

tccpage2.py: Dynamic generation of TCC-style web pages with lxml



John W. Shipman

2011-11-12 17:15

Abstract

Describes a Python-language module for the generation of static and dynamic Web pages in the New Mexico Tech Computer Center's style.

This publication is available in Web form¹ and also as a PDF document². Please forward any comments to tcc-doc@nmt.edu.

Table of Contents

1. Introduction	2
2. Overview	2
3. The XHTML to be generated	3
4. tccpage2test: A small test driver	4
5. Code prologue	4
6. Manifest constants	5
6.1. XHTML_DOCTYPE: Document type declaration for XHTML	5
6.2. XHTML_NS: The XHTML namespace URI	5
6.3. CSS_URL: Stylesheet location	5
6.4. TOP_NAV_SEP: Separator string for the top navigational bar	5
6.5. TCC_MAIN_URL: TCC mainpage URL	5
6.6. TCC_LOGO_URL: TCC logo URL	6
6.7. BOT_NAV_SEP: Separator for page-bottom links	6
7. addTextMixed(): Generating mixed content with lxml	6
8. class TCCPage: The page instance interface	7
8.1. TCCPage.__init__(): Constructor	8
8.2. TCCPage.__createHead(): Set up page heading	9
8.3. TCCPage.__createBody(): Set up page body	10
8.4. TCCPage.__topNav(): Set up top nav bar	11
8.5. TCCPage.__topNavItem(): Add one top navigational item	12
8.6. TCCPage.__titleBlock(): Set up page title block	13
8.7. TCCPage.__botNav(): Set up page-bottom navigational links	14
8.8. TCCPage.__botNavItem(): Generate page-bottom navigational link	14
8.9. TCCPage.__botNavShort(): Generate a short-form bottom nav link	15
8.10. TCCPage.__botNavGeneral(): Generate a general bottom nav link	16
8.11. TCCPage.__colophon(): Add colophon section	17

¹ <http://www.nmt.edu/tcc/projects/tccpage2/>

² <http://www.nmt.edu/tcc/projects/tccpage2/tccpage2.pdf>

8.12. <code>TCCPage.write()</code> : Output the finished page	18
9. class <code>NavLink</code> : Describes one navigational feature	18

1. Introduction

The *PyStyler* application is used to maintain the bulk of the pages in the Tech Computer Center³ page structure and the TCC Help System⁴. This application is rather antiquated, the design substantially stable since 1996; see *Building informational webs with PyStyler*⁵ (PDF format) for user-level documentation.

One major limitation of this system is that it treats its web pages as static structure, to be maintained with an ordinary text editor.

A number of TCC scripts generate Web pages outside the *PyStyler* system. Some scripts generate one or more static pages; other scripts are CGI scripts that generate pages dynamically in response to user requests. Ideally, *PyStyler* should be rewritten so that its content template can be applied easily to dynamically generate pages as well as to static pages inside a *PyStyler* document tree. This is a fairly sizeable project, however.

Here, we describe a Python module named `tccpage2.py` that is intended for use by Python scripts that generate web pages. Its purpose is to mimic the appearance of pages generated by *PyStyler* so that they blend visually with static pages.

Warning

Here's why this is a short-term solution. The *PyStyler* system uses a *template file* named `Template` to describe the overall layout of a TCC page. The templates used in the TCC web and TCC help web haven't changed much lately, so they're the basis of the style imitated by the `tccpage2.py` module. But if in the future we want to modify these templates, we'll have to modify the logic in the `tccpage2.py` module in parallel. The best long-term solution is to do a whole new version of *PyStyler* that has a single source for template information that can drive both static and dynamic page generation.

This package is called `tccpage2.py` because it is a rewrite of an earlier package, described in *tccpage.py: Dynamic generation of TCC-style web pages*⁶.

Files generated from this document:

- `tccpage2.py`: The module itself.
- `tccpage2test`: The test driver.

2. Overview

This module uses the techniques of XML generation described in *Python XML processing with lxml*⁷.

