2/47

LICENSING SUPPORT SYSTEM (LSS) TEST BED SYSTEM DESCRIPTION REPORT

Prepared for

Nuclear Regulatory Commission
Contract NRC-02-93-005

Prepared by

Center for Nuclear Waste Regulatory Analyses
San Antonio, Texas

September 1996

3/41

LICENSING SUPPORT SYSTEM (LSS) TEST BED SYSTEM DESCRIPTION REPORT

Prepared for

Nuclear Regulatory Commission
Contract NRC-02-93-005

Prepared by

Aaron DeWispelare
Kent Polk
Robert Marshall

Center for Nuclear Waste Regulatory Analyses
San Antonio, Texas

September 1996

CONTENTS

5/47

FIGURES

6/47

ACKNOWLEDGMENTS

This report was prepared to document work performed by the Center for Nuclear Waste Regulatory Analyses (CNWRA) for the Nuclear Regulatory Commission (NRC) under Contract No. NRC-02-93-005. The activities reported here were performed on behalf of the NRC Office of Nuclear Material Safety and Safeguards (NMSS), Division of Waste Management (DWM). The report is an independent product of the CNWRA and does not necessarily reflect the views or regulatory position of the NRC.

QUALITY OF CODE DEVELOPMENT

ANALYSES AND CODES: This computer code described in this document is not controlled under the CNWRA Software Configuration Procedures. This report describes the development of Licensing Support System Test Bed (LSSTB) computer code, however, the code has not been sufficiently developed to be placed under the CNWRA Configuration Management system.

7/47

1 INTRODUCTION

Under the Nuclear Waste Policy Act (NWPA), Statute 114(d), the Nuclear Regulatory Commission (NRC) is required to review the U.S. Department of Energy (DOE) license application (LA) for construction of a high-level waste (HLW) repository within a three-year time period. To assist in meeting this timeframe, the NRC has revised the Commission Rule of Practice in the Code of Federal Regulations (CFR) Title 10, Part 2 (10 CFR Part 2), for the adjudicatory proceeding on the LA to incorporate the use of an electronic information management system, known as the HLW Licensing Support System (LSS).

The LSS will contain documentary material of the DOE, NRC, and other parties to the HLW licensing proceeding that may be relevant to the LA review process. It is intended to facilitate the review by supporting: (i) centralized submission of discoverable documents before the DOE application is filed, (ii) early technical review of materials by all potential parties, and (iii) electronic transmission of all filings during the hearing. To test a concept of the electronic information management, the NRC undertook the development of LSSTB.

Section 1 of this report provides the purpose and basic description of the Licensing Support System Test Bed (LSSTB). Section 2 presents a functional description of the LSSTB server including software and database structure description. Section 3 describes the current status, and Section 4 discusses potential improvements to the LSSTB to increase its utility.

1.1 OBJECTIVES

The NRC LSSTB is intended to establish a computer server accessible through the public Internet (outside the NRC network security firewall) which will (i) provide a basic document search and retrieval capability from among a sample of HLW documents, (ii) allow downloading text of documents and associated linked images selected by the user, and (iii) obtain feedback from users by allowing on-line comments. The next goal of the LSSTB is to provide a mechanism for users to submit for loading pertinent HLW documents.

1.2 DESCRIPTION

The Center for Nuclear Waste Regulatory Analyses (CNWRA) has established a public Internet computer server site which can be accessed by commonly available Word Wide Web (WWW) browsers (e.g., MOSAIC and Netscape). The Internet address of this site is *lssnet-test.cnwra.swri.edu*. Authorization was obtained from the NRC before the server was placed on the Internet for public availability. This server has an NRC LSSTB Home Page which was produced after prototyping and design consultation with the NRC. This WWW server Home Page provides the user interface for searching the loaded documents. Development of this server included producing an interface to four types of user searches (i.e., word and phrase searches on title, author, date, and document full text). The public users can view and download documents which result from their searches. The server hardware utilizes a SUN SPARC20 workstation. The search and retrieval software used by this web server is the Verity Inc. TOPIC Internet Server (TIS). This commercial, off-the-shelf (COTS) software was chosen because of compatibility with the search and retrieval software in the NRC Consolidated Document Management System (CDOCS), and because it could be acquired quickly. This WWW server application also provides efficient on-line logging of user comments.

1.3 DATABASE CONTENT

Documents in the LSSTB database are organized in "collections." Collections are groupings of documents that have the same content and characteristics. The primary reason for establishing collections is to allow the user to include or exclude collections in queries to make those queries more specific and to decrease the search time by decreasing the number of documents that need to be searched. Although the LSSTB currently does not provide for searching in selected collections, the database is organized to allow this in the future. The LSSTB presently has collections for technical reports, NUREGs, regulations, correspondence, and documents from the Commission Decision Tracking System (CDTS).

The LSSTB database contains documents loaded from the NRC HLW program including SECY papers, regulations, NUREGs, and correspondence. Loading of more NRC HLW program documents will continue in the future with many more documents consisting of technical reports, correspondence, and NUREGs to be added in the coming months.

Documents are available in the LSSTB in various formats which allow a WWW user to review and download files to meet their needs. These formats consist of: (i) header file [American Standard Code for Information Interchange (ASCII) format with bibliographic information], (ii) text file (ASCII), (iii) word processor file (currently WordPerfect 5.1), (iv) page images (TIFF graphics format), and (v) Portable Document Files (PDF). Bibliographic header and text files are available on all documents, and depending on the source of the documents, additional files consisting of a word processor version, page images, and PDF may also be available.

9/47

2 FUNCTIONAL DESCRIPTION

2.1 LICENSING SUPPORT SYSTEM TEST BED SOFTWARE DESCRIPTION

The LSSTB provides WWW document search services using two "servers" which provide responses to information requests. These two servers are linked together via custom programs.

The first server is a HyperText Transfer Protocol (HTTP) server, commonly known as a WWW or "Web" server. All direct client interactions with the LSSTB are done through the HTTP server using an HTTP client "browser." Examples of HTTP client browsers are Lynx, MOSAIC and Netscape. The LSSTB uses the "Apache" HTTP server running on a SUN UNIX workstation. The Apache server is the most common HTTP server in use. The Apache server software including source code is freely available. This server must be compiled for the system on which it is to be run using an ANSI C compiler. The LSSTB platform includes such a compiler and the Apache server executable can be rebuilt either on the LSSTB computer or a comparable computer.

The LSSTB HTTP server responds to HTTP client requests for information such as HyperText Markup Language (HTML) documents (the WWW standard), LSSTB database search results and database documents. The LSSTB HTTP server also provides the client HTTP browser with information on how to present the response to the client (user). The LSSTB logs all transactions and feedback responses and provides summaries and statistics concerning this information to administrative personnel. Figure 2-1 depicts the information flow in the LSSTB.

The second server is a commercial document "search engine" purchased from Verity Inc. The TOPIC Internet Server (TIS) was designed to provide document search capabilities to HTTP servers such as Apache. The LSSTB requires that all TIS search queries be routed through the Apache HTTP server for security and ease-of-use.

The Apache HTTP server does not contact the TIS directly. All search queries and results are processed through custom programs which prepare the search requests so they are compatible with the TIS and reformat the search results to make it more pleasing and usable by the client. These intermediate programs use the Common Gateway Interface (CGI) mechanism provided by most HTTP servers and as such are called CGI programs. Currently, all LSSTB CGI programs except the Verity CGI interface to TIS are written using the Python language. Python is an interpretive, object-oriented language often used for HTTP support applications.

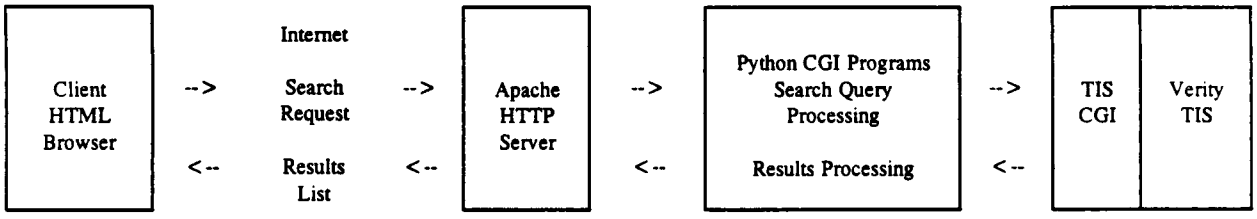

Figure 2-1. Licensing Support System Test Bed interface and server

The TIS is only used to retrieve document references and is not used to retrieve the actual documents stored in the database. The document references are listed in the search results information which is presented to the client HTTP browser as an HTML document. The client HTTP browser will display the reference list and allow the client to choose which document to retrieve from the database. The Apache HTTP server directly retrieves all documents which are selected by the client and transfers then to the client HTTP browser. This document processing flow is shown in Figure 2-2.

