

```
/*
 * Generic hashmap manipulation functions
 */
#ifdef __HASHMAP_H__
#define __HASHMAP_H__

#define MAP_MISSING -3 /* No such element */
#define MAP_FULL -2 /* Hashmap is full */
#define MAP_OMEM -1 /* Out of Memory */
#define MAP_OK 0 /* OK */

/*
 * any_t is a pointer. This allows you to put arbitrary structures in
 * the hashmap.
 */
typedef void *any_t;

/*
 * PFany is a pointer to a function that can take two any_t arguments
 * and return an integer. Returns status code..
 */
typedef int (*PFany)(any_t, any_t);

/*
 * map_t is a pointer to an internally maintained data structure.
 * Clients of this package do not need to know how hashmaps are
 * represented. They see and manipulate only map_t's.
 */
typedef any_t map_t;

/*
 * Return an empty hashmap. Returns NULL if empty.
 */
extern map_t hashmap_new();

/*
 * Iteratively call f with argument (item, data) for
 * each element data in the hashmap. The function must
 * return a map status code. If it returns anything other
 * than MAP_OK the traversal is terminated. f must
 * not reenter any hashmap functions, or deadlock may arise.
 */
extern int hashmap_iterate(map_t in, PFany f, any_t item);

/*
 * Add an element to the hashmap. Return MAP_OK or MAP_OMEM.
 */
extern int hashmap_put(map_t in, int key, any_t value);

/*
 * Get an element from the hashmap. Return MAP_OK or MAP_MISSING.
 */
extern int hashmap_get(map_t in, int key, any_t *arg);

/*
 * Remove an element from the hashmap. Return MAP_OK or MAP_MISSING.
 */
extern int hashmap_remove(map_t in, int key);

/*
 * Get any element. Return MAP_OK or MAP_MISSING.
 * remove - should the element be removed from the hashmap
 */
extern int hashmap_get_one(map_t in, any_t *arg, int remove);

/*
```

```
* Free the hashmap
*/
extern void hashmap_free(map_t in);

/*
 * Get the current size of a hashmap
 */
extern int hashmap_length(map_t in);

#endif __HASHMAP_H__
```