Here's the general procedure for generating a TCC-like web page:

1. Import the `tccpage2.py` module as:

```
import tccpage2 as tp
```

³ <http://www.nmt.edu/tcc>

⁴ <http://www.nmt.edu/tcc/help/>

⁵ <http://www.nmt.edu/tcc/projects/pystyler/>

⁶ <http://www.nmt.edu/tcc/projects/tccpage/>

⁷ <http://www.nmt.edu/tcc/help/pubs/pylxml/>

2. Create a `TCCPage` instance and supply it with the desired navigational links. For the constructor and other interfaces to the `tccpage2.py` class, see Section 8, “class `TCCPage`: The page instance interface” (p. 7).
3. If the title is not a simple string—for example, if it contains markup such as `filename` or `code` tags—the content can be added to the `.headTitle` and/or `.bodyTitle` elements.
4. Add the page's content to the `.content` attribute.
5. Use the `.write()` method to send the finished page to its destination.

Note that this module is intended for CGI scripting as well as for the generation of static pages. For a CGI script, simply send the output to the standard output stream, `sys.stdout`.

3. The XHTML to be generated

Here is a general outline of the page to be generated.

1. A `DOCTYPE` identifier for XHTML 1.1. Although the existing *PyStyler* web uses HTML, we want to move toward XHTML, and there's no reason not to use it for generated pages.

Also, the validator at the *W3C Markup Validation Service*⁸ requires a `DOCTYPE` identifier before it will rate the page as valid.

2. A `head` element containing:
 - a. The page's `title` element.
 - b. A link to the standard TCC stylesheet at `/tcc/style.css` on the server.
3. The *top navigational bar*, a set of *top navigational links*. This must fit on a single line, with the links separated by the string `" / "`.

Each link has a `short name` consisting of a generic word or phrase such as “Next” or “Site map”. The word or phrase always appears, even if it is not a link. Thus the layout of the top nav bar is static over the whole site.

4. The formal page header, formatted as a one-row, two-column table with the page title on the left and the TCC logo graphic on the right.
5. The page body, followed by a horizontal rule (`hr` element).
6. A set of *bottom navigational links*. Each appears on a separate line, and the entire line is omitted if that link points nowhere.

Each bottom nav link consists of three parts:

- A boldfaced generic link type, which will be the same as that link in the top nav link, e.g., “Next”, followed by a colon and space.
- One or more destination links, each of which uses the page title (or a close approximation) as its link text. If there are multiple links, they are separated by semicolons.

7. Another horizontal rule, followed by the *colophon*:
 - The “author credit”, which for dynamic pages will credit not a human author, but the script that wrote the page.
 - The date and time when the page was generated.
 - The URL of the page, if known.

⁸ <http://validator.w3.org/>

4. tccpage2test: A small test driver

Here is a small test driver that demonstrates how to generate a page with TCC-style Web navigation, as specified in *TCC Documentation Guidelines*⁹.

tccpage2test

```
#!/usr/bin/env python
#=====
# tccpagetest: Test driver for tccpage.py
#-----

from lxml import etree as et
import tccpage2 as tp

# - - - - -   m a i n   - - - - -

navList = [
    tp.NavLink ( "Next", [("next-title", "next-url")] ),
    tp.NavLink ( "See also",
        [ ("see-title-1", "see-url-1"),
          ("see-title-2", "see-url-2") ],
        noTop=1 ),
    tp.NavLink ( "Previous", [] ),
    tp.NavLink ( "Site map", [('', "site-url")] ),
    tp.NavLink ( "Help", [("help-title", "help-url")] ) ]

page = tp.TCCPage ( "this-is-the-title", navList, "author-text",
                    "url-text" )
p = et.SubElement ( page.content, "p" )
p.text = "Here's some body text."
page.write()
```

5. Code prologue

The implementation of `tccpage2.py` is included here in “literate programming” style. For information on the tool used for code generation, see *A source extractor for lightweight literate programming*¹⁰.

