HASH

Note: This module is provided for backwards compatibility. New applications should use the HASH-FILE Library Module instead of this module.

Hash permits information associated with string or atom keys to be stored on and retrieved from files. The information (or values) associated with the keys in a file may be numbers, strings, or arbitary Lisp expressions. The associates are maintained by a hashing scheme that minimizes the number of file operations it takes to access a value from its key.

Information is saved in a hash file, which is analogous to a hash array. Actually, a hash file can be either the file itself, or the handle on that file which is used by the Hash module. The latter, of data type HashFile, is the datum returned by `CREATEHASHFILE` or `OPENHASHFILE`, currently an array record containing the hash file name, the number of slots in the file, the used slots, and other details. All other functions with hash file arguments use this datum.

In older implementations (e.g., for Interlisp-10), hash files came in several varieties, according to the types of value stored in them. The EMYCIN system provided even more flexibility.

This system only supports the most general EXPR type of hash files and EMYCIN-style TEXT entries, in the same file. The *VALUETYPE* and *ITEMLENGTH* arguments are for the most part ignored. Two-key hashing is supported in this system, but it is discouraged as it is only used in EMYCIN, not in the Interlisp-10 system. The functions `GETPAGE`, `DELPAGE`, and `GETPNAME`, which manipulate secret pages, do not exist in this implementation. However, it is permissible to write data at the end of a hash file. That data will be ignored by the Hash module, and can be used to store additional data.

The Hash module views files as a sequence of bytes, randomly accessible. No notice is made of pages, and it is assumed that the host computer buffers I/O sufficiently.

Hash files consist of a short header section (8 bytes), a layer of pointers (`4*HASHFILE:`Size bytes), followed by ASCII data. Pointers are 3 bytes wide, preceded by a status byte. The pointers point to key PNAMES in the data section, where each key is followed by its value.

Deleted key pointers are reused but deleted data space is not, so rehashing is required if many items have been replaced.

The data section starts at `4*HASHFILE:` Size + 9, and consists of alternating keys and values. As deleted data is not rewritten, not all data in the data section is valid.

When a key hashes into a used slot, a probe value is added to it to find the next slot to search. The probe value is a small prime derived from the original hash key.

Requirements

Hash files must reside on a random-access device (not a TCP/IP file server).

Installation

Load `HASH.LCOM` from the Library.

Functions

Creating a Hash File

(CREATEHASHFILE *FILE VALUETYPE ITEMLENGTH #ENTRIES SMASH COPYFN*) [Function]

Creates a new hash file named *FILE*. All other arguments are optional.

VALUETYPE is ignored in this implementation; any hash file can accommodate both Lisp expressions and text.

ITEMLENGTH is not used by the system but is currently saved on the file (if less than 256) for future use.

#ENTRIES is an estimate of the number of entries the file will have. (This should be a realistic guess.)

SMASH is a hash file datum to reuse.

COPYFN is a function to be applied to entries when the file is rehashed (see the description of REHASHFILE below).

Opening and Closing Hash Files

Before you can use a hashfile with this module, you have to open it using the following function.

(OPENHASHFILE *FILE ACCESS ITEMLENGTH #ENTRIES SMASH*) [Function]

Reopens the previously existing hash file *FILE*.

Access may be INPUT (or NIL), in which case *FILE* is opened for reading only, or BOTH, in which case *FILE* is open for both input and output. Causes the error "not a hashfile" if *FILE* is not recognized as a hash file.

ITEM LENGTH and *#ENTRIES* are for backward compatibility with EMYCIN where OPENHASHFILE also created new hash files; these arguments should be avoided.

SMASH is a hash file datum to reuse.

If *ACCESS* is BOTH and *FILE* is a hash file open for reading only, OPENHASHFILE attempts to close it and reopen it for writing. Otherwise, if *FILE* designates an already open hash file, OPENHASHFILE is a no-op.

OPENHASHFILE returns a hash file datum.

(CLOSEHASHFILE *HASHFILE REOPEN*) [Function]

Closes HASHFILE (when you are finished using a hash file, you should close it). If *REOPEN* is non-NIL, it should be one of the accepted access types. In this case, the file is closed and then immediately reopened with *ACCESS* = *REOPEN*. This is used to make sure the hash file is valid on the disk.

Storing and Retrieving Data

(PUTHASHFILE *KEY VALUE HASHFILE KEY2*) [Function]

Puts *VALUE* under *KEY* in *HASHFILE*. If *VALUE* is NIL, any previous entry for *KEY* is deleted. *KEY2* is for EMYCIN two-key hashing; *KEY2* is internally appended to *KEY* and they are treated as a single key.

(GETHASHVILE *KEY HASHFILE KEY2*) [Function]

Gets the value stored under *KEY* in *HASHFILE*. *KEY2* is necessary if it was supplied to PUTHASHFILE.

(LOOKUPHASHFILE *KEY VALUE HASHFILE CALLTYPE KEY2*) [Function]

A generalized entry for inserting and retrieving values; provides certain options not available with GETHASHFILE or PUTHASHFILE. LOOKUPHASHFILE looks up *KEY* in *HASHFILE*.

