NAME

Tcl_InitHashTable, Tcl_DeleteHashTable, Tcl_CreateHashEntry, Tcl_DeleteHashEntry, Tcl_FindHashEntry, Tcl_GetHashValue, Tcl_SetHashValue, Tcl_GetHashKey, Tcl_FirstHashEntry, Tcl_NextHashEntry, Tcl_HashStats − procedures to manage hash tables

SYNOPSIS

#include <tclHash.h>

Tcl_InitHashTable(*tablePtr, keyType*)

Tcl_DeleteHashTable(*tablePtr*)

Tcl_HashEntry *
Tcl_CreateHashEntry(*tablePtr, key, newPtr*)

Tcl_DeleteHashEntry(*entryPtr*)

Tcl_HashEntry *
Tcl_FindHashEntry(*tablePtr, key*)

ClientData
Tcl_GetHashValue(*entryPtr*)

Tcl_SetHashValue(*entryPtr, value*)

char *
Tcl_GetHashKey(*tablePtr, entryPtr*)

Tcl_HashEntry *
Tcl_FirstHashEntry(*tablePtr, searchPtr*)

Tcl_HashEntry *
Tcl_NextHashEntry(*searchPtr*)

char *
Tcl_HashStats(*tablePtr*)

ARGUMENTS

Tcl_HashTable	**tablePtr*	(in)	Address of hash table structure (for all procedures but **Tcl_InitHashTable**, this must have been initialized by previous call to **Tcl_InitHashTable**).
int	*keyType*	(in)	Kind of keys to use for new hash table. Must be either TCL_STRING_KEYS, TCL_ONE_WORD_KEYS, or an integer value greater than 1.
char	**key*	(in)	Key to use for probe into table. Exact form depends on *keyType* used to create table.
int	**newPtr*	(out)	The word at **newPtr* is set to 1 if a new entry was created and 0 if there was already an entry for *key*.
Tcl_HashEntry	**entryPtr*	(in)	Pointer to hash table entry.
ClientData	*value*	(in)	New value to assign to hash table entry. Need not have type ClientData, but must fit in same space as ClientData.

Tcl_HashSearch	**searchPtr*	(in)	Pointer to record to use to keep track of progress in enumerating all the entries in a hash table.

DESCRIPTION

A hash table consists of zero or more entries, each consisting of a key and a value. Given the key for an entry, the hashing routines can very quickly locate the entry, and hence its value. There may be at most one entry in a hash table with a particular key, but many entries may have the same value. Keys can take one of three forms: strings, one-word values, or integer arrays. All of the keys in a given table have the same form, which is specified when the table is initialized.

The value of a hash table entry can be anything that fits in the same space as a ‘‘char *’’ pointer. Values for hash table entries are managed entirely by clients, not by the hash module itself. Typically each entry’s value is a pointer to a data structure managed by client code.

Hash tables grow gracefully as the number of entries increases, so that there are always less than three entries per hash bucket, on average. This allows for fast lookups regardless of the number of entries in a table.

Tcl_InitHashTable initializes a structure that describes a new hash table. The space for the structure is provided by the caller, not by the hash module. The value of *keyType* indicates what kinds of keys will be used for all entries in the table. *KeyType* must have one of the following values:

TCL_STRING_KEYS	Keys are null-terminated ASCII strings. They are passed to hashing routines using the address of the first character of the string.
TCL_ONE_WORD_KEYS	Keys are single-word values; they are passed to hashing routines and stored in hash table entries as ‘‘char *’’ values. The pointer value is the key; it need not (and usually doesn’t) actually point to a string.
other	If *keyType* is not TCL_STRING_KEYS or TCL_ONE_WORD_KEYS, then it must be an integer value greater than 1. In this case the keys will be arrays of ‘‘int’’ values, where *keyType* gives the number of ints in each key. This allows structures to be used as keys. All keys must have the same size. Array keys are passed into hashing functions using the address of the first int in the array.

Tcl_DeleteHashTable deletes all of the entries in a hash table and frees up the memory associated with the table’s bucket array and entries. It does not free the actual table structure (pointed to by *tablePtr*), since that memory is assumed to be managed by the client. **Tcl_DeleteHashTable** also does not free or otherwise manipulate the values of the hash table entries. If the entry values point to dynamically-allocated memory, then it is the client’s responsibility to free these structures before deleting the table.

Tcl_CreateHashEntry locates the entry corresponding to a particular key, creating a new entry in the table if there wasn’t already one with the given key. If an entry already existed with the given key then **newPtr* is set to zero. If a new entry was created, then **newPtr* is set to a non-zero value and the value of the new entry will be set to zero. The return value from **Tcl_CreateHashEntry** is a pointer to the entry, which may be used to retrieve and modify the entry’s value or to delete the entry from the table.

Tcl_DeleteHashEntry will remove an existing entry from a table. The memory associated with the entry itself will be freed, but the client is responsible for any cleanup associated with the entry’s value, such as freeing a structure that it points to.

Tcl_FindHashEntry is similar to **Tcl_CreateHashEntry** except that it doesn’t create a new entry if the key doesn’t exist; instead, it returns NULL as result.

Tcl_GetHashValue and **Tcl_SetHashValue** are used to read and write an entry’s value, respectively. Values are stored and retrieved as type ‘‘ClientData’’, which is large enough to hold a pointer value. On almost all machines this is large enough to hold an integer value too.

Tcl_GetHashKey returns the key for a given hash table entry, either as a pointer to a string, a one-word (‘‘char *’’) key, or as a pointer to the first word of an array of integers, depending on the *keyType* used to create a hash table. In all cases **Tcl_GetHashKey** returns a result with type ‘‘char *’’. When the key is a string or array, the result of **Tcl_GetHashKey** points to information in the table entry; this information will remain valid until the entry is deleted or its table is deleted.

Tcl_FirstHashEntry and **Tcl_NextHashEntry** may be used to scan all of the entries in a hash table. A structure of type ‘‘Tcl_HashSearch’’, provided by the client, is used to keep track of progress through the table. **Tcl_FirstHashEntry** initializes the search record and returns the first entry in the table (or NULL if the table is empty). Each susequent call to **Tcl_NextHashEntry** returns the next entry in the table or NULL if the end of the table has been reached. A call to **Tcl_FirstHashEntry** followed by calls to **Tcl_NextHashEntry** will return each of the entries in the table exactly once, in an arbitrary order. It is unadvisable to modify the structure of the table, e.g. by creating or deleting entries, while the search is in progress.

Tcl_HashStats returns a dynamically-allocated string with overall information about a hash table, such as the number of entries it contains, the number of buckets in its hash array, and the utilization of the buckets. It is the caller’s responsibility to free the result string by passing it to **free**.

The header file **tclHash.h** defines the actual data structures used to implement hash tables. This is necessary so that clients can allocate Tcl_HashTable structures and so that macros can be used to read and write the values of entries. However, users of the hashing routines should never refer directly to any of the fields of any of the hash-related data structures; use the procedures and macros defined here.

KEYWORDS

hash table, key, lookup, search, value