The actual code starts with the conventional Python documentation string:

tccpage2.py

```
"""tccpage2.py: For dynamic generation of web pages in TCC style.

    For documentation, see:
        http://www.nmt.edu/tcc/projects/tccpage/
    """
```

Next, some imports. We need the `sys` module so we can use `sys.stdout` as the default output file. We need the `time` module for timestamping the page.

tccpage2.py

```
#=====
# Imports
```

⁹ <http://www.nmt.edu/tcc/doc/plan/>

¹⁰ <http://infohost.nmt.edu/tcc/help/lang/python/examples/litlxml/>

```
#-----  
  
import sys  
import time
```

Also, we'll need to import the `xmlcreate.py` module discussed in *Python XML processing with lxml*¹¹.
tccpage2.py

```
from lxml import etree as et
```

6. Manifest constants

Next we set up some manifest constants, grouped at the top of the script for easy maintenance.

6.1. XHTML_DOCTYPE: Document type declaration for XHTML

Because the `lxml` package does not support generation of a DOCTYPE declaration, we'll just write one at the top of the file as ordinary text.

```
XHTML_DOCTYPE = (  
    '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"\n'  
    '    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">\n' )
```

tccpage2.py

6.2. XHTML_NS: The XHTML namespace URI

This is the namespace URI that identifies the XHTML document type.

```
XHTML_NS = "http://www.w3.org/1999/xhtml"
```

tccpage2.py

6.3. CSS_URL: Stylesheet location

Gives the URL of the stylesheet for TCC web pages.

```
CSS_URL = "http://www.nmt.edu/tcc/style.css"
```

tccpage2.py

6.4. TOP_NAV_SEP: Separator string for the top navigational bar

This is the string that separates elements of the page-top navigational bar.

```
TOP_NAV_SEP = " / "
```

tccpage2.py

6.5. TCC_MAIN_URL: TCC mainpage URL

This is the URL that people reach if they click on the TCC logo.

¹¹ <http://www.nmt.edu/tcc/help/pubs/pylxml/>

```
TCC_MAIN_URL = "http://www.nmt.edu/tcc/"
```

6.6. TCC_LOGO_URL: TCC logo URL

This is the URL of the official TCC logo.

```
TCC_LOGO_URL = "http://www.nmt.edu/tcc/images/logo.png"
```

6.7. BOT_NAV_SEP: Separator for page-bottom links

When a particular kind of page-bottom link has multiple destinations (e.g., a “See also” link), this string is placed between the links.

```
BOT_NAV_SEP = "; "
```

7. addTextMixed(): Generating mixed content with lxml

The `lxml` package, for all its virtues, does complicate life in one way for programs that generate *mixed content*, that is, XML content that is a mixture of child elements and ordinary text.

Specifically, every `Element` instance has a `.text` attribute that contains any text that occurs before any child elements, and a `.tail` attribute that contains any text that occurs *after the closing tag of that element*.

This function takes care of adding text content to the inside of a given `parent` element. Basically, if `parent` has no child elements, the new text is added to `parent.text`, appending it to the old value if there is any. However, if `parent` does have child elements, the new text is added to the `.tail` attribute of the *last* child element.

```
# - - - a d d T e x t M i x e d
def addTextMixed ( parent, s ):
    """Add text s inside a parent element.

    [ (parent is an et.Element instance) and
      (s is a string) ->
        if parent has no element children ->
            parent.text += s
        else ->
            (last element child of parent).tail += s ]
    """
```

Regardless of where we are putting the text `s`, we have to be sure not to overwrite any existing value there. We use the Python “**or**” operator to produce either the old value, or an empty string, to which `s` is appended before storing it. Note also that the `len()` of an `et.Element` is the number of element children.

```
# - - 1 - -
if len(parent) == 0:
```

```

    #-- 1.1 --
    # [ if bool(parent.text) ->
    #   parent.text += s
    #   else ->
    #     parent.text := s ]
    parent.text = (parent.text or "") + s
else:
    #-- 1.2 --
    # [ let
    #   youngest == (last element child of parent)
    #   in
    #     if bool(youngest.tail) ->
    #       youngest.tail += s
    #     else ->
    #       youngest.tail := s ]
    youngest = parent[-1]
    youngest.tail = (youngest.tail or "") + s