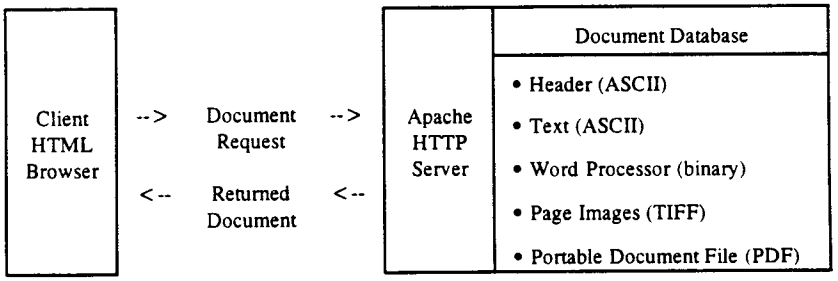

Figure 2-2. Document request flow

All custom programming has been written using the Bourne shell (UNIX scripting language) and the Python Language.

2.2 DESCRIPTION OF THE NUCLEAR REGULATORY COMMISSION WEB DIRECTORY STRUCTURE

There are two directories which comprise the NRC web directory for the LSSTB. These are: (i) database and (ii) htdocs. The information contained in these directories are described in the following sections.

2.2.1 Database Directory

The database directory contains two subdirectories, "collections" and "docs," and a python script file, "insert.py". The script file insert.py is used by the system administrator to add new documents to the collections. It may also be invoked by a batch process to automatically add new documents in an unattended mode during off-peak times. When new documents are added, the collection must be "rebuilt" by the server so that the contents of the new documents are available to all parts of the LSSTB.

The "collections" subdirectory is the parent directory of the TOPIC collections. The subdirectories under this directory correspond in name to TOPIC collections. For example, the "nuregs" subdirectory contains all the files and directories that TOPIC needs to generate a collection for NUREG documents (Figure 2-3).

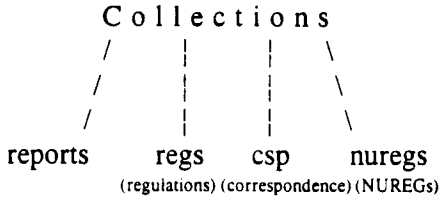

Figure 2-3. Collections subdirectories

The "docs" subdirectory is where the documents are actually stored. The directory structure of the docs directory is parallel to that of the collections directory, at the first level, where the names of the subdirectories equate to a single TOPIC collection (Figure 2-4).

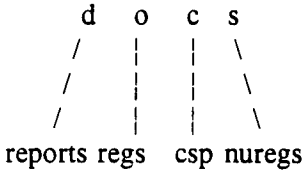

Figure 2-4. Docs subdirectory

Each of these subdirectories are broken down further into subdirectories. These lowest level subdirectories are numbered consecutively, beginning at the number zero (e.g., 0, 1, 2...), and these numbers correspond to the subdirectory name. This directory nomenclature convention facilitates document handling. Each of these subdirectories may contain up to 1,024 subdirectories (a system constraint), each of which represents a single document set, numbered 000001–001023 for the first subdirectory, 001024–002047 for the second, etc. The file names are all six characters in length to allow for a large, but not unwieldy, number of files for a given collection. Each subdirectory also includes a system generated reference file (index.html) to allow the browser to locate the document files which are identified in a document search. Finally, each of these subdirectories contain the files that are inserted with that particular document, with the following naming convention:

subdirectoryname.header	—	ASCII header file
subdirectoryname.tex	—	ASCII text file
subdirectoryname.wp	—	WordPerfect file, if one exists
subdirectoryname.doc	—	MS Word, if one exists
subdirectoryname.pdf	—	PDF file, if one exists
nnn.tif	—	Image files, numbered 001.tif, 002.tif...nnn.tif, if they exist
index.html	—	HTML index file

For example, Figure 2-5 shows a directory structure for the "reports" collection of documents.

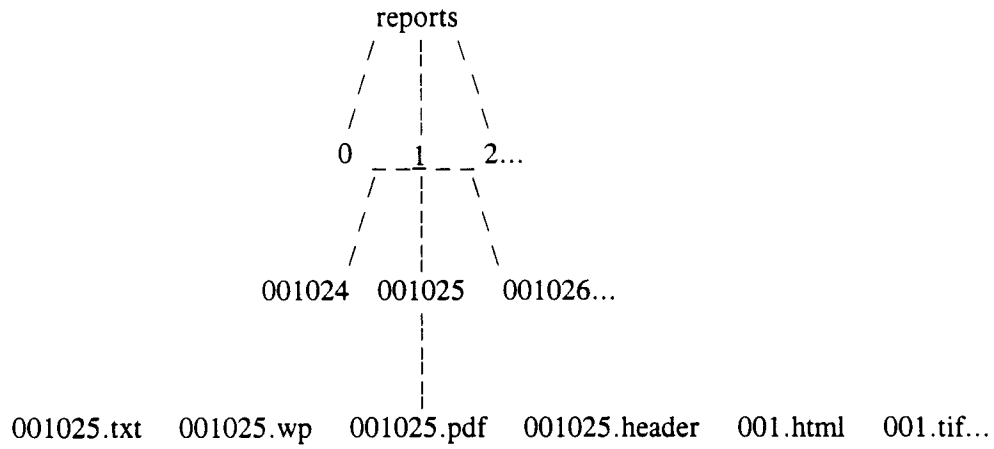

Figure 2-5. Sample directory structure used in the Licensing Support System Test Bed

2.2.2 HTDOCS Subdirectory

The htdocs subdirectory contains HTML documents that are used by the client browser to display all the pages associated with the LSSTB system. These documents include all help pages, introduction page, the welcome and purpose pages, all search pages, and the survey page. Htdocs contains no subdirectories at this time.

13/47

3 STATUS AND FUTURE PLANS

3.1 INTERNET SERVER STATUS

The LSSTB prototype was brought on-line for NRC internal testing and evaluation in August 1996. Following feedback on operability and functionality, the prototype was modified. Several sample documents from the HLW program were added to the LSSTB prototype database. The prototype was renamed the LSSTB, and the server was moved outside the NRC computer network security firewall and established on the Internet. The LSSTB was opened to public access the first part of September 1996, and has remained in continual operation since then. Comments on operability and database content are being recorded and saved for evaluation. The database of the LSSTB has over 350 sample documents and more documents will be added in FY97.

3.2 FUTURE NEEDS OF THE LICENSING SUPPORT SYSTEM TEST BED

There are several additions and modifications that will increase the utility and maintainability of the LSSTB. These improvements are briefly discussed in the following section.

3.2.1 Simple Document Submission Mechanism for Users

This mechanism would allow users to submit documents of their own into the system. Documents would be saved to a staging area to allow a custodian or administrator to examine the document(s) prior to actually writing them into the database.

Implementing this function will require producing a new submission form and an upload mechanism. The user would be queried for information about the document being submitted, as well as the number of physical files that make up that document (ASCII, word processor format, TIFF page images). Then the user would fill in path information for each file or use a file browser type mechanism to identify files. Finally, bibliographic header information would be supplied by the user and the entire set of files for the document submitted. The system would accept the files and store them in a staging area for later review by the custodian. Future WWW browsers are expected to provide simpler file upload mechanisms. The LSSTB will automatically support the new capabilities.

3.2.2 Custodian Functions

Several custodial functions are needed to facilitate routine document management.

3.2.2.1 Submitting Documents

Documents can be submitted using several techniques. Current methods only support submitting documents manually on site. Software is needed which will allow the efficient extraction of documents from the CDOCS database. This work would involve building an interface to allow using http software (WWW browser) to identify CDOCS documents and associated files and copy these into the local insertion directory. These documents will be automatically inserted into the database at off-peak hours.

3.2.2.2 Delete Documents

Currently, documents which need to be removed or replaced must be manually deleted from the database. Scripts or on-site http client capabilities can be developed to provide a more automated document deletion mechanism. Document reindexing can be done automatically each night as required to allow database changes to become effective. Alternately, the LSSTB system administrator can choose to rebuild the database immediately.

3.2.3 Maintenance Functions

A number of functions are needed to facilitate efficient database maintenance and provide for efficient notification of system status and user comments.

3.2.3.1 E-mail Responses

Simple E-mail capabilities have been set up for the LSSTB system administrator. The intended use of E-mail responses for the LSSTB is to provide a mechanism for the LSSTB system administrator to be notified of system problems. Currently, the system administrator is required to log into a special account to check for E-mail responses. A mechanism is needed to automatically notify the LSSTB server administrator if a technical problem occurs.

3.2.3.2 Feedback Responses

Feedback responses from users are currently being recorded and stored in a database. An elementary graphical statistical analysis of the feedback results can be viewed from approved hosts using an http browser. Currently, automated scripts reanalyze portions of the database at off-peak hours, but improvements are needed to provide more thorough statistical analysis, output reports, and automated daily notifications of new user responses.

3.2.3.3 Site Tracking Statistics and Summaries

Basic http site statistics can now be viewed from certain approved hosts using an http browser. Automated scripts reanalyze the database at off-peak hours. Functionality is needed to permanently archive the database. Log files would then be automatically appended to existing files and archived on local disk for potential future needs.

