homelist2: Generate an index of user homepages

John W. Shipman
2011-11-11 17:31

Abstract

Describes the implementation of a script that indexes all publicly visible personal homepages hosted at New Mexico Tech.

This publication is available in Web form\(^1\) and also as a PDF document\(^2\). Please forward any comments to tcc-doc@nmt.edu.

Table of Contents

1. How do people search for local homepages? ................................................................. 2
2. Online files ..................................................................................................................... 2
3. Design notes .................................................................................................................. 3
   3.1. Where are the personal homepages? ................................................................. 3
   3.2. Page format ......................................................................................................... 3
   3.3. Root page design ............................................................................................... 4
   3.4. Subpage design ................................................................................................. 4
4. Operation of the script .................................................................................................. 4
5. Prologue ....................................................................................................................... 5
6. Imported modules ........................................................................................................ 5
7. Manifest constants ...................................................................................................... 6
   7.1. UID_ATTR ......................................................................................................... 6
   7.2. GECOS_ATTR .................................................................................................... 6
   7.3. LDAP_SERVER ............................................................................................... 6
   7.4. ACCOUNTS_DN ............................................................................................ 6
   7.5. WEB_DIR ........................................................................................................ 7
   7.6. NMT_URL ...................................................................................................... 7
   7.7. TCC_URL ........................................................................................................ 7
   7.8. TCC_ABS ........................................................................................................ 7
   7.9. HTML_SUFFIX ............................................................................................... 7
   7.10. INDEX_NAME .............................................................................................. 7
   7.11. SUBPAGE_NAME ......................................................................................... 7
   7.12. LETTER_PAGE_NAME .............................................................................. 8
   7.13. BASE_TITLE ............................................................................................... 8
   7.14. SPACER ......................................................................................................... 8
8. main(): The main program ......................................................................................... 8
9. checkArguments(): Digest the command line arguments ........................................ 9
10. fatal(): Fatal error .................................................................................................. 10
11. buildKwic(): Survey the users and build the KWIC index .................................... 10

\(^1\) http://www.nmt.edu/tcc/projects/homelist2/
\(^2\) http://www.nmt.edu/tcc/projects/homelist2/homelist2.pdf
1. How do people search for local homepages?

Everyone who has an account at the New Mexico Tech (NMT) Computer Center (TCC) may host a personal homepage with a URL of the form http://www.nmt.edu/~acctname, where acctname is the user's login name.

The purpose of this project is to generate an index to all these homepages so that other members of the NMT community can find them.

Typically people will know either the given name or surname of the person they are looking for. Some campus clubs and other organizations also have homepages.

One venerable technology for automatically generating indexes is KeyWord In Context (KWIC) indexing. This technique is discussed in a related TCC publication, kwic.py: A Python module to generate a Key Word In Context (KWIC) index. This publication discusses how such indexes are structured.

Accordingly, this project provides a script that can be run periodically as a cron job. When run, the script generates a set of pages that index the current personal homepages. Each homepage is indexed using the words in the accountholder's Gecos field in the TCC's LDAP server that defines the attributes of their user account.

- The top-level page describes the overall structure of the index, and provides a link for each letter of the alphabet.
- For each letter of the alphabet, a subpage contains all the index entries for words starting with that letter.

2. Online files

These files pertaining to the project are online.

- homelist2: Python source for the script.
- homelist2.xml: DocBook-XML 4.3 source for this document.

---

1 http://www.nmt.edu/tcc/help/lang/python/examples/kwic/
2 http://en.wikipedia.org/wiki/Cron
3 http://en.wikipedia.org/wiki/Gecos_field
4 http://www.nmt.edu/tcc/projects/homelist2/homelist2
5 http://www.nmt.edu/tcc/projects/homelist2/homelist2.xml
3. Design notes

Implementation will use the Python\(^8\) programming language.

This project is an example of lightweight literate programming\(^9\): the script is embedded within the documentation.

This project was developed using the Cleanroom development methodology.\(^{10}\) In particular, Python comments between [ square brackets ] are Cleanroom intended functions, a form of notation describing what each piece of code is supposed to do.

3.1. Where are the personal homepages?

A user will be considered to have a homepage if they have a world-readable directory named public_html.

We could just ask the HTTP server to serve us the page with the appropriate URL, but this has the undesirable side effect of causing a page fetch that will show up in our Web statistics on user page fetches. Also, why bother the HTTP server when we can just look for the user files directly?