CALLTYPE is an atom or a list of atoms. The keywords are interpreted as follows:

RETRIEVE — If *KEY* is found, then if *CALLTYPE* is or contains RETRIEVE the old value is returned from LOOKUPHASHFILE; otherwise returns T.

DELETE — If *CALLTYPE* is or contains DELETE, the value associated with *KEY* is deleted from the file.

REPLACE — If *CALLTYPE* is or contains REPLACE, the old value is replaced with *VALUE*.

INSERT — If *CALLTYPE* is or contains INSERT, LOOKUPHASHFILE inserts *VALUE* as the value associated with *KEY*.

Combinations are possible. For example, (RETRIEVE DELETE) deletes a key and returns the old value.

(PUTHASHTEXT *KEY SRCFIL HASHFILE START END*) [Function]

Puts text from stream *SRCFIL* onto *HASHFILE* under *KEY*. *START* and *END* are passed directly to COPYBYTES.

(GETHASHTEXT *KEY HASHFILE DSTFIL*) [Function]

Uses COPYBYTES to retrieve text stored under *KEY* on *HASHFILE*. The bytes are output to the stream *DSTFIL*.

Functions for Manipulating Hash Files

(HASHFILEP *HASHFILE WRITE?*) [Function]

Returns *HASHFILE* if it is a valid, open hash file datum, or returns the hash file datum associated with *HASHFILE* if it is the name of an open hash file. If *WRITE?* is non-NIL, *HASHFILE* must also be open for write access.

(HASHFILEPROP *HASHFILE PROPERTY*) [Function]

Returns the value of a *PROPERTY* of a *HASHFILE* datum. Currently accepted properties are: NAME, ACCESS, VALUETYPE, ITEMLENGTH, SIZE, #ENTRIES, COPYFN and STREAM.

(HASHFILENAME *HASHFILE*) [Function]

Same as (HASHFILEPROP *HASHFILE* 'NAME).

(MAPHASHFILE *HASHFILE MAPFN DOUBLE*) [Function]

Maps over *HASHFILE* applying *MAPFN*. If *MAPFN* takes two arguments, it is applied to *KEY* and *VALUE*. If *MAPFN* only takes one argument, it is only applied to *KEY* and saves the cost of reading the value from the file. If *DOUBLE* is non-NIL, then *MAPFN* is applied to (*KEY1 KEY2 VALUE*), or (*KEY1 KEY2*) if the *MAPFN* only takes two arguments.

(REHASHFILE *HASHFILE NEWNAME*) [Function]

As keys are replaced, space in the data section of the file is not reused (through space in the key section is). Eventually the file may need rehashing to reclaim the wasted data space. REHASHFILE is really a special case of COPYHASHFILE, and creates a new file. If *NEWNAME* is non-NIL, it is taken as the name of the rehashed file.

The system automatically rehashes files when 7/8 of the key section is filled. The system prints a message when automatically rehashing a file if the global variable REHASHGAG is non-NIL.

Certain applications save data outside Hash's normal framework. Hash files for those applications need a custom *COPYFN* (supplied in the call to CREATEHASHFILE), which is used to copy data during the rehasing process. The *COPYFN* is used as the FN argument to COPYHASHFILE during the rehashing.

(COPYHASHFILE *HASHFILE NEWNAME FN VALUETYPE LEAVEOPEN*) [Function]

Makes a copy of *HASHFILE* under *NEWNAME*.

Each key and value pair is moved individually, and, if *FN* is supplied, is applied to (*KEY VALUE HASHFILE NEWHASHFILE*).

What is returned is used as the value of the key in the new hash file. (This lets you intervene, perhaps to copy out-of-bank data associated with *VALUE*.)

VALUETYPE is a no-op.

If *LEAVEOPEN* is non-NIL, the new hash file datum is returned open. Otherwise, the new hash file is closed and the name is returned.

(HASHFILEPLST *HASHFILE XWORD*) [Function]

Returns a Lisp generator for the keys in *HASHFILE*, usable with the spelling corrector. If *XWORD* is supplied, only keys starting with the prefix in *XWORD* are generated.

Global Variables of Hash

HASHFILEDEFAULTSIZE [Variable]

Size used when *#ENTRIES* is omitted or is too small. Default is 512.

HASHFILEDTBL [Variable]

The hash file read table. Default is ORIG.

HASHLOADFACTOR [Variable]

The ration, used slots/total slots, at which the system rehashes the file. Default is ↑A.

HFGROWTHFACTOR [Variable]

The ratio of total slots to used slots when a hash file is created. Default is 3.

REHASHGAG [Variable]

Flags whether to print message when rehashing; initially off. Default is NIL.

SYSHASHFILE [Variable]

The current hash file. Default is NIL.

SYSHASHFILELST [Variable]

An Alist of open hash files. Default is NIL.

Limitations

The system currently is able to manipulate files on CORE, DSK, FLOPPY, and over the network, via leaf servers. Hash files can be used with NS servers only if they support random access files.

Due to the pointer size, only hash files of less than 6 million initial entries can be created, though these can grow to 14 million entries before automatic rehashing exceeds the pointer limit. The total file length is limited to 16 milion bytes. No range checking is done for these limits.

Two-key files operate on pnames only, without regard to packages.

[This page intentionally left blank]