3.2.4 User Features

3.2.4.1 Highlight Search Terms

Current http clients provide limited search capabilities for locating requested search terms and words in the viewed document. The user typically initiates a browser-based local word search on the document they have received. A valuable improvement would have the http server locate and highlight (or provide hypertext anchors for) these terms and words prior to displaying the document for the user. This type of capability would require a "persistent connection" capability, which is not currently available in the LSSTB.

15/41

3.2.4.2 Search Engine Word List

A visual display of the word list from documents in the database which is used by the LSSTB search engine would be useful in assisting users in searching for documents. Options for displaying all or sections of the word list would most likely be required for this option to be manageable.

3.2.4.3 Licensing Support System Test Bed User Profile

The use of a persistent customer database for retaining information regarding document searches and personal notes can be developed. This user profile would be performed only at the user's request and security features would be invoked to limit access. This would allow a user to save and retain pertinent search information for later use or analysis.

3.2.5 Extending Database Information Classes

Database content other than simple documents needs to be expanded to support an eventual full LSS. The LSSTB could be modified to handle new classes of information with the established http user interface. A brief discussion of these potential new classes follows.

Maps can provide the user with a visual representation of the site and NRC issues. Maps do not have to be limited to geographical information, but any images or pictures could be indexed in the document database for viewing by the user.

Geographical Information Systems (GIS) capabilities could be added to assist the user in locating multidimensional database information. Database information (documents and technical data) concepts or physical relationships could be referenced by selecting items from a map.

Video tapes and other similar media forms relating to HLW issues or documents could be indexed in the database and provided to users for viewing.

All of the improvements discussed in this section would improve the utility and maintainability of the LSSTB for users and allow an evaluation of on-line concepts for the LSSTB as the system evolves toward the actual LSS.

16/47

APPENDIX A

CODE MODULE DESCRIPTION

17/47

CODE MODULE DESCRIPTION

Custom code was needed for the LSSTB to construct and maintain the document database. The modules were written in the PYTHON scripting language for compatibility with Internet servers. Appendix A provides a brief technical description of each of the modules. Appendix B contains the code listing for these modules.

```
Module mvdoc   Move an LSS document from one location to another

SYNOPSIS
--------
# Classes
class MvDoc

DESCRIPTION
-----------
Author      : Kent Polk - SWRI Div. 17
Date        : 17-Sep 96

SEE ALSO
--------
os
shutil
string

---------------------------------------------------------------
Class MvDoc - Move an LSS document from one location to another

SYNOPSIS
--------
import mvdoc

class MvDoc
  def __call__ (self, arg, srcdoc, flist)
  def __init__(self,  collecn)
  def update(self)

DESCRIPTION
-----------
Create a new object for each collection.
Calling the object as a function:
 obj (args, dir, flist)
moves the directory. arg is not currently used, but included
to be compatible with os.path.walk().

Object update must be performed after all class actions
have been performed for a collection.
```


8/47

```
Module mkcoldb - Make a collection database for the Topic Internet Server.

SYNOPSIS
--------
# Classes
class HTMLDir
class InsDir
class Rebuild

# Functions
def getfile(src)

DESCRIPTION
-----------
Walk a collection tree performing the following actions
     - Generate index.html documents for a collection.
     - Generate a bulk insert file for a collection.
     - Reindex the TIS collection.

Author     : Kent Polk          - SwRI Div. 17
             David Lincoln      - SwRI Div. 09
Date       : 17-Sep-96

SEE ALSO
--------
os
regsub
shutil
stat
string
util

------------------------------------------------------------------------
Class HTMLDir - Build an index.html file for the specified subdirectory.

SYNOPSIS
--------
import mkcoldb

class HTMLDir
  def __call__ (self, args, dir, flist)
  def icon (self, ext)
  def info (self, name)

DESCRIPTION
-----------
Create a new object for each collection.
Calling the object as a function:
 obj (args, dir, flist)
indexes the directory. arg is not currently used, but included
to be compatible with os.path.walk().

------------------------------------------------------------------------
Class InsDir - Build a database collection insertion object for mkvdk.

SYNOPSIS
--------
import mkcoldb

class InsDir
  def __call__ (self, args, dir, flist)
  def __init__ (self, collect, file)
  def reset (self)
  def set_header (self, name)
  def set_msword (self, name)
  def set_pdf (self, name)
  def set_tif (self, name)
  def set_txt (self, name)
  def set_wp (self, name)

DESCRIPTION
-----------
Create a new object for each collection
Calling the object as a function:
 obj (args, dir, flist)
indexes the directory. arg is not currently used, but included
to be compatible with os.path.walk().

------------------------------------------------------------------------
Class Rebuild - Rebuild a database collection.

SYNOPSIS
--------
import mkcoldb

class Rebuild
  def __call__(self, insFile)
  def __init__(self, collection, baseDir, startDir, vdkDir)
  def create(self)
  def reset(self)

DESCRIPTION
-----------
Object creation deletes the old collection and creates a new one.
Calling the object as a function inserts the documents referenced
in the bulk insert file.
```

```
Module squery   Simple html form query handler for the Verity Topic Internet Ser
ver.

SYNOPSIS
--------
# Classes
class do_form

# Functions
def copyDict (dict)
def escape( s )

DESCRIPTION
-----------
Fold multipart forms into one query string, send to TIS and reformat the
results. Use as a cgibin or via the PyApache module.

Author     : Kent Polk - SWRI Div. 17
Date       : 14-Aug-96

SEE ALSO
--------
cgi
os
regsub
string
urllib

------------------------------------------------------------
Class do_form   Process an LSSTB search query form.

SYNOPSIS
--------
import squery

class do_form
  def __init__(self, vdkcgi)
  def extract (self, mylist, page, rnge)
  def get_query(self)
  def send(self, cgi, qstr)

DESCRIPTION
-----------
Fold multipart forms into one query string for the TIS.
Currently used for both simple and complex forms.
Presents a form containing the folded query string.
Keeps track of which form called it and makes that form available.
Processes the TIS results into pages, etc.
```

```
Module feedb - Simple html form query handler managing the LSSTB Feedback databa
se.

SYNOPSIS
--------
# Functions
def copyDict (dict)
def getfeedback()

DESCRIPTION
-----------
Simple html form query handler managing the LSSTB Feedback database.

Stores the form query dictionary and certain environment variables
into a persistent dictionary. Uses file locking for access and keeps
a separate count key in the dictionary for appending new response
dictionaries.

The 'wholist' must be satisfied or the feedback response is rejected.

Author      : Kent Polk - SwRI Div. 17
Date        : 19-Aug-96

SEE ALSO
--------
cgi
os
regsub
shelve
string
time
urllib
```

```
Module ccdts   Move ccdts type documents from CDOCS database to LSSTB

SYNOPSIS
--------
# Classes
class ExtractDoc

# Functions
def deepcopy(x, memo = None)
def parsedoc(file, tokregx)
def redate(date)
def subd_extract_docs(src, copy, dict)

DESCRIPTION
-----------
Author     : Kent Polk          - SWRI Div. 17
             David Lincoln      - SWRI Div. 09
Date       : 17-Sep-96

SEE ALSO
--------
mkcoldb
os
regsub
string

-------------------------------------------------------------------
Class ExtractDoc - Extract pertinent info from an NRC '*hw' document.

SYNOPSIS
--------
import ccdts

class ExtractDoc
  def __getitem__(self, key)
  def __init__(self, matchDict)
  def __len__(self)
  def parse(self, filename)

DESCRIPTION
-----------
Only one ExtractDoc per set of files is required. Call extract
to extract from each file.

Creates a list item for each key provided in the matchDict['order']
provided it exists also as a key. Each list item is prefixed by the
corresponding matchDict object (label). The list keyword is header.

If there is a -TEXT- key in the document dictionary, a self.Dict
object is created for key txt.

use getitem (doc[key]) to retreive a dictionary item.

-------------------------------------------------------------------
Function parsedoc - Parse a text document and return as a dictionary.

SYNOPSIS
--------
import ccdts

def parsedoc(file, tokregx)

DESCRIPTION
---------- -
Assumes the document is of the form:
 -TOKEN1
items for token1
 -TOKEN2
items for token2

tokregx is a compiled regular expression used to locate the tokens
Doesn't assume an  END  token

-------------------------------------------------------------------
Function subd_extract_docs   Walk a subdir, extracting header and txt fileinfo

SYNOPSIS
--------
import ccdts

def subd_extract_docs(src, copy, dict)

DESCRIPTION
-----------
Read from parent dir src, place *.header and *.txt files into
a parallel copy/ dir.
```

```
Module csp   Move csp type documents from CDOCS database to LSSTB

SYNOPSIS
--------
# Classes
class ExtractDoc

# Functions
def deepcopy(x, memo = None)
def parsedoc(file, tokregx)
def redate(date)
def subd_extract_docs(names, root, skip, destdir, dict)

DESCRIPTION
-----------
Author     : Kent Polk          - SwRI Div. 17
             David Lincoln      - SwRI Div. 09
Date       : 17-Sep-96

SEE ALSO
--------
mkcoldb
os
regsub
shutil
string

------------------------------------------------------------------------
Class ExtractDoc - Extract pertinent info from a csp-type document

SYNOPSIS
--------
import csp

class ExtractDoc
  def __getitem__(self, key)
  def __init__(self, matchDict)
  def __len__(self)
  def parse(self, filename)

DESCRIPTION
-----------
Only one ExtractDoc per set of files is required. Call extract
to extract from each file.

Creates a list item for each key provided in the matchDict['order']
provided it exists also as a key. Each list item is prefixed by the
corresponding matchDict object (label). The list keyword is header.

use getitem (doc[key]) to retreive a dictionary item.

------------------------------------------------------------------------
Function parsedoc - Parse a text document and return as a dictionary.

SYNOPSIS
--------
import csp

def parsedoc(file, tokregx)

DESCRIPTION
-----------
Assumes the document is of the form.
TOKEN1: item for token1
TOKEN2: item for token2

tokregx is a compiled regular expression used to locate the tokens

---------- ---------- ---------- ---------- ----------
Function subd_extract_docs   Extracting header and txt fileinfo.

SYNOPSIS
--------
import csp

def subd_extract_docs(names, root, skip, destdir, dict)

DESCRIPTION
-----------
Read from parent dir src, place *.header and *.txt files into
a parallel copy/ dir.
```

APPENDIX B

CODE MODULE LISTINGS

25/47

CODE MODULE LISTINGS

B.1 INSERT.PY

B.2 MVDOC.PY

B.3 MKCOLDB.PY

B.4 SQUERY.

