Certify
Bernie Simon
July 10, 1996

Usage

This task is run by entering the name of the task followed a blank separated list of the reference files you wish to check. For example,

```
certify a3d1145dy.cy0 d9g1458cu.r2h
```

Wildcard 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 header line for each file that is checked, plus one line for each error that is found in the file. Output from the task is printed on `stdout`. Here is an example of the output produced by the task:

```
== Checking a3d1145dy.cy0 ==
Error in detector[1]: "red" is not in amber,blue
Error in aper_id[1]: field is blank
Error in aper_pos: field is missing
```

The task produces a header line for each file processed and one line for each error encountered. There are three principal kinds of errors: first the field value is illegal, as in the first error messages. The error output contains the field name (column name), the row number, the illegal value, and the list of legal values. The second and third error messages show the result of blank and missing fields. The user may also get other error output from certify.

```
certify test2.dy0 test3.cy0
== Checking test2.dy0 ==
Cannot determine reference file type (test2.dy0)
== Checking test3.cy0 ==
Cannot open file (test3.cy0)
```

Certify determines the reference file type from its extension. Because images do not have unique extensions, the task also looks into the image header for the INSTRUME keyword to determine the reference file type. If the task cannot determine the reference file type from this information, it produces the first error message above. If the task is unable to open the reference file, either because it does not exist or because of protection problems, it produces the second error message.

The task checks reference files to see if specific fields within the file have legal values. Which fields are checked are specified in a template file. The task checks three kinds of data fields: header keywords (both in tables and images), group parameters, and table columns. The value in the reference file is checked against a list of legal values. The elements in the list may be ranges of values. A special test for the pedigree keyword is also supported. If the list of values is empty, the task checks for the existence of the specified field.

There are three items of information in the error message that the task produces: the field name, field value, and list of legal values. The field name is the name of the header keyword, group parameter, or table column. If the field is not a header keyword, the field name has a bracketed number appended to it. This number indicates the group number in the image or the row number in the table of the field that is in error. The field value is the value as read from the reference file. The list is a comma separated list of legal values for the field to take. The list can also contain ranges. The lower and upper limits of the range are separated in the list by a colon.

Data Files

Two kinds data files control what the program checks: the catalog file and the template file. These files are stored in a controlled directory and cannot be modified by the average user. Both are ascii files and share several common formatting features. Case is not significant to the meaning of the information in the file. Blank and comment lines are ignored. Comment lines have a sharp (#) character as the first non-white character on the line. Each line in a file normally constitutes a complete record; however, lines can be continued by making the backslash (\) the last non-white character on the line. If a line is continued, all trailing whitespace on the first line and leading whitespace on the second line is ignored. Fields within a record are separated by whitespace. If a field contains whitespace, the field can be surrounded by quotation marks and all characters within quotation marks will be treated as part of the field. Some fields may also contain subfields, separated by other characters (usually a comma). Any of these separators can also be included in the subfield by surrounding them with quotation marks.

The name of the catalog file is fixed (for now) in the task as `cdbscatalog.dat`. Each record in the catalog file contains the names of the individual template files and the selection rules used to match the reference file with its template file. The match is made on the basis of the reference file extension and header keywords. The first match that succeeds selects the template file. If no records in the catalog match the reference file, the task prints a warning message. The fields in Figure 1 are contained in each record of the template catalog.

field	definition	example
instrument	The HST instrument name	`fos`
dbtable	The CDBS database table	`synphot`
reftype	The OPUS file type	`ccs0`
filetype	The reference file type	`table`
tpl_file	The template file name	`fos_cy0.tpl`
extension	The reference file extension	`.cy0`
keywords	A list of keyword names and values	`parmid=cyccs0r`

Figure 1: Template catalog fields

The fields in the catalog file have the following restrictions:

instrument The HST instrument name must be one of the following: `foc, fos, hrs, hsp, nic, stis, synphot, wfpc, wfpc2,` or `multi`.

dbtable The dbtable must be one ot the following: `foc, fos, hrs, hsp, nic, stis, synphot, wfpc,` or `wfpc2`.

reftype The OPUS reference file name associated the particular type of reference file.

filetype Legal values for this field are `image`, `table`, or `loadfile` or any abbreviation of these strings.

tpl_file Any legal filename on the system. If the filename does not contain a directory specification, it must be placed in the same directory as the template catalog. (This is the usual case.)

extension This is the first of two tests for selecting the template file. The extension *must* begin with a dot. The extension may include the wildcard characters `?` or `*`.

keywords A list of keyword = value pairs in the reference file header. This is the second test for selecting the template file. The list may be empty (blank), or contain one or two pairs. Matching is done as text and not numerically.