The TCC’s LDAP server will be used as the authority for the current set of user account names. The name of the LDAP server, and the DN (distinguished name) of the account root, will be hardcoded into the script. For more information about LDAP, see the document on the proposed TCC Secure LDAP editing facility\(^{11}\).

The server address is ldaps://ldap0.nmt.edu:636, and the base DN for accounts is ou=accounts,dc=tcc,dc=nmt,dc=edu. We need only two attributes:

- The uid attribute, which is the account name.
- The gecos attribute, which is the person’s real name.

The first version of this script caused a problem because it tried to look for user homepages for all the “machine accounts,” special accounts created for each user area workstation. Because such accounts have their home directories specified as /dev/null, the script caused many failures of the automounter to bind to /dev/null. We get around that by ignoring accounts whose uid attributes end with “$”.

3.2. Page format

In order to generate pages that conform to the standard TCC style, we use a module described in \texttt{tccpage2.py}: Dynamic generation of TCC-style web pages with \texttt{lxml}\(^{12}\).

Here are the navigational links that will appear on all generated pages. Each of these translates into a NavLink instance passed to the TCCPage constructor from the \texttt{tccpage2} module.

1. \texttt{Next}. Inactive on the root page. On all subpages but the last, points to the next subpage.
2. \texttt{Previous}. Inactive on the root page. On all subpages but the first, points to the previous subpage.
4. \texttt{TCC homepage}: http://www.nmt.edu/tcc/.

\(^8\) http://docs.python.org/
\(^9\) http://www.nmt.edu/tcc/~shipman/soft/litprog/
\(^{10}\) http://www.nmt.edu/tcc/~shipman/soft/clean/
\(^{11}\) http://www.nmt.edu/tcc/projects/ldap-edit/
\(^{12}\) http://www.nmt.edu/tcc/projects/tccpage2/
5. **NM Tech homepage**: http://www.nmt.edu/.

### 3.3. Root page design

The `homelist2` script will generate an index page plus one subpage for each letter of the alphabet. The index page is entitled “Index of New Mexico Tech user homepages”. It will start with a boilerplate\(^{13}\) paragraph explaining how the index works. This is followed by a horizontal rule, below which as a line containing the letters of the alphabet, each of which is a link to the subpage for that letter.

### 3.4. Subpage design

The subpage for a given letter will have the title “Index of NMT user homepages: \(X\)” where \(X\) is that page’s letter.

The body of the page will be structured as a two-column table displaying the homepage URL and the KWIC index entry for each user. Because URLs vary much less in width than user names (especially when the user names are the names of organizations), we’ll place the URLs in the left-hand column and the KWIC index entries in the right-hand column. Both will be links to the user page.

Here is how the HTML will look in general:

```html
<table border='1' cellpadding='8'>
<tr>
    <td><a href="http://www.nmt.edu/~username">http://www.nmt.edu/~username</a></td>
    <td><a href="http://www.nmt.edu/~username">keyword suffix, prefix</a></td>
</tr>
... etc. ...
</table>
```

### 4. Operation of the script

The script takes one command line argument:

```
homelist2 path
```

The `path` argument is a path name relative to the directory that is the starting point for the official TCC web:

- `http://www.nmt.edu/tcc/` is the base URL.
- `/u/www/docs/tcc/` is the equivalent absolute pathname of files in that URL.

The script will produce an `index.html` file in that directory for the top-level page, and will also build files for each letter of the alphabet whose names are `x-\(X\).html` where \(X\) is the letter indexed on that page.

For example, if the base page is to have this URL:

```
http://www.nmt.edu/tcc/homelist/index.html
```

then the `crontab` entry on `infohost.nmt.edu` would look like this:

```
0 4 * * * (cd /u/www/docs/tcc/projects/homepage2; \
    nice homelist2 homelist)
```

This runs the script at 4am daily, niced down.

### 5. Prologue

The script starts here, with a line to make it self-executing on Unix boxen, and a comment directing the reader back to this documentation.

```
#!/usr/bin/env python
#================================================================
# homelist2: Script to index NMTCC user homepages.
# # Do not edit this script. It is extracted automatically from
# # the documentation at this location:
# # http://www.nmt.edu/tcc/projects/homelist2/
# #----------------------------------------------------------------
PROGRAM_NAME = "homelist2"
PROGRAM_VERSION = "0.0"
```

### 6. Imported modules

Here we import the modules used by this application.

```
# - - - - - I m p o r t s
We need `sys` for access to the command line and the standard I/O streams. We'll also need the standard `os` and `stat` modules to look for directories and files and determine their permissions.