B.5 FEEDB.PY

B.6 CCDTS.PY

B.7 CSP.PY

B.1 INSERT.PY

```
#!/usr/local/bin/python
#-------------------------------------------------------------------------
# Title      : insert.py   Intended to be run as a script every night to process
new
#              documents submitted to the system.
#
# Purpose    : Walk collection trees performing the following actions
#              - Move LSS documents from insert to docs.
#              - Generate index.html documents for a collection.
#              - Generate a bulk insert file for a collection.
#              - Reindex the collection.
#              - Restart the TIS server.
#
# Author     : Kent Polk       - SwRI Div. 17
#              David Lincoln   - SwRI Div. 09
# Date       : 17-Sep-96
#-------------------------------------------------------------------------
import mkcoldb
import os
import csp
import ccdts
import mvdoc

coldir      = '/u04/nrcweb/database/collections/'
bulkdir     = '/u04/nrcweb/database/collections/bulkinsert/'
vdkdir      = '/u04/netscape/tis/_ssol23/bin/'
scriptsdir  = '/u04/nrcweb/scripts/'
absdocs     = '/u04/nrcweb/database/docs/'
docsdir     = 'docs'
insdir      = 'insert'

def rebuildCollection(list):
   returnValue = 0
   for coll in list:
      docscol = os.path.join(docsdir, coll)

      remakeFile = docscol+'.remake'
      insFile = '%s.ins' % coll

      # copy new documents from insert into docs, delete insert
      # files. Rename as required.
      insertcol = mvdoc.MvDoc(coll)
      os.path.walk(os.path.join(insdir, coll), insertcol, '')
      insertcol.update()

      # Is there anything to remake?
      if os.path.exists(remakeFile) :

         # rebuild the html indices
         html = mkcoldb.HTMLDir()
         os.path.walk(docscol, html, '')

         # build the bulkinsert file
         file = open(bulkdir+insFile,'w')
         inscol = mkcoldb.InsDir(coll, file)
         os.path.walk(docscol, inscol, '')
         file.close()

         # reindex the collection
         collection = mkcoldb.Rebuild(coll, absdocs, coldir, vdkdir)
         collection(bulkdir+insFile)

         os.unlink(remakeFile)

         returnValue = 1
   return(returnValue)

if __name__ == '__main__' :

   if rebuildCollection(['ccdts', 'csp', 'nuregs', 'regs', 'reports']):
      # restart the TIS
      os.system(os.path.join(scriptsdir, 'startvdk'))
```

B.2 MVDOC.PY

```
#!/usr/local/bin/python
#
# Title      : mvdoc.py
# Purpose    :
"""Move an LSS document from one location to another

Author     : Kent Polk - SwRI Div. 17
Date       : 17 Sep 96
"""
#
import os
import sys
import string
import shutil

file_ext = ['.txt', '.text', '.pdf', '.wp', '.doc', '.header']
icon_ext = ['.gif', '.iff', '.jpg', '.jpeg', '.png', '.pgm', '.ppm', '.ras', '.r
ast', '.rgb', '.tif', '.tiff', '.xbm']

base_todir = '/u04/nrcweb/database/docs/'

#-----------------------------------------------------
# Move from this dir structure :
#                  insert
#                 /  |   |  \
#         reports regs csp auregs
#         /  |  \
#   analysis 0001 0002 ...1023
#
# To this dir structure :
#                  docs
#                 /  |   |  \
#         reports
#        /   |   \         / This dir level will contain a
#       01   02   03 ... <  collection 'count' and 'xx.remake'
#     / | \                \ file.
# 0000 0001 0002 ...1023
#
#  Move from dir to collection
#  For example, from insert/reports/analysis to "reports" (calculate /00/0000)
#
```

B.2-1

```
class MVDoc:
    """Move an LSS document from one location to another

    Create a new object for each collection
    Calling the object as a function:
     obj (args, dir, flist)
    moves the directory. arg is not currently used, but included
    to be compatible with os.path.walk().

    Object update *must* be performed after all class actions
    have been performed for a collection.
    """
    # Interface -- initialize and reset this instance
    # collecn is the collection name
    def __init__(self, collecn):
        self.count   = 0
        self.collecn= collecn
        self.srcdir = os.path.join(base_todir, collecn)
        self.cfile  = os.path.join(self.srcdir,'count')
        if os.path.exists(self.cfile):
            cfile    = open(self.cfile, 'r')
            self.count = string.atoi(cfile.read()[:6])
            cfile.close
        self.oldcount  = self.count
        print collecn, self.srcdir, self.count

    def update(self):
        cfile = open(self.cfile, 'w')
        cfile.write('%6.6d' % self.count)
        cfile.close
        if self.count > self.oldcount :
            print (os.path.join(self.srcdir,self.collecn+'.remake'))
            cfile = open(os.path.join(base_todir,self.collecn+'.remake'), 'w')
            cfile.write('please remake')
            cfile.close

    # This is the entry point
    # srcdoc is the source 'document'
    # flist is the list of files in the 'document'
    # arg is unused at this time
    def __call__ (self, arg, srcdoc, flist):

        # don't try to copy files in the collection directory
        if os.path.basename(srcdoc) == self.collecn:
            return

        document = '%6.6d' % self.count

        # read the docs/collectn/count file
        # figure out what docs/collectn/coldir to put it in.
        # mkdir() the docs/collectn/coldir/(count%coldir)
        colsubd = os.path.join(self.srcdir, '%2.2d' % (divmod(self.count,1024)[0]

        coldir = os.path.join(colsubd, document)
        if not os.path.exists(colsubd):
            os.mkdir(colsubd, 0755)
        if not os.path.exists(coldir):
            os.mkdir(coldir, 0755)
```

29/47

```
print 'copying %s to %s' % (srcdoc,coldir)

flist.sort()
for fname in flist:
   absname = os.path.join(srcdoc,fname)
   if not os.path.isdir(absname):
      ext = os.path.splitext(fname)[1]
      if ext in file_ext:
         namejoin = document+ext
         newname = os.path.join(coldir, namejoin)
      else:
         newname = os.path.join(coldir, fname)
      if not ext == '.html' :
         print 'copy from %s to %s' % (absname, newname)
         shutil.copy2(absname, newname)
      os.unlink(absname)

os.rmdir(srcdoc)
self.count = self.count +1
```



```
coldir      = '/u04/nrcweb/database/collections/'
absdocs     = '/u04/nrcweb/database/docs/'
docsdir     = 'docs'

# This object can be executed as a script, as an example only
# You must be at the root database directory and include the name of
# the collection for this script to work.
#
# Typical usage of this module.
if __name__ == '__main__' :

   # Please provide the collection name
   if len(sys.argv) > 1 :
      # make a few names
      coll = sys.argv[1]
      docscol = os.path.join(docsdir, coll)
      insFile = '%s.ins' % coll

      insertcol = MyDoc(coll)
      os.path.walk(os.path.join(insdir, coll), insertcol, '')
      insertcol.update()
```

B.3 MDCOLDB.PY

```
#!/usr/local/bin/python
#
# Title     : mkcoldb.py
"""Make a collection database for the Topic Internet Server.

Walk a collection tree performing the following actions
      Generate index.html documents for a collection.
      Generate a bulk insert file for a collection.
      Reindex the TIS collection.

Author    : Kent Polk          - SwRI Div. 17
            David Lincoln      - SwRI Div. 09
Date      : 17-Sep-96
"""
#-------------------------------------------------------------------------
import os
import sys
import string
import time
import util
import shutil
import regsub
import regex
import stat

#-----------------------------------------------------
def getfile(src):
    """Get a file and return as a list of strings.
    """
    file=open(src,'r')
    lines = file.readlines()
    file.close()
    return lines

# This is the literal HTML document
HTML_PAGE = """\
<HTML><HEAD>
<TITLE>Original Source Documents</TITLE>
</HEAD><BODY>
<H1>Original Source Documents for: </H1><P>
<PRE>
%(header)s</PRE>
<HR><DL>
%(results)s
</DL><HR>
<A HREF=\"/intro.map\"><IMG ISMAP SRC=\"/images/introbar.gif\"></A><BR>
</BODY></HTML>
"""

ext_pdf  = ['.pdf']
ext_wp   = ['.wp', '.doc']
ext_ps   = ['.ps', '.eps']
ext_txt  = ['.txt', '.text']
ext_bin  = ['.bil', '.bip', '.tar', '.Z']
ext_pic  = ['.gif', '.iff', '.jpg', '.png', '.pgm', '.ppm', '.ras', '.rast', '.r
gb', '.tif', '.tiff', '.xbm']
excepts  = ['.txt', '.pdf', '.html', '.header', '.count', '.remake']
```

