Mkload
Bernie Simon
July 25, 1996

Usage

The task is run by entering the name of a task followed by a list of the reference files to create load files for. For example,

```
mkload a3d1145dy.cy0 d9g1458cu.r2h
```

Wild card patterns can be used in the list of filenames, but care should be taken that the pattern expands to the image header file name and not the data file name. The output of the task is a load file for each file name on the command line. The load file has the same root file name as the original file, but has the extension `.lod`.

Each load file describes a single reference file. A load file is an ASCII file with no upper limit on the length of its fields. Field widths are variable, so no space is wasted. Because the load file is ASCII, it can be viewed and editing with any text editor and transferred between computer systems without conversion.

The loadfile is divided into two parts: a header and body. The body is further subdivided into rows. Each divison has a terminator. The termintaor for the heading is as ENDHEADER line. For the body it is an ENDFILE line. For each row it is an ENDROW line. A terminator is a single word, in either case, starting in the first column, as the sole word on the line. Comments can be placed anywhere in the loadfile before the ENDFILE line. Comments are any line whose first non-white character is a sharp (#). Blank lines are treated as comments, except that they can also follow the ENDFILE line.

This task copies the information that will be stored in the database from the reference file to the load file. It reads the CDBS database to get the comparsion file name sets the COMPARISON_FILE to this value. If no comparison file is found in the database, it sets the field to "(INITIAL)". If the useafter date is not found in the reference file header, it sets the USEAFTER_DATE field to the useafter date of the comparison file on the assumption that the useafter date of the reference file is the same as that of the comparsion file.

Some other fields are left blank that must be filled in by the user. These are the OPUS_FLAG, CHANGE_LEVEL, and COMMENT in the header and PEDIGREE, OBSERVATION_BEGIN_DATE, OBSERVATION_END_DATE, and COMMENT in the body of the load file.

Data Files

Mkload uses the catalog file to determine which type of reference file it is reading The catalog file contains the selection rules used to determine the instrument name and reference file type for a reference file. The first match that succeeds determines the reference file type. If no records in the catalog match the reference file, the task prints a warning message. The fields in the catalog file are described in the documentation for the certify task.

Environment Variables

Mkload uses three environment variables that control its access to the database. `CDBS_SERVER` sets the database server the task accesses. `CDBS_DB` set the database that is used by the task. And `CDBS_USER` sets the user name used to access the database. If these three environment variables are not found, the values used by the task default to `ROBBIE, cdbs_ops,` and the user's login name, repectively.

Algorithm

The main routine in `mkload` is named `task.` It reads the catalog file and the loops over each reference file on the command line, calling `mkload` to create the load file. After processing all the reference files, the main routine closes the catalog file.

`Mkload` determines the type of the reference file from the catalog file, creates an empty table, and calls `mkload_header` and `mkload_body` to read the information from the reference file into the table. It then calls `mkload_name` to create the name of the load file, writes the table to the load file, and frees the table. `Mkload_header` writes the load file header keywords. Some of these keywords are read from the database, so, for the present, they are left blank. `Mkload_body` writes the observation mode keywords and the pedigree information to the body of the table. If the reference file is an image, it writes one row to the body of the table from the information in the image header. If the reference file is a table, it writes one row for each row in the table. The observation mode keywords are read from an array in `obsmode.h` which contains all the observation mode keywords indexed by instrument. The function searches the reference file for the keyword and if it finds it, it is written to the table. If not, that particular keyword is not written. `Mkload_compfile` sets the comparsion file name and useafter dates in the header to values read from the CDBS database. `Mkload_name` creates the name of the load file by changing the extension of the reference file to `.lod.`

The `loadfile` library handles reading and writing of load files. The load file is read into an in memory structure called a table, which is then accessed by the program. The functions to read and write a load file are:

```
tab = read_loadfile (fname)
```

```
write_loadfile (fname, tab)
```

`Write_loadfile` does not free the table structure or modify it in any way. To free the table, the function `free_table` must be called.

All characters are converted to upper case when read into the table and superfluous blanks are deleted. Superfluous blanks are leading and trailing blanks and more than one blank between words. So when retrieving information from the table, specify keyword names and column names in upper case with only one blank between words.

The table functions implement an in-memory ASCII table. The table has header keywords accessed by name and cells accessed by column name and row number. The values returned by the get functions are pointers to the contents of the table and should not be modified by the calling function. Also, doing a put to the table may invalidate the pointers returned if memory is shuffled. Copy the results of a get to another string before doing a put. Table access is oriented towards random access, although sequential access is also supported. The get functions return a value of NULL if the get does not succeed.

For random access, function `get_tab_key` reads a header keyword and `get_tab_cell` reads a table cell. For sequential access, `get_tab_keyname_bynum` retrieves a header keyword name, `get_tab_keyval_bynum` retrieves a header keyword value, and `get_tab_cell_bynum` reads a table cell. The macros `nkey_table, nrow_table,` and `ncol_table` get the dimensions of a table.

```
str = get_tab_key (tab, keyname)
str = get_tab_cell (tab, colname, row)

str = get_tab_keyname_bynum (tab, keynum)
str = get_tab_keyval_bynum (tab, keynum)
str = get_tab_cell_bynum (tab, colnum, row)

nkey = nkey_table (tab)
nrow = nrow_table (tab)
ncol = ncol_table (tab)
```

Most often a table is created by reading the contents of a load file into it. However, it is possible to create a table in memory or modify the contents of a table read from a load file. The use of the add column routine is optional, as columns are implicitly added when the first row of the table is written. The add column routine can be used when the table column names are known before the table is created. The add column routine returns the number of the column created or -1 if the column cannot be created because the table is already written to.

The put functions return a value of -1 if the put does not succeed and 0 if it does. In addition to the error codes returned by the get and put functions, an error message is stored in the global variable `lod_error.`

The functions `put_tab_key` and `put_tab_cell` put a new value to the table header and body respective. The functions `cat_tab_key` and `cat_tab_cell` concatenate a value to an existing value in a table.

```
ok = put_tab_key (tab, keyname, value)
ok = put_tab_cell (tab, colname, row, value)
ok = cat_tab_key (tab, colname, value)
ok = cat_tab_cell (tab, colname, row, value)
```

Test Data

Two test files are stored in the `test` subdirectory. To test this task, run it with the command line

```
mkload a3d1145dy.cy0 d9g1458cu.r2h
```