```
import sys
import os
import stat
```

Python's LDAP interface comes from a SourceForge site\(^\text{14}\).

```
import ldap
```

The `kwic`\(^\text{15}\) module builds the KWIC index.

```
import kwic
```


The `tccpage2`\textsuperscript{16} module handles generation of TCC pages with the standard appearance. Because we use this name in lots of places, we'll shorten it to `tp`.

```python
import tccpage2 as tp
```

To work with the `tccpage2` module, we'll use the XML generation technique described in *Python XML processing with `lxml`*\textsuperscript{17}. We'll refer to this `ElementTree` implementation as `et` for short; `E` is the element factory; `subElement()` adds a child node; and `addText()` adds text to an existing node.

```python
from etbuilder import et, E, subElement, addText
```

## 7. Manifest constants

Python does not have explicit named constants. We use the convention that names in capital letters, with underbars (`_`) between the words, are constants and must never appear on the left side of an equal sign, except in this section where they are defined.

# - - - - - Manifest constants

### 7.1. UID_ATTR

The name of the LDAP attribute that contains the user's account name.

```python
UID_ATTR = 'uid'
```

### 7.2. GECOS_ATTR

Name of the LDAP attribute that contains the name attached to the account, normally of the form “First Last”.

```python
GECOS_ATTR = 'gecos'
```

### 7.3. LDAP_SERVER

The URL for the TCC's LDAP server.

```python
LDAP_SERVER = 'ldaps://ldap0.nmt.edu:636'
```

### 7.4. ACCOUNTS_DN

Distinguished name (DN) of the LDAP root node for user accounts.

```python
ACCOUNTS_DN = 'ou=accounts,dc=tcc,dc=nmt,dc=edu'
```

\textsuperscript{16} [http://www.nmt.edu/tcc/projects/tccpage2/](http://www.nmt.edu/tcc/projects/tccpage2/)

\textsuperscript{17} [http://www.nmt.edu/tcc/help/pubs/pyxml/](http://www.nmt.edu/tcc/help/pubs/pyxml/)
7.5. **WEB_DIR**

Name of the top-level subdirectory in each user account where all their Web pages must reside.

```
WEB_DIR = 'public_html'
```

7.6. **NMT_URL**

The base URL for all NMT homepages (without the following “~”).

```
NMT_URL = "http://www.nmt.edu/
```

7.7. **TCC_URL**

The base URL for all TCC pages.

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

7.8. **TCC_ABS**

Absolute pathname equivalent to TCC_URL.

```
TCC_ABS = "/u/www/docs/tcc/
```

7.9. **HTML_SUFFIX**

Extension identifying HTML files.

```
HTML_SUFFIX = '.html'
```

7.10. **INDEX_NAME**

The preferred name for a top-level page in a directory.

```
INDEX_NAME = 'index' + HTML_SUFFIX
```

7.11. **SUBPAGE_NAME**

This string is used to form the name of the subpage for one letter. For example, the index page for keywords starting with letter J would be named "x_j.html".

```
SUBPAGE_NAME = "x_"
```
7.12. LETTER_PAGE_NAME
This string, plus a letter of the alphabet, forms the name of the page listing links starting with that letter.

```
LETTER_PAGE_NAME = "x_
```

7.13. BASE_TITLE
Title of the index page, and used to form the titles of subpages.

```
BASE_TITLE = "Index of TCC user homepages"
```

7.14. SPACER
Three non-break spaces to place between the letters that link to subpages on the index page.

```
SPACER = u'\xa0' * 3
```

8. main(): The main program
Here is an intended function for the entire script. Not mentioned here is a blanket precondition that the LDAP server is available; if it isn't, nothing else is likely to be working.

```
# - - - m a i n

def main():
    '''Build a KWIC index to all TCC personal homepages.
    
    [ if (command line arguments are valid) and
      (directory specified on command line is writeable) ->
      directory specified on command line := index.html
      top-level page and index X.html subpages for
      every letter X of the alphabet
    else ->
      sys.stderr +:= error message(s) ]

    ...
```

See Section 9, “checkArguments(): Digest the command line arguments” (p. 9).