B.3-1

```
#-------------------------------------------------------------------------
class HTMLDir:
    """Build an index.html file for the specified subdirectory

    Create a new object for each collection.
    Calling the object as a function:
     obj (args, dir, flist)
    indexes the directory. arg is not currently used, but included
    to be compatible with os.path.walk().
    """
    def info (self, name):
        st    = os.stat(name)
        ctime = string.split(time.ctime(st[stat.ST_ATIME]))
        return "(%d bytes) %2s %2s %4s" % (st[stat.ST_SIZE], ctime[2], ctime[1], c
time[4])

    def icon (self, ext):
        icon = 'text.gif'
        if   ext in ext_txt  : icon = 'text.gif'
        elif ext in ext_wp   : icon = 'layout.gif'
        elif ext in ext_pdf  : icon = 'spdf.gif'
        elif ext in ext_pic  : icon = 'image2.gif'
        elif ext in ext_bin  : icon = 'binary.gif'
        elif ext in ext_ps   : icon = 'ps.gif'
        return '/icons/'+icon

    # This is the entry point
    def __call__ (self, args, dir, flist):
        createindex = 0
        results     = []
        names       = { }

        # get the contents of the header file
        item = os.path.basename(dir)+'.header'
        if item in flist :
            names['header'] = (util.readfile(os.path.join(dir,item)))
            flist.remove(item)
            createindex = 1
        else: names['header'] = ''

        flist.sort()
        for fname in flist:
            absname = os.path.join(dir,fname)
            if not os.path.isdir(absname):
                ext = os.path.splitext(fname)[1]
                if ext not in excepts:
                    results.append ("<DD> <IMG SRC=\"%s\"> <A HREF=\"%s\">%s</A> %s\
                        % (self.icon(ext), fname, fname, self.info(absname)))
                    createindex = 1

        if createindex :
            names['results'] = string.joinfields(results,'\n')
            file=open(os.path.join(dir,'index.html'),'w')
            file.writelines([HTML_PAGE % names])
            file.close
            print 'indexing: ',dir # where am I?
```

```
#-------------------------------------------------------
# Class object for building a collection insertion object for mkvdk

titlelen = 35

INSDATA = """\
%(hdr)s
VDKVGWKEY:\t%(gw)s
DOC_DIR:\t\t%(dir)s
DOC_FN:\t\t./%(name)s
DOC_SZ:\t\t%(size)s
DOC_DATE:\t%(date)s
TXT_ANCH:\t%(ta)s
HDR_ANCH:\t%(ha)s
ORG_ANCH:\t%(oa)s
PDF_ANCH:\t%(pa)s
<<EOD>>
"""
```



```
#-------------------------------------------------------
class InsDir:
    """Build a database collection insertion object for mkvdk

    Create a new object for each collection.
    Calling the object as a function:
     obj (args, dir, flist)
    indexes the directory. arg is not currently used, but included
    to be compatible with os.path.walk().
    """

#   Example:
#   --------
#   TITLE: 10 CFR 60
#   AUTHOR: NRC
#   DATE: 1-Jan 1993
#   PUBLISHER: NRC
#
#   VDKVGWKEY:   /reqs/00/000000/000000.txt
#   DOC_DIR:     /reqs/00/000000
#   DOC_FN:      ./reqs/00/000000/000000.txt
#   DOC_SZ:      257388
#   DOC_DATE:    16 Sep 1996
#   TXT_ANCH:    <A HREF="/reqs/00/000000/000000.txt">10 CFR 60</A>
#   HDR_ANCH:    <A HREF="/reqs/00/000000/000000.header"><IMG SRC="/icons/header.g
if" ALT="hdr "></A>
#   ORG_ANCH:    <A HREF="/reqs/00/000000"><IMG SRC="/icons/comp.gray.gif" ALT="st
c "></A>
#   PDF_ANCH:    <IMG SRC="/icons/bblank.gif" ALT="    ">
#   <<EOD>>
#   --------
#   Note that entries prior to 'VDKVGWKEY' are read directly from the *.header
#   file 'as is'.

    # Initialize instance for this collection
    def __init__(self, collecn, file):
        self.collecn    = collecn
        self.file       = file

    def reset (self):
        # This is the list where info is built.
        self.doc        = []

        # These go in the list...
        self.servdir    = ""
        self.name       = ""
        self.size       = ""
        self.date       = ""
        self.title      = ""
        self.gateway    = ""
        self.txt_anchor = ""
        self.hdr_anchor = "<IMG SRC \"/icons/bblank.gif\" ALT=\"        \">"
        self.org_anchor = "<IMG SRC=\"/icons/bblank.gif\" ALT=\"        \">"
        self.pdf_anchor = "<IMG SRC=\"/icons/bblank.gif\" ALT=\"    \">"

    def set_txt (self, name):
        # Set the ascii text version document information
        st = os.stat(name)
```

```
      self.size = str(st[stat.ST_SIZE])
      ctime = string.split(time.ctime(st[stat.ST_ATIME]))
      self.date = "%2s %2s %4s" % (ctime[2], ctime[1], ctime[4])
      pad = titlelen - len(self.title)
      self.txt_anchor="<A HREF=\"%s\">%s</A>%s" % (self.gateway, self.title, str
ing.ljust(" ", pad))

   def set_header (self, name):
      hdr_name = os.path.join(self.servdir, os.path.basename(name))
      self.hdr_anchor="<A HREF=\"%s\"><IMG SRC=\"/icons/header.gif\" ALT=\"hdr \
"></A>" % hdr_name

      # read the header file as a list of strings and extract the title
      temp = getfile(name)
      for line in temp:
         if string.find(string.lower(line), 'title') > -1 :
            self.title = string.strip(string.splitfields(line, ":")[1])
            if len(self.title) > titlelen :
               self.title = self.title[:(titlelen-3)]+'...'
            break
      return temp

   def set_tif (self, name):
      # Set the Original source document information since I found a tiff file..
.
      # Hack. :^(
      org_name = self.servdir
      self.org_anchor="<A HREF=\"%s\"><IMG SRC=\"/icons/comp.gray.gif\" ALT=\"sr
c \"></A>" % org_name

   def set_wp (self, name):
      # Set the WordPerfect Original source document information
      org_name = self.servdir
      self.org_anchor="<A HREF=\"%s\"><IMG SRC=\"/icons/comp.gray.gif\" ALT=\"sr
c \"></A>" % org_name

   def set_msword (self, name):
      # Set the MSWord Original source document information
      org_name = self.servdir
      self.org_anchor="<A HREF=\"%s\"><IMG SRC=\"/icons/comp.gray.gif\" ALT=\"sr
c \"></A>" % org_name

   def set_pdf (self, name):
      # Set the PDF version document information
      pdf_name = os.path.join(self.servdir, os.path.basename(name))
      self.pdf_anchor="<A HREF=\"%s\"><IMG SRC=\"/icons/spdf.gif\" ALT=\"pdf \">
</A>" % pdf_name

   # This is the entry point
   def __call__ (self, args, dir, flist):
      self.reset()
      writeindex = 0

      # Create the dirname in the form: "analysis"
      bname = os.path.basename(dir)
      # Create the dirname in the form: "./analysis/"
      reldir  = dir + "/"
      # Create the dirname in the form: "/docs/analysis/"
      self.servdir  = '/'+os.path.join(self.collect, os.path.join(
                      os.path.basename(os.path.split(dir)[0]), bname))

      fname = bname+".header"
      if fname in flist:
         # Send the name as : "./analysis/analysis.header"
         self.doc = self.set_header(reldir+fname)

      fname = bname+".txt"
      if fname in flist:
         # Create the filename in the form: "/reports/00/000000/000000.txt"
         self.name = os.path.join(self.servdir, fname)
         self.gateway = os.path.join(self.servdir, os.path.basename(self.name))
         self.set_txt(reldir+fname)
         # We have to have a textfile to have a document
         writeindex = 1

      fname = bname+".wp"
      if fname in flist:
         self.set_wp(reldir+fname)

      fname = bname+".doc"
      if fname in flist:
         self.set_msword(reldir+fname)

      fname = bname+".pdf"
      if fname in flist:
         self.set_pdf(reldir+fname)

      fname = bname+".tif"
      if fname in flist:
         self.set_tif(reldir+fname)

      if writeindex :
         print 'indexing: ',dir # where am i?
         self.file.writelines([INSDATA % {
                        'hdr': string.joinfields(self.doc,''),
                        'gw': self.gateway,
                        'dir': self.servdir,
                        'name': self.name,
                        'size': self.size,
                        'date': self.date,
                        'ta': self.txt_anchor,
                        'ha': self.hdr_anchor,
                        'oa': self.org_anchor,
                        'pa': self.pdf_anchor,
                        }])
```

```
#-------------------------------------------------------
class Rebuild:
   """Rebuild a database collection.

   Object creation deletes the old collection and creates a new one
   Calling the object as a function inserts the documents referenced
   in the bulk insert file.

   """
   # Interface -- initialize and reset this instance
   def __init__(self, collection, baseDir, startDir, vdkDir):
      self.collection = collection
      self.startDir = startDir
      self.colldir = os.path.join(self.startDir, self.collection)
      self.styledir = os.path.join(self.colldir, 'style')
      self.sourcedir = baseDir
      self.mkvdk = os.path.join(vdkDir, 'mkvdk')
      self.currdir = os.getcwd()
      self.reset()
      self.create()

   # Reset the collection for rebuilding
   def reset(self):
      os.chdir(self.colldir)
      if os.path.exists('collectn.lck'):
         os.unlink('collectn.lck')
      os.system('rm -r assists morgue parts pdd temp topicidx trans work')
      os.chdir(self.currdir)

   # Create the collection for document insertion
   def create(self):
      os.system('%s -create -synch -style %s -collection %s' % \
         (self.mkvdk, self.styledir, self.colldir))

   # Insert all documents for a collection into the collection
   def __call__(self, insFile):
      os.system('%s -nohousekeep -collection %s -datapath %s -bulk %s' % \
         (self.mkvdk, self.colldir, self.sourcedir, insFile))
```