```

8. class TCCPage: The page instance interface

Here's the interface to this class:

tccpage2.py

```

#=====
# Functions and classes
#-----

# - - - - - c l a s s   T C C P a g e   - - - - -

class TCCPage:
    """Represents one page in the TCC style.

    Exports:
        TCCPage ( title=None, navList=None, author=None, url=None,
                  logoImage=None, logoLink=None, cssUrl=None ):
            [ (title is the page's title as a string, default empty) and
              (navList describes the page's standard set of
               navigational links as NavLink instances, default empty) and
              (author is the page's author credit string, default empty) and

              (url is the page's URL as a string, default empty) and
              (logoImage is the URL of the logo image, default
               TCC_LOGO_URL) and
              (logoLink is the URL the logo links to, default
               TCC_MAIN_URL) and
              (cssURL is the stylesheet URL, default CSS_URL) ->
              return a new, empty TCCPage instance representing
              those values ]
        .root: [ self's "html" element as an et.Element ]
        .headTitle:
            [ self's head's title as an et.Element,
              initially the title argument, read/write ]
    """

```

```

.bodyTitle:
  [ self's body title as an h1 et.Element, initially the
    title argument, read/write ]
.content:
  [ a div et.Element in self's body between the top
    and bottom nav links ]
.address:
  [ self's address as an address et.Element, initially
    the author argument, read/write ]
.url:    [ as passed to constructor, read-only ]
.write ( outFile=None ):
  [ outFile is a writeable file, defaulting to sys.stdout ->
    outFile += an XHTML representation of self ]

State/Invariants:
.doc:      [ self's Document instance ]
.head:    [ self's head element ]
.body:    [ self's body element ]
"""

```

The `navList` argument describes a sequence of standard navigational links that appear on the bottom of each page, and optionally on the top as well. This argument is a list of zero or more `NavLink` instances, each of which describes one navigational link and the place or places where that link goes. See Section 9, “class `NavLink`: Describes one navigational feature” (p. 18).

8.1. `TCCPage.__init__()`: Constructor

The constructor for this class creates the XHTML page as a DOM `Document` tree, fills in the top and bottom content, and creates `self.content` as an empty `div` element.

tccpage2.py

```

# - - -   T C C P a g e . _ _ i n i t _ _   - - -

def __init__ ( self, title=None, navList=None, author=None,
              url=None, logoImage=None, logoLink=None, cssUrl=None ):
    """Constructor for a TCCPage."""

```

We divide the construction of the page into these parts: creation of the `Document` instance; creation of its `head` element; and creation of its `body` element.

tccpage2.py

```

#-- 1 --
self.title    = title or ""
self.navList  = navList or []
self.author   = author or ""
self.url      = url or ""
self.logoImage = logoImage or TCC_LOGO_URL
self.logoLink  = logoLink or TCC_MAIN_URL
self.cssUrl    = cssUrl or CSS_URL

#-- 2 --
# [ self.doc := a new ElementTree with root node "html"
# [ self.root := that root node ]
self.root = et.Element ( "html", xmlns=XHTML_NS )

```

```

self.doc = et.ElementTree ( self.root )

#-- 3 --
# [ self.doc := self.doc with a new head element added
#     containing self's head content
#   self.head := that element ]
self.__createHead()

#-- 4 --
# [ self.doc := self.doc with a new body element added
#     containing self's body content
#   self.body := that element
#   self.bodyTitle := the h1 element containing self.title
#   self.content := a div element for the body content
#   self.address := the address element in the colophon ]
self.__createBody()

```

8.2. TCCPage.__createHead(): Set up page heading

This method sets up the page's head element and all its content.

tccpage2.py

```

# - - -   T C C P a g e . _ _ c r e a t e H e a d   - - -

def __createHead ( self ):
    """Set up the page head element.

       [ self.doc := self.doc with a new head element added
         containing self's head content
         self.head := that element ]
    """

```

First we create the actual head element.

tccpage2.py

```

#-- 1 --
# [ self.doc := self.doc with a new head element added
#   self.head := that element ]
self.head = et.SubElement ( self.root, "head" )

```

Next we set up the title element, if any, and set attribute `self.headTitle` to point to it. Unless the value of `self.title` is an empty string, we then add the title text.

tccpage2.py

```

#-- 2 --
# [ self.head := self.head with a new title element
#     added containing self.title (if nonempty)
#   self.headTitle := that element ]
self.headTitle = et.SubElement ( self.head, "title" )
if self.title:
    self.headTitle.text = self.title

