```python
#!/usr/bin/env python
R"""Xmlt merges layout and text

Usage: xmlt [-n ns.py..] dtml-file xmlt-files.. html-file

Xmlt (Xmlt Merges Layout and Text) uses the DocumentTemplate module from Zope [1] to merge a layout
(described in a dtml-file) and the text (found in xmlt-files). The generated HTML document is written to the
html-file. Optionally you can specify one or more Python name space files containing attributes and functions
used in the dtml-file (or in the xmlt-files).

(C) Copyright Anders Andersen 1999-2002 (see COPYING for details).

[1] http://www.zope.org/

"""

# Inserted by rcs (cvs)
rcs_author = "$Author: aa $"
rcs_date = "$Date: 2002/04/11 12:42:07 $"
rcs_revision = "$Revision: 1.23 $"

# We need sys to access arguments
import sys, os, getopt

# We need regular expression to find attributes in the document file
import string, re

# We are using DocumentTemplate from Zope (http://www.zope.org/)
import DocumentTemplate

# Parse arguments
opts, args = getopt.getopt(sys.argv[1:], "n:")

# We need at least three arguments
if len(args) < 3:
    print "xmlt: Need at least three arguments.\n"
    print __doc__,
    sys.exit(1)

# We can read two types of configuration variables; flags and attributes.
flag_expr = re.compile(r"<!--\s*([a-zA-Z]\w*)\s*-->")
attr_expr = re.compile(r"<!--\s*([a-zA-Z]\w*):\s*(.*\S)\s*-->")

# The title and body
tag_string = r"<%s>\s*(.*?)\s*</%s>"
title_expr = re.compile(tag_string % (("title",) * 2), re.MULTILINE | re.DOTALL)
#body_expr = re.compile(tag_string % (("body",) * 2), re.MULTILINE | re.DOTALL)
body_begin_expr = re.compile("<body>")
body_end_expr = re.compile("</body>")

# Flags and attributes are saved in the doc instance
class Dummy: pass
doc = Dummy()

# Add xmlt info
rcs_expr = re.compile(r"\$\w+:\s+(.*)\s+\$$")
doc.xmlt_version = rcs_expr.match(rcs_revision).group(1)
doc.xmlt_date = rcs_expr.match(rcs_date).group(1)
doc.xmlt_msg = '''This document has been created using
<a href="ftp://starship.python.net/pub/crew/anders/xmlt/">xmlt</a>
version %s (%s), &copy; Anders Andersen 1999-2002''' % (
```

```
    doc.xmlt_version, doc.xmlt_date)

# Any external namespaces (-n ns.py):
for (nflag, nfile) in opts:
    g = {}
    try:
        execfile(nfile, g, doc.__dict__)
    except IOError:
        print "xmlt: Unable to read '%s' (name space file).\n" % (nfile,)
        print __doc__,
        sys.exit(1)
    except:
        print "xmlt: Problem executing '%s' (name space file).\n" % (nfile,)
        print __doc__,
        sys.exit(1)

# Default values (tags and toc not initialised)

# Name of current document
doc.current = os.path.basename(args[-1])

# User defined value tags (use only one of each in an xmlt file)
vtag_expr = re.compile(r"<!--\s*[Vv]tags:\s*(.*\S)\s*-->")
vtag_exprs = {}

# User defined tags (with a defined translation: in group (2) and (3) of expr)
tag_expr = re.compile(
    r"<!--\s*[Tt]ag:\s*(\w*)\s*:(.*?)::(.*?)\s*-->",
    re.MULTILINE | re.DOTALL)
tag_exprs = {}
tag_impl = {}

