklmnopqrstuvwxyz"

class MyIndenter(Indenter):
 # Bugs in reportlab?
 def draw(self):
  pass
 width=0
 height=0

def depth(node):
  if node.parent==None:
    return 0
  else:
    return 1+depth(node.parent)

decoration={'header':None, 'footer':None, 'endnotes':[]}

def gather_pdftext(node, depth, in_line_block=False,replaceEnt=True):
 return ''.join([gen_pdftext(n,depth,in_line_block,replaceEnt) for n in node.children])

def gen_pdftext(node, depth, in_line_block=False,replaceEnt=True):

 pre=""
 post=""

 if isinstance(node, docutils.nodes.paragraph)\
  or isinstance(node, docutils.nodes.title)\
  or isinstance(node, docutils.nodes.subtitle)\
  :
  node.pdftext=gather_pdftext(node,depth)+"\n"

 elif isinstance(node, docutils.nodes.Text):
  node.pdftext=node.astext()
  if replaceEnt:
   node.pdftext=escape(node.pdftext,True)
  node.pdftext=pre+node.pdftext+post

 elif isinstance(node, docutils.nodes.strong):
  pre="<b>"
  post="</b>"
  node.pdftext=gather_pdftext(node,depth)
  if replaceEnt:
   node.pdftext=escape(node.pdftext,True)
  node.pdftext=pre+node.pdftext+post

 elif isinstance(node, docutils.nodes.emphasis):
  pre="<i>"
  post="</i>"
  node.pdftext=gather_pdftext(node,depth)
  if replaceEnt:
   node.pdftext=escape(node.pdftext,True)
  node.pdftext=pre+node.pdftext+post

 elif isinstance(node, docutils.nodes.literal):
  pre='<font face="%s">'%styles['code'].fontName
  post="</font>"
```

```
 node.pdftext=gather_pdftext(node,depth)
 if replaceEnt:
  node.pdftext=escape(node.pdftext,True)
 node.pdftext=pre+node.pdftext+post

elif isinstance(node, docutils.nodes.superscript):
 pre='<super>'
 post="</super>"
 node.pdftext=gather_pdftext(node,depth)
 if replaceEnt:
  node.pdftext=escape(node.pdftext,True)
 node.pdftext=pre+node.pdftext+post

elif isinstance(node, docutils.nodes.subscript):
 pre='<sub>'
 post="</sub>"
 node.pdftext=gather_pdftext(node,depth)
 if replaceEnt:
  node.pdftext=escape(node.pdftext,True)
 node.pdftext=pre+node.pdftext+post

elif isinstance(node, docutils.nodes.title_reference):
 # FIXME needs to work as a link
 pre='<font color="blue">'
 post="</font>"
 node.pdftext=gather_pdftext(node,depth)
 if replaceEnt:
  node.pdftext=escape(node.pdftext,True)
 node.pdftext=pre+node.pdftext+post

elif isinstance(node, docutils.nodes.reference) :
 pre='<font color="blue">'
 post="</font>"
 uri=node.get('refuri')
 if uri:
  if urlparse(uri)[0]:
   pre+=u'<a href="%s">'%uri
   post='</a>'+post
 else:
   uri=node.get('refid')
   if uri:
     pre+=u'<a href="#%s">'%uri
     post='</a>'+post
 node.pdftext=node.astext()
 if replaceEnt:
  node.pdftext=escape(node.pdftext,True)
 node.pdftext=pre+node.pdftext+post

elif  isinstance(node, docutils.nodes.option_string) \
  or isinstance(node, docutils.nodes.option_argument)\
  :
 node.pdftext=node.astext()
 if replaceEnt:
  node.pdftext=escape(node.pdftext,True)

elif  isinstance(node, docutils.nodes.header)\
  or isinstance(node, docutils.nodes.footer)\
    :
 node.pdftext=gather_pdftext(node,depth)
 if replaceEnt:
  node.pdftext=escape(node.pdftext,True)

 node.pdftext=pre+node.pdftext+post

elif  isinstance(node, docutils.nodes.system_message) \
  or isinstance(node, docutils.nodes.problematic) \
  :
 sys.stderr.write(node.astext()+"\n")
 sys.stderr.flush()
 pre='<font color="red">'
```