```

The stylesheet link is to the URL given by `self.cssUrl`.

```
#-- 3 --
# [ self.head := self.head with a stylesheet link added ]
et.SubElement ( self.head, "link", rel="stylesheet",
                type="text/css",
                href=self.cssUrl )
```

8.3. TCCPage.__createBody(): Set up page body

This method sets up the overall page body structure.

```
# - - - T C C P a g e . _ _ c r e a t e B o d y - - -

def __createBody ( self ):
    """Set up the page body.

       [ self.doc      := self.doc with a new body element added
         containing self's body content
         self.body     := that element
         self.bodyTitle := the h1 element containing self.title
         self.content  := a div element for the body content
         self.address  := the address element in the colophon ]
    """
```

The principal divisions of the body content, in order, are:

1. Top navigational bar.
2. Title block with logo.
3. The `div` element we export as attribute `self.content`. Here, the caller adds the actual page content.
4. Horizontal rule and bottom navigational links.
5. Another horizontal rule and the colophon section.

```
#-- 1 --
# [ self.doc := self.doc with a new body element added
#   self.body := that element ]
self.body = et.SubElement ( self.root, "body" )

#-- 2 --
# [ self.body := self.body with the top nav bar added ]
self.__topNav()

#-- 3 --
# [ self.body := self.body with a title block added
#   self.bodyTitle := the h1 within that title block ]
self.__titleBlock()

#-- 4 --
# [ self.body := self.body with a new div element added
#   self.content := that div element ]
self.content = et.SubElement ( self.body, "div" )
```

```

#-- 5 --
# [ self.body := self.body with bottom nav links added ]
self.__botNav()

#-- 6 --
# [ self.body      := self.body with the colophon added
#   self.address := the address element within that
#         colophon ]
self.__colophon()

```

8.4. TCCPage.__topNav(): Set up top nav bar

The top navigational bar consists of a series of strings separated by " / ". The content comes from the elements of the `self.navList` argument, each of which is a `NavLink` instance. Each string may be a link or not, depending on whether the corresponding instance's `.destList` has any members or not.

tccpage2.py

```

# - - -   T C C P a g e . _ _ t o p N a v   - - -

def __topNav ( self ):
    """Build the row of navigational links across the page top.

    [ (self.navList as invariant) and
      (self.body as invariant) ->
      self.body := self.body with the top nav bar added ]
    """

```

First we create a `div` element to hold the pieces of the nav bar.

tccpage2.py

```

#-- 1 --
# [ self.body := self.body with a new div element added
#   navBar    := that element ]
navBar = et.SubElement ( self.body, "div" )
navBar.attrib["class"] = "top-nav"

```

Now all that remains is to add one link for each element of `self.navList`. The only tricky part is getting the `TOP_NAV_SEP` separator between elements, but not initially or finally. I am indebted to Dr. Allan M. Stavely for the best (and easiest to verify) way to do this. We set a variable called `prefix` to the empty string initially. Each time through the loop, we add a copy of `prefix`, then we add the new content, and then we set `prefix` to `TOP_NAV_SEP`.

tccpage2.py

```

#-- 2 --
prefix = ""

#-- 3 --
# [ navBar += (prefix) + (navBar with top elements of
#   self.navList added, separated by TOP_NAV_SEP
#   prefix := TOP_NAV_SEP ]
for navItem in self.navList:
    #-- 3 body --
    # [ navItem is a NavLink instance ->
    #   if navItem.noTop ->

```

```

#         I
#         else ->
#         navBar += (prefix) + (navItem as a top nav link)
#         prefix := TOP_NAV_SEP ]
if not navItem.noTop:
#-- 3.1 --
# [ navBar += (prefix) + (navItem as a top nav link)
self.__topNavItem ( navBar, navItem, prefix )

#-- 3.2 --
prefix = TOP_NAV_SEP

```

8.5. TCCPage.__topNavItem(): Add one top navigational item

This method translates one NavLink instance into a page-top navigational link. If the given navItem.destList has a URL in it, the link has link text navItem.shortName and points at that URL. (This is intended only for links such as “Next” that have a single URL. If there are multiple URLs, we use the first one.) If there are no URLs, the navItem.shortName attribute is inserted as plain text (not a link).