```
#-- 1 --
# [ if command line arguments are valid ->
#   dirPath := absolute pathname to directory specified on
#   command line
# else ->
#   sys.stderr +:= error message(s)
#   stop execution ]

dirPath = checkArguments()
```

The work progresses in two main phases: accumulating the information for each user into a KwicIndex instance, and building the index pages.
• Section 26, “class WebUser: Encapsulate user data” (p. 23) defines the class that encapsulates the relevant data for one user.
• Section 7.3, “LDAP_SERVER” (p. 6) is the name of the LDAP server.
• Section 11, “buildKwic(): Survey the users and build the KWIC index” (p. 10) does the work.

```python
#-- 2 --
# [ kwicIndex := a kwic.KwicIndex instance indexing the names of
#   users in LDAP_SERVER who have a readable WEB_DIR
#   subdirectory, with the user data for each user
#   represented as a WebUser instance ]
kwicIndex = buildKwic()
```

See Section 16, “buildAllPages(): Build the output pages” (p. 14).

```python
#-- 3 --
# [ if directory (dirPath) is writeable ->
#   directory (dirPath) := (file INDEX_NAME containing links
#   to letter pages) + (one page for each letter (X), named
#   LETTER_PAGE_NAME+(X)+HTML_SUFFIX, displaying the
#   entries in kwicIndex that start with (X)
# else ->
#   sys.stderr += error message(s) ]
buildAllPages(dirPath, kwicIndex)
```

### 9. checkArguments(): Digest the command line arguments

Command line processing is straightforward. There should be exactly one argument, the directory path where the files are to be built.

```python
# - - - c h e c k A r g u m e n t s

def checkArguments():
    #''Check and return the command line argument.
    #
    # [ if command line arguments are valid ->
    #   return the absolute pathname to the directory specified on
    #   the command line
    # else ->
    #   sys.stderr += error message(s)
    #   stop execution ]

    #-- 1 --
    print "=== %s %s ===" % (PROGRAM_NAME, PROGRAM_VERSION)
    argList = sys.argv[1:]

    #-- 2 --
    if len(argList) != 1:
        fatal("Takes one argument, the path to the output directory.")
    else:
        rawPath = argList[0]
```

New Mexico Tech Computer Center
Because the argument is to be interpreted as a path relative to http://www.nmt.edu/, it must not start with a slash, but it must end with one. Consider also the degenerate case where the operator provided "/" as the argument: do not overwrite http://www.nmt.edu/tcc/index.html!

```python
#-- 3 --
# [ if rawPath starts with '/' ->
#   return rawPath[1:]  
# else ->
#   return rawPath ]
if rawPath.startswith('/'): relPath = rawPath[1:]  
else: relPath = rawPath

if len(relPath) == 0:  
    fatal("Path '%s' is not long enough." % rawPath)
if not relPath.endswith('/'):  
    relPath = relPath + '/'
return relPath
```

10. **fatal(): Fatal error**

```python
#-- 3 --
def fatal(*L):
    """Write an error message and stop execution.
    [ L is a list of strings ->
      sys.stderr += concatenation of elements of L
      stop execution ]

    print >>sys.stderr, "*** Fatal error: %s" % (''.join(L))
    raise SystemExit
```

11. **buildKwic(): Survey the users and build the KWIC index**

```python
#-- 3 --
def buildKwic():
    """Build the index of personal web page users.
    [ return a kwic.KwicIndex instance indexing the names of users
      in LDAP_SERVER who have a readable WEB_DIR subdirectory,
      with the user data for each user represented as a WebUser
      instance ]