```
 post="</font>"
 node.pdftext=gather_pdftext(node,depth)
 if replaceEnt:
  node.pdftext=escape(node.pdftext,True)
 node.pdftext=pre+node.pdftext+post

elif isinstance(node,docutils.nodes.generated):
 node.pdftext=gather_pdftext(node,depth)
 if replaceEnt:
  node.pdftext=escape(node.pdftext,True)
 node.pdftext=pre+node.pdftext+post

elif isinstance(node,docutils.nodes.image):
 node.pdftext='<img src="%s" />'%node.get('uri')

elif isinstance(node,docutils.nodes.footnote_reference):
 # Fixme link to the right place
 node.pdftext=u'<super><font color="blue">%s</font></super>'%node.astext()
elif isinstance(node,docutils.nodes.citation_reference):
 # Fixme link to the right place
 node.pdftext=u'<font color="blue">[%s]</font>'%node.astext()

# FIXME nodes we are ignoring for the moment
elif isinstance(node,docutils.nodes.target):
 # FIXME: make it work as a target for links
 node.pdftext=gather_pdftext(node,depth)
 if replaceEnt:
  node.pdftext=escape(node.pdftext,True)

elif isinstance(node,docutils.nodes.inline):
 ftag=styleToFont(node['classes'][0])
 if ftag:
  node.pdftext="%s%s</font>"%(ftag,gather_pdftext(node,depth))
 else:
  node.pdftext=gather_pdftext(node,depth)

else:
 _log("Unkn. node (gen_pdftext): %s"%str(node.__class__))
 _log(node)
 #print node.transform
 sys.exit(1)

return node.pdftext



def PreformattedFit(text,style):
 """Preformatted section that gets horizontally compressed if needed."""
 print "PFFit:",text
 w=max(map(lambda line:stringWidth(line,style.fontName,style.fontSize),text.splitlines()))
 mw=tw-style.leftIndent-style.rightIndent
 if w>mw:
  style=copy(style)
  f=max((0.375,mw/w))
  style.fontSize*=f
  style.leading*=f
 return XPreformatted(text,style)


def gen_elements(node, depth, in_line_block=False, style=None):
 print node
 if style is None:
  style=styles['bodytext']

 if node['classes']:
  # Supports only one class, sorry ;-)
  try:
   style=styles[node['classes'][0]]
  except:
   _log("Unknown class %s, using class bodytext."%node['classes'][0])
```

```
global decoration

if isinstance(node, docutils.nodes.document):
 node.elements=gather_elements(node,depth,style=style)


########################
## Tables
########################

elif isinstance(node, docutils.nodes.table):
   node.elements=gather_elements(node,depth)

elif isinstance(node, docutils.nodes.tgroup):
 rows=[]
 hasHead=False
 for n in node.children:
  if isinstance(n,docutils.nodes.thead):
   hasHead=True
   for row in n.children:
    r=[]
    for cell in row.children:
     r.append(cell)
    rows.append(r)
  elif isinstance(n,docutils.nodes.tbody):
   for row in n.children:
    r=[]
    for cell in row.children:
     r.append(cell)
    rows.append(r)

 spans=filltable(rows)

 data=[]

 for row in rows:
  r=[]
  for cell in row:
   if isinstance(cell,str):
    r.append("")
   else:
    r.append(gather_elements(cell,depth))
  data.append(r)

 st=spans+tstyleNorm

 if hasHead:
  st+=[tstyleHead]

 node.elements=[Table(data,style=TableStyle(st))]

elif isinstance(node, docutils.nodes.title):
 # Special cases: (Not sure this is right ;-)
 if isinstance(node.parent, docutils.nodes.document):
  # FIXME maybe make it a coverpage?
  node.elements=[Paragraph(gen_pdftext(node,depth), styles['title'])]
 elif isinstance(node.parent, docutils.nodes.topic):
  # FIXME style correctly
  node.elements=[Paragraph(gen_pdftext(node,depth), styles['heading3'])]
 elif isinstance(node.parent, docutils.nodes.admonition) or
   isinstance(node.parent, docutils.nodes.sidebar):
  node.elements=[Paragraph(gen_pdftext(node,depth), styles['heading3'])]
 else:
  node.elements=[Paragraph(gen_pdftext(node,depth), styles['heading%d'%min(depth,3)])]


elif isinstance(node, docutils.nodes.subtitle):
 if isinstance(node.parent,docutils.nodes.sidebar):
```

```
  node.elements=[Paragraph(gen_pdftext(node,depth), styles['heading4'])]
 elif isinstance(node.parent,docutils.nodes.document):
  node.elements=[Paragraph(gen_pdftext(node,depth), styles['subtitle'])]

elif isinstance(node, docutils.nodes.paragraph):
 node.elements=[Paragraph(gen_pdftext(node,depth), style)]

elif isinstance(node, docutils.nodes.docinfo):
 # A docinfo usually contains several fields.
 # We'll render it as a series of elements, one field each.