tccpage2.py

```

# - - - T C C P a g e . _ _ t o p N a v I t e m - - -

def __topNavItem ( self, navBar, navItem, prefix ):
    """Add one item to the top navigational bar.

    [ (navBar is an et.Element) and
      (navItem is a NavLink) ->
      navBar += (prefix) + (navItem as a top nav link) ]
    """

```

First we add the prefix to the parent element; see Section 7, “addTextMixed(): Generating mixed content with lxml” (p. 6).

tccpage2.py

```

#-- 1 --
# [ navBar += prefix ]
addTextMixed ( navBar, prefix )

```

The test “if navItem.destList:” succeeds if that list is nonempty. It fails if navItem.destList is either None or an empty list.

tccpage2.py

```

#-- 2 --
if navItem.destList:
#-- 2.1 --
# [ navBar += a link to navItem.destList[0] with text
#           navItem.shortName ]
title, url = navItem.destList[0]
a = et.SubElement ( navBar, "a", href=url )
a.text = navItem.shortName
else:
# [ navBar += navItem.shortName as text ]
addTextMixed ( navBar, navItem.shortName )

```

8.6. TCCPage.__titleBlock(): Set up page title block

The purpose of this method is to output the small table (one row, two columns) that positions the main page title h1 element to the left of the logo graphic.

Here's the XHTML we're generating:

```
<table width="100%">
  <tr valign="top">
    <td align="left"><h1>self.title</h1>
    <td align="right">
      <a href="self.logoLink">
        </a>
      </td>
  </tr>
</table>
```

Here's the code.

tccpage2.py

```
# - - -   T C C P a g e . _ _ t i t l e B l o c k   - - -

def __titleBlock ( self ):
    """Set up the main page title block.

        [ self.title as invariant ->
          self.body      := self.body with a title block added
          self.bodyTitle := the h1 within that title block ]
    """

    #-- 1 --
    # [ self.body := self.body with a new table element added
    #   table     := that table element ]
    table = et.SubElement ( self.body, "table", width="100%" )

    #-- 2 --
    # [ table := table with a new tr element added
    #   row   := that tr element ]
    row = et.SubElement ( table, "tr", valign="top" )

    #-- 3 --
    # [ row := row with a new cell added containing self.title
    #   inside an h1 element
    #   self.bodyTitle := that h1 element ]
    td = et.SubElement ( row, "td", align="left" )
    self.bodyTitle = et.SubElement ( td, "h1" )
    if self.title:
        self.bodyTitle.text = self.title

    #-- 4 --
    # [ row := row with a new cell added containing the
    #   TCC logo as a link to TCC_MAIN_URL ]
    td = et.SubElement ( row, "td", align="right" )
    a = et.SubElement ( td, "a", href=self.logoLink )
```

```
img = et.SubElement ( a, "img", alt="logo",
                      src=self.logoImage )
```

8.7. TCCPage. __botNav(): Set up page-bottom navigational links

This method outputs the hr element below the page body, followed by page-bottom navigational links made from self.navList.

tccpage2.py

```
# - - - T C C P a g e . _ _ b o t N a v - - -

def __botNav ( self ):
    """Add the page-bottom navigational links section.

    [ (self.body as invariant) and (self.navList as invariant) ->
      self.body := self.body with bottom nav links added
      from self.navList ]
    """
```

First we add the separator rule.

tccpage2.py

```
#-- 1 --
# [ self.body += an hr element ]
et.SubElement ( self.body, "hr" )
```

Then we add the links from self.navList.

tccpage2.py

```
#-- 2 --
# [ self.body += bottom nav links made from self.navList ]
for navItem in self.navList:
    #-- 2 body --
    # [ navItem is a NavLink ->
    #   self.body += a bottom nav link made from navItem ]
    self.__botNavItem ( navItem )
```

8.8. TCCPage. __botNavItem(): Generate page-bottom navigational link

For each NavItem instance in self.navList, there are three cases:

- If the NavItem instance's .destList is empty, no content is generated.
- For some links, such as "Site map", the short name is the same as the title. It would be silly to generate the text "Site map: Site map" with the second repetition being the link.