# The modified expression (comment) is also a part of xmlt (originally from
# html-helper mode in emacs).
modified_expr = re.compile(
    r"<!-- hhmts start -->\s*" +
    "Last [Mm]odified: (.*)\s*" +
    "<!-- hhmts end -->", re.MULTILINE)

# RCS (CVS) support (use either rcs or the modified expression above)
rcs_expr = re.compile(r'\$(\w*):\s*(.*\S)\s*\$')

# This is used to generate a table of contents
h_expr = re.compile('^\s*<[Hh]([2-6])>(.*)</[Hh][2-6]>\s*$')
hb_expr = re.compile('^(\s*<[Hh][2-6]>)(.*)(</[Hh][2-6]>\s*)$')
doc.clevel = 4

# Strip surounding dblquotes
def stripquotes(s):
    if s[0] == '"' and s[-1] == '"':
        return s[1:-1]
    else:
        return s

# Create toc information and modify body (insert named anchors in body)
def mkToc(body):
    from StringIO import StringIO
    bdf = StringIO(body)
```

```
    level = 2; toc = {}; body=""
    name = [""] * (doc.clevel-1)
    line = bdf.readline()
    while line:
        if h_expr.match(line):
            folder = h_expr.match(line).groups()
            tlevel = string.atoi(folder[0])
            if tlevel <= doc.clevel:
                name[tlevel-2] = folder[1]
                folder = hb_expr.match(line).groups()
                body = body + folder[0] + '<A NAME="'
                for i in range(tlevel-1):
                    body = body + name[i]
                    if i < tlevel-2: body = body + ' '
                body = body + '">%s</A>%s' % (folder[1], folder[2])
                if tlevel == level:
                    toc = toc + '%s<LI><A HREF="#' % ('  ' * (tlevel-1),)
                    for i in range(tlevel-1):
                        toc = toc + name[i]
                        if i < tlevel-2: toc = toc + ' '
                    toc = toc + '">%s</A>\n' % (name[tlevel-2],)
                elif tlevel > level:
                    for i in range(level-2, tlevel-2):
                        toc = toc + '%s<UL>\n' % ('  ' * (i+1),)
                    toc = toc + '%s<LI><A HREF="#' % ('  ' * (tlevel-1),)
                    for i in range(tlevel-1):
                        toc = toc + name[i]
                        if i < tlevel-2: toc = toc + ' '
                    toc = toc + '">%s</A>\n' % (name[tlevel-2],)
                    level = tlevel
                elif tlevel < level:
                    for i in range(level-2, tlevel-2, -1):
                        toc = toc + '%s</UL>\n' % ('  ' * i,)
                    toc = toc + '%s<LI><A HREF="#' % ('  ' * (tlevel-1),)
                    for i in range(tlevel-1):
                        toc = toc + name[i]
                        if i < tlevel-2: toc = toc + ' '
                    toc = toc + '">%s</A>\n' % (name[tlevel-2],)
                    level = tlevel
            else:
                body = body + line
        else:
            body = body + line
        line = bdf.readline()
    bdf.close()
    if toc:
        toc = "<UL>\n" + toc + "</UL>\n"
    return (toc, body)

# Create toc and modify body (replaced by mkToc above soon)
def makeToc(body):
    from StringIO import StringIO
    bdf = StringIO(body)
    level = 2; toc = ""; body=""
    name = [""] * (doc.clevel-1)
    line = bdf.readline()
    while line:
        if h_expr.match(line):
```

```
            folder = h_expr.match(line).groups()
            tlevel = string.atoi(folder[0])
            if tlevel <= doc.clevel:
                name[tlevel-2] = folder[1]
                folder = hb_expr.match(line).groups()
                body = body + folder[0] + '<A NAME="'
                for i in range(tlevel-1):
                    body = body + name[i]
                    if i < tlevel-2: body = body + ' '
                body = body + '">%s</A>%s' % (folder[1], folder[2])
                if tlevel == level:
                    toc = toc + '%s<LI><A HREF="#' % ('  ' * (tlevel-1),)
                    for i in range(tlevel-1):
                        toc = toc + name[i]
                        if i < tlevel-2: toc = toc + ' '
                    toc = toc + '">%s</A>\n' % (name[tlevel-2],)
                elif tlevel > level:
                    for i in range(level-2, tlevel-2):
                        toc = toc + '%s<UL>\n' % ('  ' * (i+1),)
                    toc = toc + '%s<LI><A HREF="#' % ('  ' * (tlevel-1),)
                    for i in range(tlevel-1):
                        toc = toc + name[i]
                        if i < tlevel-2: toc = toc + ' '
                    toc = toc + '">%s</A>\n' % (name[tlevel-2],)
                    level = tlevel
                elif tlevel < level:
                    for i in range(level-2, tlevel-2, -1):
                        toc = toc + '%s</UL>\n' % ('  ' * i,)
                    toc = toc + '%s<LI><A HREF="#' % ('  ' * (tlevel-1),)
                    for i in range(tlevel-1):
                        toc = toc + name[i]
                        if i < tlevel-2: toc = toc + ' '
                    toc = toc + '">%s</A>\n' % (name[tlevel-2],)
                    level = tlevel
            else:
                body = body + line
        else:
            body = body + line
        line = bdf.readline()
    bdf.close()
    if toc:
        toc = "<UL>\n" + toc + "</UL>\n"
    return (toc, body)

# The document file is the second (and the following) argument (not the last)
document = ""
for arg in args[1:-1]:
    try:
        document = document + open(arg, 'r').read()
    except IOError:
        print "xmlt: Unable to read `%s'.\n" % (arg,)
        print __doc__,
        sys.exit(1)

# Parse the source (xmlt) files
i = 0;
while i < len(document):
    rest = document[i:]
```

```