Each record in the template file identifies an item in a reference file to be matched. Items may either be header keywords (from an image or a table), image group parameters, or table column names. The fields in the record identify the item name, type, datatype, and legal values. If the data item does not exist or does not match the list of legal values, a message is printed. The fields in Figure 2 are in each record of the template file.

field	definition	example
name	Name of data item	`aper_id`
type	Type of data item	`column`
datatype	Data type of item	`character`
presence	whether item is required	verb+required+
values	List of legal values	`a-1:a-4,b-1:b-4,c-1:c-4`

Figure 2: Template file fields

The fields in the template file have the following restrictions:

name Any name that is legal for the specified data item.

type There are three supported values for this field: `header`, `group`, and `column`. Values may also be any abbreviation of these strings.

datatypes The legal datatypes are `integer, real, logical, double,` and `character` or any abbreviation of these strings.

presence The legal values for this field are `optional`, `present`, and `required` or any abbreviation of these words. The values control whether an error message is printed when the field is missing or blank. If the value is `required` , an error message is printed if the field is missing or blank. If the value is `present`, an error message is only printed if the field is missing. Id the value is `optional`, an error message is not printed in either of these cases.

values A comma separated list of legal values. Elements in the list may be single values or ranges of values. If the element is a range, the minimum value appears first and is separated from the maximum value by a colon. The endpoints of the range are included in the range. The value may be preceded by an ampersand (&), which indicates that a special function is used to check the data item. There are two special function currently supported: `pedigree` and `sybdate.` The list of values may be empty, in which case only the existence of the data item will be checked.

Algorithm

The main routine, `task`, reads and parses the template catalog and then loops over each reference file and calls `certify` to process the file. `Certify` selects the template file to process and reads and parses it. It then loops over each rule (record) in the template file. If the rule names a header keyword, it calls `check_rule` to process it. Otherwise it calls `study_rule` to sort the values and merge ranges to optimize the rule for checking.

After looping over the rules once to check the header keywords, the task loops over each group in the reference file if the file is an image or each row in the reference file if it is a table. It does an inner loop over each rule in template file, calling `check_rule` to process the rules for group parameters and table columns.

Most of the code in the task is divided into three structures and the functions that support them. The first structure is the `lookup` table, which stores the information read from the catalog file. The table stucture is the same structure that is used to hold information read from a load file. The functions which use this structure are `open_lookup, find_lookup,` and `close_lookup. Open_lookup` reads the catalog file, parses it, and stores the information in the lookup table. `Find_lookup` returns the row of the table which matches the information in the reference file. And `close_lookup` releases the table.

The second structure is the `reffile` structure. The functions associated with this structure provide a uniform interface to the different types of reference files that the task checks. The structure use the abstraction that each reference file has a header and a body. The body contains one or more sections. A section of an image is a group and a section of a table is a row. The functions which provide this abstraction are `open_reffile, close_reffile, hdr_reffile, body_reffile,` and `nrow_reffile.` The open_reffile and `close_reffile` open and close the reference file, respectively. `Hdr_reffile` reads a keyword from the reference file header and `body_reffile` reads reads a value from a row or group in the reference file body. Row numbers start with zero. `Nrow_reffile` returns the number of rows or groups in a reference file.

The `template` structure holds the contents of the template file. The functions which access this structure are `open_template, close_template, next_template,` and `skip_template. Open_template` reads the information from the template file, parses it, and stores the information in the structure. `Close_template` releases the memory used by the structure. `Next_template` returns a pointer to the next rule in the structure. It is used to loop over rules in the template. `Skip_template` returns `YES` if all the rules in the template are marked as skip. Header rules and body rules that only check for existence are marked as skip after the first time they are used. To prevent unecessay processing, the loop over the body of the reference file exits if `skip_template` returns `YES.`

Several procedures also use single rules within the template structure: `body_rule, check_rule, study_rule, parse_rule,` and `free_rule. Body_rule` returns `YES` if a rule describes a body data item (group parameter or table column) and `NO` if it describes a header data item. `Check_rule` compares a data item to a rule and writes a message if they do not match. `Study_rule` sorts the values contained in a rule and merges overlapping ranges. It is used to optimize multiple searches performed by body rules. `Parse_rule` parses a single record within the template file and stores it in the template structure. It is called by `open_template. Free_rule` releases memory held by the rule in the template structure and is called by `close_template.`

System dependencies

The code for this task has been written to be a system independent as possible. However, there are some code differences between the VMS and Unix versions. These differences are containd in `#ifdef` blocks selected by the macro `VMS.` The code differences concern wildcard expansion on the command line, which is not done in VMS, and the exit status returned by the program.

In addition to these operating system differences, the name of the directory containing the template catalog and template files is specified by the macro `DATADIR` defined in the file `system.h`.

The task also needs to be linked with the iraf and stsdas libraries. The location of these libraries can chnage from system to system so the build procedure must be changed to take this into account. However, the build procedures for the new version of CDBS are still TBD.

Test Data

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

```
certify a3d1145dy.cy0 d9g1458cu.r2h
```