 node.elements=gather_elements(node,depth,style=style)

elif isinstance(node, docutils.nodes.field):
 # A field has two child elements, a field_name and a field_body.
 # We render as a two-column table, left-column is right-aligned,
 # bold, and much smaller

 fn=Paragraph(gather_pdftext(node.children[0],depth)+":",style=styles['fieldname'])
 fb=gen_elements(node.children[1],depth)
 node.elements=[Table([[fn,fb]],style=tstyles['field'],colWidths=[fieldlist_lwidth,None])]

elif isinstance(node, docutils.nodes.decoration):
 # This is a tricky one. We need to switch our document's
 # page templates based on this. If decoration contains a
 # header and/or a footer, we need to use those
 # right now, we avoid trouble.
 # FIXME Implement
 node.elements=gather_elements(node,depth,style=style)

elif isinstance(node, docutils.nodes.header):
 decoration['header']=gather_pdftext(node,depth)
 node.elements=[]
elif isinstance(node, docutils.nodes.footer):
 decoration['footer']=gather_pdftext(node,depth)
 node.elements=[]

elif isinstance(node, docutils.nodes.author):
 if isinstance(node.parent,docutils.nodes.authors):
  # Is only one of multiple authors. Return a paragraph
  node.elements=[Paragraph(gather_pdftext(node,depth), style=style)]

 else:
  # A single author: works like a field
  fb=gather_pdftext(node,depth)
  node.elements=[Table([[Paragraph("Author:",style=styles['fieldname']),
      Paragraph(fb,style)]],style=tstyles['field'],colWidths=[fieldlist_lwidth,None])]

elif isinstance(node, docutils.nodes.authors):
 # Multiple authors. Create a two-column table. Author references on the right.
 td=[[Paragraph("Authors:",style=styles['fieldname']),gather_elements(node,depth,style=style)]]
 node.elements=[Table(td,style=tstyles['field'],colWidths=[fieldlist_lwidth,None])]

elif isinstance(node, docutils.nodes.organization):
 fb=gather_pdftext(node,depth)
 t=Table([[Paragraph("Organization:",style=styles['fieldname']),
     Paragraph(fb,style)]],style=tstyles['field'],colWidths=[fieldlist_lwidth,None])
 node.elements=[t]
elif isinstance(node, docutils.nodes.contact):
 fb=gather_pdftext(node,depth)
 t=Table([[ Paragraph("Contact:",style=styles['fieldname']),
     Paragraph(fb,style)]],style=tstyles['field'],colWidths=[fieldlist_lwidth,None])
 node.elements=[t]
elif isinstance(node, docutils.nodes.address):
 fb=gather_pdftext(node,depth)
 t=Table([[ Paragraph("Address:",style=styles['fieldname']),
     Paragraph(fb,style)]],style=tstyles['field'],colWidths=[fieldlist_lwidth,None])
 node.elements=[t]
elif isinstance(node, docutils.nodes.version):
 fb=gather_pdftext(node,depth)
```

```
 t=Table([[ Paragraph("Version:",style=styles['fieldname']),
     Paragraph(fb,style) ]],style=tstyles['field'],colWidths=[fieldlist_lwidth,None])
 node.elements=[t]
elif isinstance (node, docutils.nodes.revision):
 fb=gather_pdftext(node,depth)
 t=Table([[ Paragraph("Revision:",style=styles['fieldname']),
     Paragraph(fb,style) ]],style=tstyles['field'],colWidths=[fieldlist_lwidth,None])
 node.elements=[t]
elif isinstance (node, docutils.nodes.status):
 fb=gather_pdftext(node,depth)
 t=Table([[ Paragraph("Version:",style=styles['fieldname']),
     Paragraph(fb,style) ]],style=tstyles['field'],colWidths=[fieldlist_lwidth,None])
 node.elements=[t]
elif isinstance (node, docutils.nodes.date):
 fb=gather_pdftext(node,depth)
 t=Table([[ Paragraph("Date:",style=styles['fieldname']),
     Paragraph(fb,style) ]],style=tstyles['field'],colWidths=[fieldlist_lwidth,None])
 node.elements=[t]
elif isinstance (node, docutils.nodes.copyright):
 fb=gather_pdftext(node,depth)
 t=Table([[Paragraph("Copyright:",style=styles['fieldname']),
     Paragraph(fb,style) ]],style=tstyles['field'],colWidths=[fieldlist_lwidth,None])
 node.elements=[t]