# The declaration (definition) of a new user defined value tag?
if vtag_expr.match(rest):
    mo = vtag_expr.match(rest)
    for vtag in string.split(mo.group(1), ","):
        vtag_exprs[vtag] = re.compile(
            tag_string % ((vtag,) * 2),
            re.MULTILINE | re.DOTALL)
    i = i + mo.end()

# The definition of new user defined tag?
elif tag_expr.match(rest):
    mo = tag_expr.match(rest)
    tag_exprs[mo.group(1)] = re.compile(
        tag_string % ((mo.group(1),) * 2),
        re.MULTILINE | re.DOTALL)
    tag_impl[mo.group(1)] = (mo.group(2), mo.group(3))
    i = i + mo.end()

# Is a flag set?
elif flag_expr.match(rest):
    mo = flag_expr.match(rest)
    doc.__dict__[mo.group(1)] = 1
    i = i + mo.end()

# Is an attribute given a value?
elif attr_expr.match(rest):
    mo = attr_expr.match(rest)
    doc.__dict__[mo.group(1)] = stripquotes(mo.group(2))
    i = i + mo.end()

# Is this the title?
elif title_expr.match(rest):
    mo = title_expr.match(rest)
    doc.title = mo.group(1)
    i = i + mo.end()

# Is this the body (the actual contents of the document)?
elif body_begin_expr.match(rest):
    #mo = body_expr.match(rest)
    #body = mo.group(1)
    bb = body_begin_expr.match(rest).end()
    mo = body_end_expr.search(rest)
    if mo:
        eb = mo.start()
        ebb = mo.end()
    else:
        eb = ebb = len(rest)
    body = rest[bb:eb]
    j=0; doc.body = ""
    while j < len(body):
        brest = body[j:]

        # Search the body for user defined tags (and do the translation)
        for (name, expr) in tag_exprs.items():
            if expr.match(brest):
                mt = expr.match(brest)
                doc.body = (
```

```
                        doc.body +
                        tag_impl[name][0] +
                        mt.group(1) +
                        tag_impl[name][1])
                    j = j + mt.end()
                    break
            else:
                doc.body = doc.body + brest[0]
                j = j + 1
        i = i + ebb

    # Is this an rcs expression?
    elif rcs_expr.match(rest):
        mo = rcs_expr.match(rest)
        setattr(doc, 'rcs' + mo.group(1), mo.group(2))
        i = i + mo.end()

    # Is this the modified time-stamp?
    elif modified_expr.match(rest):
        mo = modified_expr.match(rest)
        doc.timestamp = mo.group(1)
        i = i + mo.end()

    # Is this any of the user defined value tags? (not inside the body)
    else:
        for (name, expr) in vtag_exprs.items():
            if expr.match(rest):
                mo = expr.match(rest)
                doc.__dict__[name] = mo.group(1)
                i = i + mo.end()
                break

        # The rest of the source is ignored
        else:
            i = i + 1

# Create toc?
if hasattr(doc, "toc"):
    if doc.toc:
        (doc.toc, doc.body) = makeToc(doc.body)

# Open up and parse the layout (dtml file)
try:
    layout = DocumentTemplate.HTMLFile(args[0])
    predoc = DocumentTemplate.HTML(layout(doc))
except IOError:
    print "xmlt: Unable to read `%s'.\n" % (args[0],)
    print __doc__,
    sys.exit(1)

# Open the output file
try:
    output = open(args[-1], "w")
except IOError:
    print "xmlt: Unable to open `%s'.\n" % (args[-1],)
    print __doc__,
    sys.exit(1)
```

```
# Generate the contents of the output file (the HTML document)
output.write(predoc(doc))
```