In that case, the .destList attribute has only one (title, URL) tuple, and the title is empty. We generate a link containing the boldfaced .shortName attribute, pointing at the given URL.

- In the general case, we start by generating the boldfaced .shortName attribute, but not as a link. This is followed by all links to each element of .destList, using the title of the (title, URL) tuple as the link text and the URL as the destination.

tccpage2.py

```
# - - - T C C P a g e . _ _ b o t N a v I t e m - - -
```

```

def __botNavItem ( self, navItem ):
    """Generate one page-bottom navigational link.

    [ navItem is a NavLink instance ->
      self.body += a bottom nav link made from navItem ]
    """

```

First we eliminate the case where nothing is generated.

tccpage2.py

```

#-- 1 --
# [ if navItem.destList is empty or None ->
#   return
#   else ->
#     self.body += a new, empty div element
#     div := that div element ]
if not navItem.destList:
    return
else:
    div = et.SubElement ( self.body, "div" )

```

Next we check for the special case where the short name is the same as the title.

tccpage2.py

```

#-- 2 --
title, url = navItem.destList[0]

#-- 3 --
# [ if bool(title) is false ->
#   div += a link to url with link text navItem.shortName
#   else ->
#     div += (navItem.shortName, boldfaced) +
#             (elements of navItem.destList as links,
#              separated by BOT_NAV_SEP) ]
if not title:
    #-- 3.1 --
    # [ div += a link to url with link text
    #     navItem.shortName ]
    self.__botNavShort ( div, url, navItem.shortName )
else:
    #-- 3.2 --
    # [ div += (navItem.shortName, boldfaced) +
    #           (elements of navItem.destList as links,
    #            separated by BOT_NAV_SEP) ]
    self.__botNavGeneral ( div, navItem )

```

8.9. TCCPage.__botNavShort(): Generate a short-form bottom nav link

This method generates a bottom navigational link in the special case that the link text and the title are the same.

tccpage2.py

```

# - - - T C C P a g e . _ _ b o t N a v S h o r t

def __botNavShort ( self, div, url, shortName ):

```

```

"""Generate a short-form navigational link.

    [ (div is an et.Element) and
      (url is a URL as a string) and
      (shortName is a string) ->
      div += a link to url with link text shortName ]
"""
#-- 1 --
# [ div += a new "a" child element with href=url
#   a := that new child element ]
a = et.SubElement ( div, "a", href=url )

#-- 2 --
# [ a += a new "b" child element with text shortName ]
b = et.SubElement ( a, "b" )
b.text = shortName

```

8.10. TCCPage.__botNavGeneral(): Generate a general bottom nav link

This method generates a bottom navigational link in the general case. The content starts with the bold-faced short name, then all the elements of `navItem.destList` separated by `BOT_NAV_SEP`.

tccpage2.py

```

# - - - T C C P a g e . _ _ b o t N a v G e n e r a l

def __botNavGeneral ( self, div, navItem ):
    """Generate a bottom navigational link in the general case.

        [ (div is an et.Element) and
          (navItem is a NavItem instance) ->
          div += (navItem.shortName, boldfaced) +
                (elements of navItem.destList as links,
                 separated by BOT_NAV_SEP) ]
    """

```

First we add the boldfaced short name. To get separators between multiple links, we use the same prefix trick as `prefix` to the empty string, then set it to the separator after each piece is added.

tccpage2.py

```

#-- 1 --
# [ div += a new "b" child element with text
#         (navItem.shortName)
#   prefix := "" ]
b = et.SubElement ( div, "b" )
b.text = "%s: " % navItem.shortName
prefix = ""

```

Since there may be multiple links, we'll loop through the list of destinations, adding a link each time.

tccpage2.py

```

#-- 2 --
# [ div += (prefix) + (elements of navItem.destList,
#                   made into links, separated by BOT_NAV_SEP)
#   prefix := BOT_NAV_SEP ]