First we build an empty KWIC index instance.
```

```python
#-- 1 --
# [ if kwic.KwicIndex can find its stop word list ->
```
Section 12, “genWebUsers(): Find users with homepages” (p. 11) handles extraction of the usernames from LDAP and selecting the ones that have a personal homepage. For each such user, it creates an instance of Section 26, “class WebUser: Encapsulate user data” (p. 23). These instances are then passed to Section 15, “indexUser(): Add one user to the KWIC index” (p. 14) to be added to kwicIndex.

The logic that queries the LDAP server for user accounts is Section 13, “genAllUsers(): Find all the users in LDAP” (p. 11). The logic that checks to see if a given user has a personal Web page is Section 14, “isWebUser(): Does this user have a personal Web?” (p. 13).
def genAllUsers():
    '''Generate the LDAP entries for all real user accounts.

    [ generate a sequence of WebUser instances representing all
      users in LDAP_SERVER ]
    ...

First, we bind to the LDAP server. Anonymous binding is sufficient to get the account information we
want. See Section 7.3, “LDAP_SERVER” (p. 6).

#-- 1 --
# [ LDAP_SERVER names an accessible LDAP server ->
#      anon := an LDAP object representing an anonymous bind
to LDAP_SERVER ]
try:
anon = ldap.initialize ( LDAP_SERVER )
anon.bind ( "", "", ldap.AUTH_SIMPLE )
except ldap.LDAP_Error, detail:
fatal("Couldn't bind to the LDAP server.")

Next, we use the LDAP module's search function to search first-level children of the node specified by
ACCOUNTS_DN. The filterstr argument filters out entries that have no uid attribute. The attrlist
argument requests that only the attributes named in the list are returned. See Section 7.4, “ACCOUNTS_DN” (p. 6), Section 7.1, “UID_ATTR” (p. 6), and Section 7.2, “GECOS_ATTR” (p. 6).

#-- 2 --
# [ anon is a bound LDAP object ->
#    ldapResult := a list of LDAP result tuples for
#    a search of the immediate children of ACCOUNTS_DN that
#    have a UID_ATTR attribute, containing the UID_ATTR and
#    GECOS_ATTR attributes ]
ldapResult = anon.search_s ( ACCOUNTS_DN, ldap.SCOPE_ONELEVEL,
    filterstr="(%s=*)" % UID_ATTR,
    attrlist=[UID_ATTR, GECOS_ATTR] )

The structure of ldapResult is described by this excerpt from LDAP programming with Python:

    Each result tuple is of the form (dn, attrs), where dn is a string containing the DN
    (distinguished name) of the entry, and attrs is a dictionary containing the attributes
    associated with the entry. The keys of attrs are strings, and the associated values are
    lists of strings.

So to produce the return value, we pull out the second element of each tuple in ldapResult to get the
attribute map, then form a tuple from the first element of the list for each attribute we want.

Two more complications. Some LDAP entries don't have a GECOS field. In that case we just skip that
entry. Also, in the Bad Old Days, there was a convention that GECOS fields contained not just the name,
but three more fields separated by commas. So we use only the part of the GECOS field up to the first
comma, if there is one.

#-- 3 --
# [ ldapResult is an LDAP-search-format result list ->
#    generate a sequence of WebUser instances representing
#    the usernames and GECOS fields made from ldapResult ]
for dn, attrs in ldapResult:
    uid = attrs[UID_ATTR][0]
    try:
        rawGecos = attrs[GECOS_ATTR][0]
        commaPos = rawGecos.find‚”‚”
        if commaPos >= 0:
            gecos = rawGecos[:commaPos]
        else:
            gecos = rawGecos
        yield WebUser(uid, gecos)
    except KeyError:
        pass

#-- 4 --
raise StopIteration

14. isWebUser(): Does this user have a personal Web?

    # - - - i s W e b U s e r

    def isWebUser(webUser):
        """See if this user has a personal homepage.

        [ webUser is a WebUser instance ->
          if webUser has a personal homepage ->
            return True
          else -> return False ]
    ""

First of all, we ignore account names that end in “$”; they are special TCC accounts.

    #-- 1 --
    if webUser.login[-1] == '$':
        return False

For a user account N, their home directory is at path “/u/N”, and their HTML directory must be a first-level directory whose name is given by Section 7.5, “WEB_DIR” (p. 7).

    #-- 1 --
    # [ webPath := absolute path to the HTML directory for user
    #   uid ]
    webPath = ’/u/%s/%s/’ % (webUser.login, WEB_DIR)

    #-- 2 --
    # [ if webPath names an existing path ->
    #   mode := permissions word (mode bits) for webPath
    #   else -> return 0 ]
    try:
        statusTuple = os.stat (webPath)
        mode = statusTuple[stat.ST_MODE]
    except OSErr0:...
return 0

#-- 3 --
# [ if mode has world read and world execute bits both set ->
#    return 1
#  else ->
#    return 0 ]
return bool ( (mode & stat.S_IROTH) and
             (mode & stat.S_IXOTH) )

15. indexUser(): Add one user to the KWIC index

def indexUser(kwicIndex, webUser):
    '''Index all the keywords in one user's GECOS name.

    [ (kwicIndex is a kwic.KwicIndex instance) and
      (webUser is a WebUser instance) ->
      kwicIndex := kwicIndex with entries added indexing
      the user represented by webUser ]

    ...'

The entries in a KwicIndex carry a user data value. In our case, that value will be the WebUser instance,
so that the generated index entry can link to their homepage URL.

16. buildAllPages(): Build the output pages

def buildAllPages(dirPath, kwicIndex):
    '''Generate the index page and all subpages.

    [ (dirPath is a path relative to http://www.nmt.edu/) and
      (kwicIndex is a kwic.KwicIndex instance) ->
      if directory (dirPath) is writeable ->
      directory (dirPath) := (file INDEX_NAME containing links
to letter pages) + (one page for each letter (X), named
LETTER_PAGE_NAME+(X)+HTML_SUFFIX, displaying the
entries in kwicIndex that start with (X)
      else ->
      sys.stderr += error message(s) ]

Although we say there is a page for each letter, in practice, because Unicode is involved, we can't just
spit out pages for A through Z. Also, if there don't happen to be any keywords that start with a partic-
ular letter, we don’t want to build a page for that letter. Furthermore, the navigational links on each subpage need to know the title and URL of the preceding and following pages.

So our first step is to build a `letterList` containing the single Unicode characters in whatever order their code points sort them. See Section 17, “`scanInitialLetters()`: What initial letters occur in keywords?” (p. 15).