B.3-4

```
coldir      = '/u04/hrcweb/database/collections/'
bulkdir     = '/u04/hrcweb/database/collections/bulkinsert/'
vdkdir      = '/u04/netscape/fts/ssol24/bin/'
absdocs     = '/u04/hrcweb/database/docs/'
docsdir     = 'docs'
insdir      = 'insert'

# This set of objects can be executed as a script, as an example only
# You must be at the root database directory and include the name of
# the collection for this script to work.
#
# Typical usage of this module:
if __name__ == '__main__' :

   # Please provide the collection name
   if len(sys.argv) > 1 :
      # make a few names
      coll = sys.argv[1]
      docscol = os.path.join(docsdir, coll)
      insFile = '%s.ins' % coll

      # Build index.html files for this collection
      html = mkcoldb.HTMLDir()
      os.path.walk(docscol, html, '')

      # Build the insert file for this collection
      file = open(bulkdir+insFile,'w')
      insdir = InsDir(coll, file)
      os.path.walk(docscol, insdir, '')
      file.close()

      # Rebuild (reindex) the collection database
      collection = mkcoldb.Rebuild(coll, absdocs, coldir, vdkdir)
      collection(bulkdir+insFile)
```

B.4 SQUERY.

```
#!/usr/local/bin/python
#
# Title      : squery
"""Simple html form query handler for the Verity Topic Internet Server.

Fold multipart forms into one query string, send to TIS and reformat the
results. Use as a cgibin or via the PyApache module.

Author     : Kent Polk - SwRI Div. 17
Date       : 14-Aug 96
"""
#---------------------------------------------------------------------------
import os
import sys
import string
import cgi
import regsub
import urllib

HTMLPAGE = """\
Content-type: text/html

<HTML><HEAD>
<TITLE>Search Results</TITLE>
</HEAD><BODY>

<FORM METHOD-POST ACTION-"/cgi/squery.py">
You may modify the query line here or return to the <A HREF-%(searchee)s>Search<
/A> page for query
modification.<BR>
<INPUT NAME-"qparser" TYPE-"hidden" VALUE-"simple">
<INPUT NAME-"searchr" TYPE-"hidden"  VALUE-"%(searcher)s">
<INPUT NAME-"query"   SIZE-60 value - "%(query)s"><INPUT TYPE-"submit" VALUE-" S
earch "> <BR>
</FORM>

<H1>Search Results</H1>
%(results)s
%(page)s
<HR><A HREF-"/intro.map"><IMG ISMAP SRC-"/images/introbar.gif"></A>
</BODY></HTML>
"""

NEWPAGE = """<FORM METHOD-POST ACTION-"/cgi/squery.py">
<INPUT TYPE-"hidden" NAME-"qparser" VALUE-"simple">
<INPUT TYPE-"hidden" NAME-"query" VALUE - "%(query)s">
<INPUT TYPE-"hidden" NAME-"pgnum" VALUE-"%(pgnum)d">
<INPUT TYPE-"submit" VALUE-" %(which)s (page %(pgnum)d of %(pages)d) ">
</FORM>
"""

SSEARCHEE - "/search.html"
ASEARCHEE - "/search_as.html"

LINESPP  - 10

querykeys- ['qop','query','queryThresh','qparser','dbname','sortSpec','exitURL']

def escape( s ):
   s - regsub.gsub('"', '&quot;', s)
   s - regsub.gsub('<', '&lt;', s)
   s - regsub.gsub('>', '&gt;', s)
   return s

def copyDict (dict) :
   newdict - {}
   for key in dict.keys() :
      newdict[key] - dict[key]
   return newdict

class do_form:
   """Process an LSSTB search query form.

   Fold multipart forms into one query string for the TIS.
   Currently used for both simple and complex forms
   Presents a form containing the folded query string.
   Keeps track of which form called it and makes that form available.
   Processes the TIS results into pages, etc.
   """

   def __init__(self, vdkcgi):
      self.page      - 1
      self.searcher  - 'simple'
      self.searchee  - SSEARCHEE
      self.queryform - ''
      self.nextform  - ''

      # send the query string to vdkcgi and get the results pipe handle (popen)
      pipe - self.send(vdkcgi, self.get_query())

      # read the pipe handle, reformat and print the results
      # Note that order is important... extract() has to called before
      # the queryform is filled out.
      pndx - (self.page  1) * LINESPP
      sys.stdout.writelines([(HTMLPAGE % {
         'results' : self.extract(pipe.readlines(), self.page, range(pndx, pndx+
LINESPP)),
         'searchee': self.searchee,
         'searcher': self.searcher,
         'query'   : self.queryform,
         'page'    : self.nextform})])
      pipe.close()

   def get_query(self):

      # Go get the form pairs from stdin and put them in the 'form' dictionary
      # so we can build 'QUERY STRING'
      formDict - cgi.FormContentDict()
      cpDict   - copyDict (formDict)

      key - 'pgnum'
      if cpDict.has_key(key) and len(cpDict[key][0]) > 0 :
         self.page - string.atoi(cpDict[key][0])
         del cpDict[key]

      key - 'searchr'
      if cpDict.has_key(key) and len(cpDict[key][0]) > 0 :
         self.searcher - cpDict[key][0]
         if self.searcher in ['advanced'] :
            self.searchee - ASEARCHEE
```

B.4-1

37/47

```
        del cpDict[key]

    # Find if the user selected a logical operator
    op    = ''
    qop   = ' <or> '
    key   = 'qop'
    if cpDict.has_key(key) and len(cpDict[key][0]) > 0 :
        qop = ' <%s> ' % cpDict[key][0]
        del cpDict[key]

    # Build the 'query' item string - start with the initial 'query' item
    # If there is a 'query' item, we need to set up to add the operator
    query = ''
    key   = 'query'
    if cpDict.has_key(key) and len(cpDict[key][0]) > 0 :
        query = string.strip(cpDict[key][0])
        if (query) > 0 :
            op = qop

    # fold extraneous form pairs into the 'query' item, delete the dict item
    for key in cpDict.keys() :
        if key not in querykeys :
            query = "%s%s(%s <contains> %s)" % (query, op, key, cpDict[key][0])
            op    = qop
            del cpDict[key]

    # reassign the new 'query' item
    cpDict['query'] = [query]
    # Mosaic, etc. need *only* the '>' changed to &gt,
    # otherwise they try to parse the form value...
    self.queryform = escape(query)

    # build the QUERY_STRING from the remaining form pairs
    qlist = []
    for key in cpDict.keys() :
        qlist.append ("%s=%s" % (key, cpDict[key][0]))

    # build the 'QUERY_STRING' from the form pairs, convert spaces to '+' and
    # url quote it.
    return urllib.quote(regsub.gsub(' ','+', ('\"%s\"' \
                        % (string.joinfields(qlist, "&")))),'"=+&')

def send(self, cgi, qstr):
    pipe= os.popen("QUERY_STRING=%s REQUEST_METHOD=GET CONTENT_LENGTH=0 \
            SCRIPT_NAME=/cgi/%s %s" % (qstr, cgi, cgi), 'r')
    return pipe

def extract (self, mylist, page, rnge):
    count    = 0
    capture  = -3
    results  = ""
    cntline = []

    for line in mylist:
        # Have to extract the "# of #" line from the results (add '<HR>')
        if string.find (line, 'shown') > -1 :
            cntline = string.split(line)

            # See how many results pages we have to display, and build
            # the 'next page' form button if required.
            pages, temp = divmod (string.atoi(cntline[6][1:]), LINESPP)
            if temp > 0 :
                pages = pages + 1
            if pages > 1 and page < pages :
                self.nextform = (NEWPAGE % {
                    'query'   : self.queryform
                    'which'   : "Next",
                    'pages'   : pages,
                    'pgnum'   : (page + 1) })
            continue

        # Now get the results list and convert to a string.
        # Start capturing results.  -2 means we have to extract t
        # first two lines regardless of what page we are displayi
        if string.find (line, '<PRE') > -1 :
            capture = -2

        # all done, exit
        if string.find (line, '</PRE') > -1 :
            results  = results + line
            break

        if capture in [ 2,  1] + rnge :
            if capture in rnge :
                count = count + 1
            results  = results + line

        if capture > -3 :
            capture  = capture + 1

    # how many results entries did we actually get on this page?
    if len (cntline) > 6 :
        cntline[6] = "(%d " % count
    else:
        sys.stderr.write(self.queryform)
    return string.join(cntline) + "<HR>\n" + results

if __name__ == '__main__' :

    dump = do_form("./reports@9080")
```