elif  isinstance (node, docutils.nodes.topic)       \
  or isinstance (node, docutils.nodes.field_body)     \
  :
 node.elements=gather_elements(node,depth,style=style)

elif isinstance (node, docutils.nodes.section):
 if depth<page_break_level:
  node.elements=[PageBreak()]+gather_elements(node,depth+1)
 else:
  node.elements=gather_elements(node,depth+1)

elif  isinstance (node, docutils.nodes.bullet_list)     \
  or isinstance (node, docutils.nodes.enumerated_list)   \
  or isinstance (node, docutils.nodes.definition_list)   \
  or isinstance (node, docutils.nodes.option_list)     \
  or isinstance (node, docutils.nodes.field_list)     \
  or isinstance (node, docutils.nodes.definition)     \
  :

 node.elements=gather_elements(node,depth,style=style)

elif isinstance (node, docutils.nodes.option_list_item):
 og  = gather_elements(node.children[0],depth,style)
 desc = gather_elements(node.children[1],depth,style)

 node.elements=[Table([[og,desc]],style=tstyles['field'])]


elif isinstance (node, docutils.nodes.definition_list_item):

 # I need to catch the classifiers here
 tt=[]
 dt=[]
 for n in node.children:
  if isinstance(n,docutils.nodes.term) or\
   isinstance(n,docutils.nodes.classifier) :
   tt.append(gather_pdftext(n,depth,style))
  else:
   dt=dt+gen_elements(n,depth,style)

 node.elements=[Paragraph(':'.join(tt),style),MyIndenter(left=10)]+dt+[MyIndenter(left=-10)]

elif isinstance (node, docutils.nodes.list_item):
 # A list_item is a table of two columns.
 # The left one is the bullet itself, the right is the
 # item content. This way we can nest them.
```

```
 el=gather_elements(node,depth,style=style)
 b=""

 if node.parent.get('bullet') or isinstance(node.parent,docutils.nodes.bullet_list):
  # FIXME: use correct bullet symbols, check inter-paragraph spacing
  b=str(node.parent.get('bullet'))
  if b=="None":
   b=""

 elif node.parent.get('enumtype')=='arabic':
  b=str(node.parent.children.index(node)+1)+'.'

 elif node.parent.get('enumtype')=='lowerroman':
  b=str(lowerroman[node.parent.children.index(node)])+'.'

 elif node.parent.get('enumtype')=='upperroman':
  b=str(lowerroman[node.parent.children.index(node)].upper())+'.'

 elif node.parent.get('enumtype')=='loweralpha':
  b=str(loweralpha[node.parent.children.index(node)])+'.'
 elif node.parent.get('enumtype')=='upperalpha':
  b=str(loweralpha[node.parent.children.index(node)].upper())+'.'
 else:
  _log("Unknown kind of list_item")
  _log(node.parent)
  sys.exit(1)
 el[0].bulletText = b
 node.elements=el

elif isinstance(node, docutils.nodes.transition):
 node.elements=[Separation()]


elif  isinstance(node, docutils.nodes.system_message) \
  or isinstance(node, docutils.nodes.problematic) \
  :
 # FIXME show the error in the document, red, whatever
 sys.stderr.write(node.astext()+"\n")
 sys.stderr.flush()
 node.elements=[]

elif isinstance(node, docutils.nodes.block_quote):
 node.elements=[MyIndenter(left=20)]+gather_elements(node,depth,style)+[MyIndenter(left=-20)]

elif isinstance(node, docutils.nodes.attribution):
 node.elements=[Paragraph(gather_pdftext(node,depth),styles['attribution'])]

elif isinstance(node, docutils.nodes.comment):
 # Class that generates no output
 node.elements=[]

elif isinstance(node, docutils.nodes.line_block):
 # Obsolete? Let's do something anyway.
 # FIXME: indent or not?
 qstyle=copy(style)
 qstyle.leftIndent+=30
 node.elements=gather_elements(node,depth,style=qstyle)

elif isinstance(node, docutils.nodes.line):
 # All elements in one line
 node.elements=[Paragraph(gather_pdftext(node,depth),style=style)]