```python
#-- 1 --
# [ letterList := a list of the unique initial letters of keywords
#   in kwicIndex, in ascending order by code point, upshifted ]
letterList = scanInitialLetters(kwicIndex)
```

See Section 18, “`buildIndexPage()`: Build the top-level page” (p. 16). The index page doesn’t need the actual KWIC index, just the list of subpage letters.

```python
#-- 2 --
# [ directory (dirPath) is writeable ->
#   directory (dirPath) += an INDEX_NAME file linking
#   to the subpages in (letterList) ]
buildIndexPage(dirPath, letterList)
```

See Section 23, “`buildSubpage()`: Build the subpage for one letter” (p. 19).

```python
#-- 3 --
# [ directory (dirPath) += (subpages for each letter in
#   letterList, each displaying the entries in kwicIndex
#   for that letter) ]
for k in range(len(letterList)):
  #-- 3 body --
  # [ directory (dirPath) += (a subpage for letterList[k],
  #   using letterList to find the next and previous
  #   pages, displaying entries in kwicIndex for
  #   letterList[k])
  buildSubpage(dirPath, letterList, k, kwicIndex)
```

### 17. `scanInitialLetters()`: What initial letters occur in keywords?

```python
#--- scanInitialLetters

def scanInitialLetters(kwicIndex):
    '''Find the sequence of letters to be indexed.

    [ kwicIndex is a kwic.KwicIndex instance ->
      return a list of the unique initial letters of keywords in
      kwicIndex, in ascending order by code point, upshifted ]
    ...
```

We’ll use a Python set to find the unique letters in all the keywords. The `kwic.KwicIndex.genWords()` method generates a sequence of `kwic.KwicWord` instances; in each of these instances, the `.word` attribute is the keyword in Unicode form.
letterSet = set()

for kwicWord in kwicIndex.genWords():
    letterSet.add(kwicWord.word[0].upper())

Python's built-in sorted() function produces a sorted list from any sequence, including sets.

return sorted(letterSet)

---

18. buildIndexPage(): Build the top-level page

```python
# - - - buildIndexPage

def buildIndexPage(dirPath, letterList):
    '''Build the top-level page pointing to the subpages.

    (dirPath is a path relative to http://www.nmt.edu/tcc/) and
    (letterList is a list of the initial letters of the
    subpages in ascending order) ->
    directory (dirPath) is writeable ->
    directory (dirPath) += an INDEX_NAME file linking
    to the subpages in (letterList) ]
    ...'''

    For a discussion of the navigational elements, see Section 3.2, “Page format” (p. 3). See also Section 19,
    “relToURL(): URL of a relative path” (p. 17) and Section 20, “relToAbs(): Absolute path from a
    relative path” (p. 17).

    # - - -
    # [ indexURL := URL of dirPath+INDEX_NAME
    # indexPath := absolute pathname to dirPath+INDEX_NAME ]
    # page := a new tp.TCCPage instance with standard
    # TCC navigation ]
    indexURL = relToURL(dirPath, INDEX_NAME)
    indexPath = relToAbs(dirPath, INDEX_NAME)
    navList = [
        tp.NavLink("Next", ()),
        tp.NavLink("Previous", ()),
        tp.NavLink("TCC help system",
            ["", "http://www.nmt.edu/tcc/help/"]],
        tp.NavLink("TCC homepage",
            ["", "http://www.nmt.edu/tcc/"]],
        tp.NavLink("NM Tech homepage",
            ["", "http://www.nmt.edu/"])
    ]
    page = tp.TCCPage(BASE_TITLE, navList, url=indexURL)
```
At this point, `page.content` is a `div` element under which we add the page’s content. First to be added is the instructional paragraph, then a horizontal rule.