B.5 FEEDB.PY

```
#!/usr/local/bin/python
#-----------------------------------------------------------------------
# Title      : feedb.py
"""Simple html form query handler managing the LSSTB Feedback database

Author      : Kent Polk - SwRI Div. 17
Date        : 19 Aug 96
"""
#-----------------------------------------------------------------------
import   os
import   sys
import   string
import   cgi
import   shelve
import   time

RESPONSE = """\
Content-type: text/html

<HTML><HEAD>
<TITLE>NRC Licensing Support System Test Bed (LSSTB) Feedback</TITLE>
</HEAD><BODY>
<CENTER>
<IMG ALIGN=LEFT SRC="/images/NRC.gif" ALT="">
<H2>NRC </H1>
<H3>Licensing Support System Test Bed</H3>
<H3>(LSSTB)<BR CLEAR=LEFT></H3>
</CENTER>
<HR>
<P>
%(response)s
<HR>
<A HREF="/intro.map"><IMG BORDER=0 ISMAP SRC="/images/introbar.gif"></A>
</BODY></HTML>
"""

NOWHO = """\
Sorry...<BR>
You must provide a name, organization and phone number for your response
to be recorded.
<P>
"""

THANKS = """\
%(name)s, thank you for responding to this survey (response #%(num)s).
<P>
"""

dbfilename  = 'feedback/responses'
lockfilename= '/tmp/lsstbresp.lock'

sellist = ['overall', 'useful', 'access', 'ease', 'email']
wholist = ['name', 'company', 'phone']

def copyDict (dict) :
   newdict = {}
   for key in dict.keys() :
      newdict[key] = dict[key]
   return newdict

def getfeedback() :
   """Read and process the LSSTB response feedback

   Stores the form query dictionary and certain environment variables
   into a persistent dictionary. Uses file locking for access and keeps
   a separate count key in the dictionary for appending new response
   dictionaries.

   The 'wholist' must be satisfied or the feedback response is rejected.
   """
   error = 0
   getDict = cgi.FormContentDict()
   formDict = copyDict (getDict)

   # Add a few items to the database dictionary
   key = 'HTTP_USER_AGENT'
   if key in os.environ.keys() :
      formDict['browser'] = [os.environ[key]]
   key = 'REMOTE_HOST'
   if key in os.environ.keys() :
      formDict['rhost'] = [os.environ[key]]
   formDict['time'] = [time.ctime(time.time())]

   # reject if wholist isn't complete
   for key in wholist :
      if not key in formDict.keys() :
         sys.stdout.writelines([(RESPONSE % {'response': NOWHO } )])
         error = 1
         break

#  file = open('log','w')
#  sys.stderr = file
#  file.write ('\n')
#  file.close

   if not error :

      # Wait till the lockfile becomes available
      while os.path.exists(lockfilename) :
         pass

      # Grab the lockfile
      lock = open(lockfilename,'w')

      # Open the database and find the current key
      db = shelve.open(dbfilename)
      if db.has_key('lastkey') :
         keynum = string.atoi(db['lastkey']) + 1
      else :
         keynum = 1 # Hmmm must not be a database - new one..
      key = "%5d" % keynum
      db['lastkey'] = key

      # Append the current entry and release the db/lock
      db[key] = formDict
      db.close()
      lock.close()
      os.unlink(lockfilename)

      sys.stdout.writelines([(RESPONSE % {
         'response': string.join([(THANKS % {
```

```
                'name': formDict['name'][0],
                'num':  string.strip(key),
                })])
            })])

if __name__ == '__main__' :
    getfeedback()
```

B.6 CCDTS.PY

```
#
#--------   ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
# Title      : ccdts.py
# Purpose    :
"""Move ccdts type documents from CDOCS database to LSSTB

Author     : Kent Polk          - SwRI Div. 17
             David Lincoln      - SwRI Div. 09
Date       : 17 Sep 96
"""
#--------   ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
import os
import sys
import string
import regsub
import regex
import mkcoldb

ccdtsDict = {
            'order': ['-TITLE-','-SIGN_NAME-','-ADDRESS_NAME-','-ISSUE_DT-','-IS
SUES-'],
            'regex': '^-[A-Z_]+-',
            'redate': '-ISSUE_DT-',
            '-TITLE-':'TITLE',
            '-SIGN_NAME-':'AUTHOR',
            '-ADDRESS_NAME-':'ADDRESSEE',
            '-ISSUE_DT-':'DATE',
            '-ISSUES-':'ISSUES',
            }

from rfc822 import _monthnames
def redate(date):
    """Change datestr from yyyymmdd to dd-mmm-yyyy
    """
    return "%s-%s-%s" % (date[6:8],_monthnames[string.atoi(date[4:6])-1],date[:4]
)

#--------------------------------------------------------
def parsedoc(file, tokregx):
    """Parse a text document and return as a dictionary.

    Assumes the document is of the form:
    -TOKEN1-
    items for token1
    -TOKEN2-
    items for token2

    tokregx is a compiled regular expression used to locate the tokens
    Doesn't assume an -END- token
    """
    # some documents don't start with a token so toklist has to
    # be created here.
    toklist = []
    tokDict = {}
    last_token = ""
    f = open(file, 'r')
    while 1:
        line = f.readline()
        if not line:
            # some document don't end with a token so we have to
            # make sure we clean up
            tokDict[last_token] = toklist
            break
        if tokregx.match(line) > 0:
            toklist.append(line)
        else:
            if len(last_token) > 0:
                tokDict[last_token] = toklist
            toklist = []
            last_token = regsub.gsub('\n','',line)
    return tokDict

from copy import deepcopy
class ExtractDoc:
    """Extract pertinent info from an NRC '*hw' document.

    Only one ExtractDoc per set of files is required. Call 'extract'
    to extract from each file.

    Creates a list item for each key provided in the matchDict['order']
    provided it exists also as a key. Each list item is prefixed by the
    corresponding matchDict object (label). The list keyword is 'header'.

    If there is a ' TEXT ' key in the document dictionary, a self.Dict
    object is created for key 'txt'.

    use getitem (doc[key]) to retreive a dictionary item
    """
    # Interface -- initialize and reset this instance

    def __init__(self, matchDict):
        # Create one object per document class.
        # Note that parse() resets for each new file provided.
        #
        #Compile the regular expression and set the header order list
        self.Dict    = {}
        self.docDict= {}
        self.matDict= deepcopy(matchDict)
        self.order   = []
        self.redate = ""

        # if redate exists, it must reference a key in the matchDict
        # that represents a date string in the form yyyymmdd so it can
        # convert the date to dd mmm yyyy format
        key = 'redate'
        if matchDict.has_key(key):
            self.redate = matchDict[key]

        # if order exists, it will be used to provide an ordered list
        # of keys which should be in matchDict for the header list
        # building operation. If order isn't provided, the header
        # list will be in an arbitrary order.
        key = 'order'
        if matchDict.has_key(key):
            self.order   = matchDict[key]

        # regex must exist    it is the regular expression used to
        # parse the document
        key = 'regex'
        if matchDict.has_key(key)
```

```
            self.regtok = regex.compile(matchDict[key])
        else : return None

    #def keys(self):
    #return self._index.keys()
    #
    #def has_key(self, key):
    #return self._index.has_key(key)

        def __len__(self):
            return len(self.Dict) +len(self.docDict) +len(self.matDict)

        def __getitem__(self, key):
            #Get an object associated with the specified key.
            #Search all internal class dictionaries.
            if self.Dict.has_key(key) :
                return self.Dict[key]
            elif self.docDict.has_key(key) :
                return self.docDict[key]
            elif self.matDict.has_key(key) :
                return self.matDict[key]
            else : return None

        def parse(self, filename):
            #Extract header and text lists from the parse dictionary of a document.

            self.docDict = parsedoc(filename, self.regtok)

            # Do we need to change the date format?
            # ought to provide a conversion function someday...
            key = self.redate
            if key in self.docDict.keys() :
                self.docDict[key] = [redate(self.docDict[key][0])]

            headerlist = []
            if len(self.order) : # build this list in the order provided
                for key in self.order :
                    if key in self.matDict.keys() and key in self.docDict.keys() :
                        headerlist.append(
                            self.matDict[key]+': '+\
                            regsub.gsub('\n','',string.join(self.docDict[key]))+'\n')
            else:                   # else this list will be unordered
                for key in self.docDict.keys() :
                    if key in self.matDict.keys() :
                        headerlist.append(
                            self.matDict[key]+': '+\
                            regsub.gsub('\n','',string.join(self.docDict[key]))+'\n')

            self.Dict['header'] = headerlist
            key = '-TEXT-'
            if self.docDict.has_key(key) :
                del self.docDict[key][0]    # remove the filename line
                self.Dict['txt'] = self.docDict[key]
            return 1

    def subd_extract_docs(src, copy, dict):
        """Walk a subdir, extracting header and txt fileinfo.

        Read from parent dir 'src', place *.header and *.txt files into
        a parallel copy/ dir.
```