elif  isinstance(node, docutils.nodes.literal_block)\
  or isinstance(node, docutils.nodes.doctest_block)\
  or isinstance(node, docutils.nodes.option)\
 :
 node.elements=[PreformattedFit(gather_pdftext(node,depth,replaceEnt=False),styles['code'])]

elif  isinstance(node, docutils.nodes.attention) \
```

```
  or isinstance(node, docutils.nodes.caution)   \
  or isinstance(node, docutils.nodes.danger)    \
  or isinstance(node, docutils.nodes.error)     \
  or isinstance(node, docutils.nodes.hint)      \
  or isinstance(node, docutils.nodes.important) \
  or isinstance(node, docutils.nodes.note)      \
  or isinstance(node, docutils.nodes.tip)       \
  or isinstance(node, docutils.nodes.warning)   \
  or isinstance(node, docutils.nodes.admonition) \
:
 node.elements=[Paragraph(node.tagname.title(),style=styles['heading3'])]+gather_elements(node,depth,s

elif isinstance(node, docutils.nodes.image):
 # FIXME handle all the other attributes
 i=Image(filename=str(node.get("uri")))
 if node.get('align'):
  i.hAlign=node.get('align').upper()

 node.elements=[i]
elif isinstance(node, docutils.nodes.figure):
 # The sub-elements are the figure and the caption, and't ugly if
 # they separate
 node.elements=[KeepTogether(gather_elements(node,depth,style=style))]

elif isinstance(node, docutils.nodes.caption):
 node.elements=[Paragraph('<i>'+gather_pdftext(node,depth)+'</i>',style=style)]

elif isinstance(node, docutils.nodes.legend):
 node.elements=gather_elements(node,depth,style=style)

elif isinstance(node, docutils.nodes.sidebar):
 node.elements=[Table([[ gather_elements(node,depth,style=style)]],style=tstyles['sidebar'])]

elif isinstance(node, docutils.nodes.rubric):
 node.elements=[Paragraph(gather_pdftext(node,depth),styles['rubric'])]

elif isinstance(node, docutils.nodes.compound):
 # FIXME think if this is even implementable
 node.elements=gather_elements(node,depth,style)

elif isinstance(node, docutils.nodes.container):
 # FIXME think if this is even implementable
 node.elements=gather_elements(node,depth,style)

elif isinstance(node, docutils.nodes.substitution_definition):
 node.elements=[]

elif isinstance(node, docutils.nodes.tbody):
 rows=[gen_elements(n,depth) for n in node.children]
 t=[]
 for r in rows:
  if not r:
   continue
  t.append(r)
 node.elements=[Table(t,style=tstyles['normal'])]

elif isinstance(node, docutils.nodes.footnote):
 # It seems a footnote contains a label and a series of elements

 label=Paragraph(gather_pdftext(node.children[0],depth),style)
 contents=gather_elements(node,depth,style)[1:]
 decoration['endnotes'].append([label,contents])
 node.elements=[]

elif isinstance(node, docutils.nodes.label):
 node.elements=[Paragraph(gather_pdftext(node,depth),style)]
elif isinstance(node, docutils.nodes.Text):
 node.elements=[Paragraph(gather_pdftext(node,depth),style)]
elif isinstance(node, docutils.nodes.entry):
 node.elements=gather_elements(node,depth,style)
```

```
 # FIXME nodes we are ignoring for the moment
 elif  isinstance(node, docutils.nodes.target)      \
   or isinstance(node, docutils.nodes.footnote) \
   or isinstance(node, docutils.nodes.citation) \
   or isinstance(node, docutils.nodes.reference) \
   or isinstance(node, docutils.nodes.raw) \
  :
  node.elements=[]
 else:
  _log("Unkn. node (gen_elements): %s"%str(node.__class__))
  _log(node)
  sys.exit(1)

 # set anchors for internal references
 for id in node['ids']:
  node.elements.insert(
    node.elements and isinstance(node.elements[0], PageBreak) and 1 or 0,
    Paragraph('<a name="%s"/>'%id,style))

 return node.elements

def gather_elements(node, depth, in_line_block=False,style=None):
 if style is None:
  style=styles['bodytext']
 r=[]
 for n in node.children:
  r=r+(gen_elements(n,depth,in_line_block,style=style))
 return r

class Separation(Flowable):
 " A simple <hr>-like thingie"

 def wrap(self,w,h):
  self.w=w
  return (w,1*cm)

 def draw(self):
  self.canv.line(0,0.5*cm,self.w,0.5*cm)


class FancyPage(PageTemplate):
 def __init__(self,_id,pw,ph,tm,bm,lm,rm,head,foot):


  tw=pw-lm-rm

  if head:
   hh=Paragraph(head,style=styles['header']).wrap(tw,ph)[1]
  else:
   hh=0
  if foot:
   fh=Paragraph(foot,style=styles['footer']).wrap(tw,ph)[1]
  else:
   fh=0