```

```

for title, url in navItem.destList:
    #-- 2 body --
    # [ div += (prefix) + (a link to url with link
    #       text=title)
    #   prefix := BOT_NAV_SEP ]

    #-- 2.1 --
    # [ if bool(prefix) ->
    #   div += prefix
    #   else -> I ]
    if prefix:
        addTextMixed ( div, prefix )

    #-- 2.2 --
    # [ div += (link to url with link text (title)) ]
    a = et.SubElement ( div, "a", href=url )
    a.text = title

    #-- 2.3 --
    prefix = BOT_NAV_SEP

```

8.11. TCCPage.__colophon(): Add colophon section

The last part of the page to be generated contains a horizontal rule, the address element, the time of last update, and the URL (if known).

tccpage2.py

```

# - - - T C C P a g e . _ _ c o l o p h o n - - -

def __colophon ( self ):
    """Add the colophon section to the page.

    [ (self.author as invariant) and (self.url as invariant) ->
      self.body := self.body with the colophon added
      self.address := the address element within that
                    colophon ]

    """

```

First, the horizontal rule:

tccpage2.py

```

#-- 1 --
# [ self.body += an hr element ]
et.SubElement ( self.body, "hr" )

```

Next, the address element. We insert self.author if it is nonempty.

tccpage2.py

```

#-- 2 --
# [ self.body += a new address element containing
#               self.author
#   self.address := that element ]
self.address = et.SubElement ( self.body, "address" )
if self.author:
    self.address.text = self.author

```

We package the timestamp in a `div` element to make it small, since this is not content most people care about.

tccpage2.py

```
#-- 3 --
# [ self.body += current time ]
div = et.SubElement ( self.body, "div" )
div.text = ( "Last updated: %s" %
            time.strftime ( "%Y-%m-%d %H:%M %Z" ) )
```

Finally, we add another small heading with the URL, if it is known.

tccpage2.py

```
#-- 4 --
# [ if self.url ->
#     self.body += self.url ]
if self.url:
    div = et.SubElement ( self.body, "div" )
    div.text = "URL: "
    tt = et.SubElement ( div, "tt" )
    tt.text = self.url
```

8.12. TCCPage.write(): Output the finished page

This method sends the XHTML to the output.

tccpage2.py

```
# - - - T C C P a g e . w r i t e - - -

def write ( self, outFile=None ):
    """Write the page as XHTML."""
    #-- 1 --
    # [ if outFile is None ->
    #     out := sys.stdout
    # else ->
    #     out := outFile ]
    out = outFile or sys.stdout

    #-- 2 --
    # [ out += DOCTYPE for XHTML 1.1 ]
    out.write ( XHTML_DOCTYPE )

    #-- 3 --
    # [ out += XHTML rendering of self.doc ]
    out.write(et.tostring(self.root, pretty_print=True,
                          encoding="utf-8"))
```

9. class NavLink: Describes one navigational feature

Most navigational links have three attributes:

- A *short name* such as “Previous” or “Help”.
- The link text to be used for this link in the bottom nav links. This is generally the page title or a close approximation; see the discussion of “navigational shock” in the *PyStyler* documentation.

- The destination URL of the link.

There are two principal design complications for navigational links:

- Some nav links have multiple destinations. For example, in the TCC help system, some pages list one or more “See also” links: the first is always the parent page in the structure, but additional related pages may be named.
- Some links do not appear in the top nav bar. For example, in TCC help system pages, the “See also” links appear only in the bottom nav link section.

So, here's the interface to the NavLink instance that represents one navigational link.

tccpage2.py

```
# - - - - - c l a s s   N a v L i n k   - - - - -

class NavLink:
    """Represents one navigational feature with zero or more destinations.

    Exports:
        NavLink ( shortName, destList=None, noTop=0 ):
            [ (shortName is the short name of this feature) and
              (destList is a list of (title, url) tuples representing
                places this link should point, defaulting to none) and
              (noTop is true iff this feature should be omitted from
                the top nav bar) ->
                return a new NavLink instance representing those values
            ]
        .shortName:      [ as passed to constructor, read-only ]
        .destList:       [ as passed to constructor, read-only ]
        .noTop:          [ as passed to constructor, read-only ]
    """
    def __init__ ( self, shortName, destList=None, noTop=0 ):
        """Constructor for NavLink."""
        self.shortName = shortName
        self.destList  = destList
        self.noTop     = noTop
```