B.6-2

```
    """

    if os.path.isdir(src):
        names = os.listdir(src)
        names.sort()
        docdir = ExtractDoc(dict)
        if docdir :
            for name in names:
                # Create the basename in the form: "ac850313"
                bname = os.path.splitext(name)[0]
                # actual pathname : "./insert/ccdts/ac850313.hw"
                srcname  = os.path.join(src, name)
                # Create the target dirname in the form: "./docs/ccdts/ac850313/"
                target   = os.path.join(copy,
                            os.path.join(os.path.basename(regsub.gsub('/$','',s
                            (bname+'/') ))

                if docdir.parse(srcname) :
                    if not os.path.exists(target) :
                        os.mkdir(target,0754)
                else:
                    print 'Error parsing', srcname
                    break

                res = docdir['header']
                if res :
                    mkcoldb.putfile(os.path.join(target, bname+'.header'), res)

                res = docdir['txt']
                if res :
                    mkcoldb.putfile(os.path.join(target, bname+'.txt'), res)

                print srcname,' >', target
        else: print 'Error creating ExtractDoc'
```

B.7 CSP.PY

```
#
#-------------------------------------------------------------------
# Title      : csp.py
# Purpose    :
"""Move csp type documents from CDOCS database to ISTB

Author     : Kent Polk          - SwRI Div. 17
             David Lincoln      - SwRI Div. 09
Date       : 17 Sep 96
"""
#-------------------------------------------------------------------
import os
import sys
import string
import regsub
import regex
import shutil
import mkcoldb

cspDict = {
            'order': ['TITLE','AUTHOR','ADDRESSEES','DOCUMENT_DATE'],
            'regex': '^[A-Z_]+:',
            'redate': 'DOCUMENT_DATE',
            'TITLE':'TITLE',
            'AUTHOR':'AUTHOR',
            'ADDRESSEES':'ADDRESSEE',
            'DOCUMENT_DATE':'DATE',
            }

from rfc822 import _monthnames
def redate(date):
   """Change datestr from dd MMM yyy to dd-Mmm-yyyy
   """
   dd, mm, yy = tuple(string.split(date))
   return "%s-%s-%s" % (dd, mm[:1]+string.lower(mm[1:]), yy)

#--------------------------------------------------------
def parsedoc(file, tokregx):
   """Parse a text document and return as a dictionary.

   Assumes the document is of the form:
   TOKEN1: item for token1
   TOKEN2: item for token2

   tokregx is a compiled regular expression used to locate the tokens
   """
   tokDict = {}
   f = open(file, 'r')
   while 1:
      line = f.readline()
      if not line:
         break
      matchlen = tokregx.match(line)
      if matchlen > -1 :
         line = regsub.gsub('\n','',line)
         key = line[:matchlen-1]    # Don't include the colon
         tokDict[key] = [line[matchlen+1:]]
   return tokDict

from copy import deepcopy
```

```
class ExtractDoc:
   """Extract pertinent info from a CDOCS document

   Only one ExtractDoc object per set of files is required.
   Creates a list item for each key provided in the matchDict['order']
   provided it exists also as a key.  Each list item is prefixed by the
   corresponding matchDict object (label).  The first keyword is 'header'

   use getitem (doc[key]) to retrieve a dictionary item
   """
   # Interface -- initialize and reset this instance

   def __init__(self, matchDict):
      # Create one object per document class.
      # Note that parse() resets for each new file provided.
      #
      # Compile the regular expression and set the header order list
      self.Dict    = {}
      self.docDict = {}
      self.matDict = deepcopy(matchDict)
      self.order   = []
      self.redate = ""

      # if redate exists, it must reference a key in the matchDict
      # that represents a date string in the form yyyymmdd so it can
      # convert the date to dd mmm yyyy format
      key = 'redate'
      if matchDict.has_key(key) :
         self.redate = matchDict[key]

      # if order exists, it will be used to provide an ordered list
      # of keys which should be in matchDict for the header list
      # building operation.  If order isn't provided, the header
      # list will be in an arbitrary order.
      key = 'order'
      if matchDict.has_key(key) :
         self.order  = matchDict[key]

      # regex must exist... it is the regular expression used to
      # parse the document.
      key = 'regex'
      if matchDict.has_key(key) :
         self.regtok = regex.compile(matchDict[key])
      else : return None

#def keys(self):
#return self._index.keys()
#
#def has_key(self, key):
#return self._index.has_key(key)

   def __len__(self):
      return len(self.Dict) +len(self.docDict) +len(self.matDict)

   def __getitem__(self, key):
      #Get an object associated with the specified key.
      #Search all internal class dictionaries.
      if self.Dict.has_key(key) :
         return self.Dict[key]
      elif self.docDict.has_key(key) :
```

```
            return self.Dict[key]
        elif self.docDict.has_key(key) :
            return self.docDict[key]
        elif self.matDict.has_key(key) :
            return self.matDict[key]
        else : return None

    def parse(self, filename):
        #Extract header list from the parse dictionary of a document.

        self.docDict = parsedoc(filename, self.regtok)

        # Do we need to change the date format?
        # ought to provide a conversion function someday...
        key = self.redate
        if key in self.docDict.keys() :
            self.docDict[key] = [redate(self.docDict[key][0])]

        headerlist = []
        if len(self.order) : # build this list in the order provided
            for key in self.order :
                if key in self.matDict.keys() and key in self.docDict.keys() :
                    headerlist.append(
                        self.matDict[key]+': '+\
                        regsub.gsub('\n','',string.join(self.docDict[key]))+'\n')
        else:                       # else this list will be unordered
            for key in self.docDict.keys() :
                if key in self.matDict.keys() :
                    headerlist.append(
                        self.matDict[key]+': '+\
                        regsub.gsub('\n','',string.join(self.docDict[key]))+'\n')

        self.Dict['header'] = headerlist

        return 1

def subd_extract_docs(names, root, skip, destdir, dict):
    """Extracting header and txt fileinfo.

    Read from parent dir 'src', place *.header and *.txt files into
    a parallel copy/ dir.
    """
    docobj = ExtractDoc(dict)
    if docobj :

        arcpath = os.path.join(root,os.path.join('archive',skip))

# For example:
#  hsd = '/samson/u03/prod1/data/p_fcsp/p_fcspl/'
#  tsd = '/samson/u03/prod1/archive/p_fcsp/p_fcspl/txt/'
#  dsd = '/samson/u03/prod1/archive/p_fcsp/p_fcspl/bin/'
#  isd = '/samson/u03/prod1/images/p_fcsp/p_fcspl/'

        hdrsrc = os.path.join(root,os.path.join('data',skip))
        txtsrc = os.path.join(arcpath,'txt/')
        docsrc = os.path.join(arcpath,'bin/')
        imgsrc = os.path.join(root,os.path.join('images',skip))

        for name in names:
            # Create the target dirname
```



```
target = os.path.join(destdir, name+'/')

# actual pathname : " /insert/cddis/ac850113.hw"
srcname = os.path.join(hdrsrc, name)

# build srcname
if docobj.parse(srcname+'.001')
    if not os.path.exists(target)
        os.mkdir(target,0754)
else:
    print 'Error parsing', srcname
    break

res = docobj['header']
if res :
    mkcoldb.putfile(os.path.join(target, name+'.header'), res)

# Copy the txt file
src = os.path.join(txtsrc, name+'.txt')
dst = os.path.join(target, name+'.txt')
if os.path.exists(src) :
    shutil.copy2(src, dst)

# Copy the wp file
src = os.path.join(docsrc, name+'.bin')
dst = os.path.join(target, name+'.wp')
if os.path.exists(src) :
    shutil.copy2(src, dst)

# Copy the tif files
srcdir = os.path.join(imgsrc, name)
if os.path.isdir(srcdir):
    names = os.listdir(srcdir)
    names.sort()
    for tifname in names:
        ext = os.path.splitext(tifname)[1]
        if ext == '.tif' :
            src = os.path.join(os.path.join(imgsrc,name), tifname)
            dst = os.path.join(target, tifname)
            shutil.copy2(src, dst)

print srcname,' >', target
```