```python
#-- 2 --
# [ page.content += instructional paragraph ]
page.content.append (E.p("To find a homepage, click on the first letter of their "
    "first or last name.") )
page.content.append (E.hr())
```

Each subpage link is followed by three nonbreak spaces.

```python
#-- 3 --
# [ page.content += links to the subpages for elements of letterList ]
for letter in letterList:
    addSubpageLink(dirPath, page.content, letter)
    addText(page.content, SPACER)
```

```python
#-- 4 --
# [ if indexPath is writeable ->
#     file (indexPath) := XHTML rendering of page
# else ->
#     sys.stderr += error message(s) ]
try:
    outFile = file(indexPath, 'w')
except IOError, details:
    fatal("Can't write index page '%s': %s" %
        (indexPath, details))
page.write(outFile)
outFile.close()
```

### 19. `relToURL()`: URL of a relative path

Converts `dirPath` plus a page name to a URL.

```python
# - - - r e l T o U R L
def relToURL(dirPath, pageName):
    '''Find the URL of a given page.

    [ (dirPath is a path relative to TCC_URL) and
      (pageName is a page name) ->
      return the equivalent URL of that page ]
    ...
    return ''.join((TCC_URL, dirPath, pageName))
```

### 20. `relToAbs()`: Absolute path from a relative path

Converts `dirPath` plus a page name to an absolute path.
# relToAbs

def relToAbs(dirPath, pageName):
    '''Find the absolute path to a given page.
    [ (dirPath is a path relative to TCC_ABS) and
      (pageName is a page name) ->
      return the equivalent absolute path name ]
    ...'''
    return ''.join((TCC_ABS, dirPath, pageName))

21. addSubpageLink(): Link from the index page to one subpage

# addSubpageLink

def addSubpageLink(dirPath, node, letter):
    '''Build a link from the index page to the subpage for (letter).
    [ (dirPath is a path relative to TCC_URL) and
      (node is a div et.Element) and
      (letter is a single unicode character) ->
      node += a link to the subpage for letter ]
    ...'''

The URL of the subpage is determined by Section 22, “letterInfo(): Information about a subpage” (p. 18).

#-- 1 --
# [ subURL := URL for letter ]
subURL, subPath, subTitle = letterInfo(dirPath, letter)

#-- 2 --
# [ node := node with a new link to (subURL) using (letter) as the link text ]
nodes.append ( E.a ( letter, href=subURL ) )

22. letterInfo(): Information about a subpage

# letterInfo

def letterInfo(dirPath, letter):
    '''Find the URL, path, and title of the subpage for one letter.
    [ (dirPath is a path relative to TCC_URL) and
      (letter is a single unicode character) ->
      return (URL of letter's page, absolute path to letter's
      ...'''

Because the letter may be a non-ASCII Unicode character, we can’t use it directly in the name of the letter page. Instead, for code points past a lowercase “z”, we’ll use a string of the form “uNNNN” where NNNN is the hexadecimal value of the code point.

```python
n = ord(letter)
if n <= ord('z'): suffix = str(letter)
else: suffix = "u%04x" % n
```

See Section 19, “relToURL() : URL of a relative path” (p. 17) and Section 20, “relToAbs() : Absolute path from a relative path” (p. 17). Because letter may be a non-ASCII Unicode character, the returned title must be Unicode.

```python
url = relToURL(dirPath, pageName)
path = relToAbs(dirPath, pageName)
title = (u"%s: %s" % (BASE_TITLE, letter))
```

## 23. buildSubpage() : Build the subpage for one letter

```python
def buildSubpage(dirPath, letterList, k, kwicIndex):
    """Build the subpage for letterList[k]."
    [ (dirPath is a path relative to http://www.nmt.edu/tcc/) and
    (letterList is a list of the unique initial letters in kwicList) and
    (0 <= k < len(letterList)) and
    (kwicIndex is a kwic.KwicIndex instance) ->
    directory (dirPath) += (a subpage for letterList[k],
    using letterList to find the next and previous pages, displaying entries in kwicIndex for
```
letterList[k])

For the logic that creates a TCCPage, see Section 24, “emptySubpage(): Set up an empty subpage” (p. 20).