  #textframe=Frame(lm,tm+hh,tw,ph-tm-bm-hh-fh)
  textframe=Frame(lm,tm+hh,tw,ph-tm-bm-hh-fh,topPadding=hh,bottomPadding=fh)

  self.head=head
  self.hx=lm
  self.hy=ph-tm

  self.foot=foot
  self.fx=lm
  self.fy=bm
  self.tw=tw
  self.ph=ph

  PageTemplate.__init__(self,_id,[textframe])
```

```
 def beforeDrawPage(self,canv,doc):

  # Replace ###Page### with the actual page number


  if self.head:
   para=Paragraph(self.head.replace('###Page###',str(doc.page)),style=styles['header'])
   para.wrap(self.tw,self.ph)
   para.drawOn(canv,self.hx,self.hy)
  if self.foot:
   para=Paragraph(self.foot.replace('###Page###',str(doc.page)),style=styles['footer'])
   para.wrap(self.tw,self.ph)
   para.drawOn(canv,self.fx,self.fy)

def filltable(rows):

 # If there is a multicol cell, we need to insert Continuation Cells
 # to make all rows the same length


 for y in range(0,len(rows)):
  for x in range(0,len(rows[y])):
   cell=rows[y][x]
   if isinstance(cell,str):
    continue
   if cell.get("morecols"):
    for i in range(0,cell.get("morecols")):
     rows[y].insert(x+1,"")

 for y in range(0,len(rows)):
  for x in range(0,len(rows[y])):
   cell=rows[y][x]
   if isinstance(cell,str):
    continue
   if cell.get("morerows"):
    for i in range(0,cell.get("morerows")):
     rows[y+i+1].insert(x,"")

 # Create spans list for reportlab's table style
 spans=[]
 for y in range(0,len(rows)):
  for x in range(0,len(rows[y])):
   cell=rows[y][x]
   if isinstance(cell,str):
    continue
   if cell.get("morecols"):
    mc=cell.get("morecols")
   else: mc=0
   if cell.get("morerows"):
    mr=cell.get("morerows")
   else: mr=0

   if mc or mr:
    spans.append(('SPAN',(x,y),(x+mc,y+mr)))
 return spans

from optparse import OptionParser

def main():
 global styles
 parser = OptionParser()
 parser.add_option('-o', '--output',dest='output',help='Write the PDF to FILE',metavar='FILE')
 parser.add_option('-s', '--stylesheet',dest='style',help='Custom stylesheet',metavar='STYLESHEET')
 parser.add_option('--print-stylesheet',dest='printssheet',action="store_true", default=False,help='Pri
 (options,args)=parser.parse_args()

 if options.printssheet:
  print open(os.path.join(os.path.abspath(os.path.dirname(__file__)), 'styles.json')).read()
  sys.exit(0)
```

```
 if len(args) <> 1:
  _log('Usage: %s file.txt [ -o file.pdf ]'%sys.argv[0])
  sys.exit(1)


 infile=args[0]
 if options.output:
  outfile=options.output
 else:
  outfile=infile+'.pdf'

 if options.style:
  styles=getStyleSheet(options.style)
 else:
  styles=getStyleSheet(os.path.join(os.path.abspath(os.path.dirname(__file__)),'styles.json'))


 input=open(infile).read()
 import docutils.core
 doc=docutils.core.publish_doctree(input)
 elements=gen_elements(doc,0)


 # Put the endnotes at the end ;-)
 endnotes = decoration['endnotes']
 if endnotes:
   elements.append(Spacer(1,2*cm))
   elements.append(Separation())
   for n in decoration['endnotes']:
    elements.append(Table([[n[0],n[1]]],style=tstyles['endnote'],colWidths=[endnote_lwidth,None]))

 head=decoration['header']
 foot=decoration['footer']

 # So, now, create the FancyPage with the right sizes and elements
 FP=FancyPage("fancypage",pw,ph,tm,bm,lm,rm,head,foot)

 pdfdoc = BaseDocTemplate(outfile,pageTemplates=[FP],showBoundary=0,pagesize=ps)
 pdfdoc.build(elements)

if __name__ == "__main__":
 main()
```