```python
def emptySubpage(dirPath, letterList, k):
    '''Create an empty subpage for letterList[k].

    (dirPath is a path relative to TCC_URL) and
    (letterList is the list of initial letters of keywords) and
    (0 <= k < len(letterList)) ->
    return ((absolute path to page for letterList[k]),
            (a new tp.TCCPage instance for letterList[k]
             with links to previous and next letters in letterList)
            )
```

The .genWords() method on kwicIndex generates all the keywords that start with our letter, but we have to test to see when that sequence terminates. See Section 25, “addRow(): Add one row to the subpage table” (p. 22).

```python
for kwicRef in kwicWord.genRefs():
    addRow(table, kwicRef)
else:
    break
```

```python
try:
    outFile = file(pagePath, 'w')
except IOError, details:
    fatal("Can't open subpage '%s' for letter '%r': %s" %
          (pagePath, letter, str(details)))
    page.write(outFile)
```
First we find the letter for this subpage and upshift it. Then we figure out the subpage's URL, path, and title; see Section 22, "letterInfo(): Information about a subpage" (p. 18).

```python
#-- 1 --
# [ letter := letterList[k].upper()
# pageURL := URL for that letter
# pageAbs := absolute path name to subpage for that letter
# pageTitle := title for that letter ]
letter = letterList[k].upper()
pageURL, pageAbs, pageTitle = letterInfo(dirPath, letter)
```

Then we have to determine whether the Next and Previous links are live or not. The constructor for tccpage2.NavList specifies navigational link targets as a list of (title, URL) tuples: an empty list results in a dead link.

```python
#-- 2 --
# [ if k > 0 ->
#     prevTitleURL := [(title of page for letterList[k-1],
#     URL of page for letterList[k-1])]
# else ->
#     prevTitleURL := an empty list ]
if k > 0:
    prevLetter = letterList[k-1].upper()
    prevURL, prevPath, prevTitle = letterInfo(dirPath, prevLetter)
    prevTitleURL = [(prevTitle, prevURL)]
else:
    prevTitleURL = []

#-- 3 --
# [ if (k+1) < len(letterList) ->
#     nextTitleURL := [(title of page for letterList[k+1],
#     URL of page for letterList[k+1])]
# else ->
#     nextTitleURL := an empty list ]
if (k+1) < len(letterList):
    nextLetter = letterList[k+1].upper()
    nextURL, nextPath, nextTitle = letterInfo(dirPath, nextLetter)
    nextTitleURL = [(nextTitle, nextURL)]
else:
    nextTitleURL = []

Now we're ready to build the TCCPage instance.

```python
#-- 3 --
# [ page := a new tp.TCCPage instance with title (pageTitle),
#     next page target (nextTitleURL), previous page target
#     (prevTitleURL), and other standard TCC navigation ]
navList = [
    tp.NavLink("Next", nextTitleURL),
    tp.NavLink("Previous", prevTitleURL),
]```
The only page content is the table. For the HTML structure, see Section 3.4, “Subpage design” (p. 4).

25. addRow(): Add one row to the subpage table

```python
#- - - a d d R o w

def addRow(table, kwicRef):
    """Add one row to the subpage's table of homepage links.

    [ let
        uid = kwicRef.userData.login
        in ->
            (table is a table et.Element) and
            (kwicWord is a kwic.KwicRef) ->
                table += a tr et.Element containing two td et.Elements,
                the first a link to the homepage URL for uid
                with that URL as the link text, the second a link
                to the same place with str(kwicRef) as the
                link text ]
    ...
```

For the HTML generated here, see Section 3.4, “Subpage design” (p. 4).
26. **class WebUser: Encapsulate user data**

An instance of this class holds the relevant data about one user with a personal Web: basically just their name (from the `gecos` field in LDAP) and their account name.

```python
class WebUser(object):
    '''Represents one user with a personal web.

    Exports:
    WebUser(login, gecos):
        [ (login is a user's account name) and
          (gecos is that user's GECOS name) ->
          return a new WebUser with those values ]
    .login: [ as passed to the constructor, read-only ]
    .gecos: [ as passed to the constructor, read-only ]

    def __init__(self, login, gecos):
        '''Constructor.
        '''
        self.login = login
        self.gecos = gecos
```

27. **Epilogue**

```python
if __name__ == '__main__':
    main()
```