6. HASHARRAYS

Hash arrays let you associate arbitrary Lisp objects ("hash keys") with other objects ("hash values"), so you can get from key to value quickly. There are functions for creating hash arrays, putting a hash key/value pair in a hash array, and quickly retrieving the hash value associated with a given hash key.

By default, the hash array functions use EQ for comparing hash keys. This means that if non-symbols are used as hash keys, the exact same object (not a copy) must be used to retrieve the hash value. However, you can specify the function used to compare hash keys and to "hash" a hash key to a number. You can, for example, create hash arrays where EQUAL but non-EQ strings will hash to the same value. Specifying alternative hashing algorithms is described below.

In the description of the functions below, the argument *HARRAY* should be a hasharray created by HASHARRAY. For convenience in interactive program development, it may also be NIL, in which case a hash array (SYSHASHARRAY) provided by the system is used; you must watch out for confusions if this form is used to associate more than one kind of value with the same key.

Note: For backwards compatibility, the hash array functions will accept a list whose CAR is a hash array, and whose CDR is the "overflow method" for the hash array (see below). However, hash array functions are guaranteed to perform with maximum efficiency only if a direct value of HASHARRAY is given.

Note: Interlisp hash arrays and Common Lisp hash tables are the same data type, so functions from both may be intermixed. The only difference between the functions may be argument order, as in MAPHASH and CL:MAPHASH (see below).

(**HASHARRAY** *MINKEYS OVERFLOW HASHBITSFN EQUIVFN RECLAIMABLE REHASH-THRESHOLD*) [Function]

Creates a hash array with space for at least *MINKEYS* hash keys, with overflow method *OVERFLOW*. See discussion of overflow behavior below.

If *HASHBITSFN* and *EQUIVFN* are non-NIL, they specify the hashing function and comparison function used to interpret hash keys. This is described in the section on user-specified hashing functions below. If *HASHBITSFN* and *EQUIVFN* are NIL, the default is to hash EQ hash keys to the same value.

If *RECLAIMABLE* is *T* the entries in the hash table will be removed if the key has a reference count of one and the table is about to be rehashed. This allows the system, in some cases, to reuse keys instead of expanding the table.

Note: CL:MAKE-HASH-TABLE does not allow you to specify your own hashing functions but does provide three built-in types specified by *Common Lisp, the Language*.

(**HARRAY** *MINKEYS*) [Function]

Provided for backward compatibility, this is equivalent to (HASHARRAY MINKEYS 'ERROR), i.e. if the resulting hasarray gets full, an error occurs.

(**HARRAYP** *X*) [Function]

Returns *X* if it is a hash array; otherwise NIL.

HARRAYP returns NIL if *X* is a list whose CAR is an HARRAYP, even though this is accepted by the hash array functions (see below).

(**PUTHASH** *KEY VAL HARRAY*) [Function]

Associates the hash value *VAL* with the hash key *KEY* in *HARRAY*. Replaces the previous hash value, if any. If *VAL* is NIL, any old association is removed (hence a hash value of NIL is not allowed). If *HARRAY* is full when PUTHASH is called with a key not already in the hash array, the function HASHOVERFLOW is called, and the PUTHASH is applied to the value returned (see below). Returns *VAL*.

(**GETHASH** *KEY HARRAY*) [Function]

Returns the hash value associated with the hash key *KEY* in *HARRAY*. Returns NIL, if *KEY* is not found.

(**CLRHASH** *HARRAY*) [Function]

Clears all hash keys/values from *HARRAY*. Returns *HARRAY*.

(**HARRAYPROP** *HARRAY PROP NEWVALUE*) [NoSpread Function]

Returns the property *PROP* of *HARRAY*; *PROP* can have the system-defined values SIZE (the maximum occupancy of *HARRAY*), NUMKEYS (number of occupied slots), OVERFLOW (overflow method), HASHBITSFN (hashing function) and EQUIVFN (comparison function). Except for SIZE and NUMKEYS, a new value may be specified as *NEWVALUE*.

By using other values for *PROP*, the user may also set and get arbitrary property values, to associate additional information with a hash array.

The HASHBITSFN or EQUIVFN properties can only be changed if the hash array is empty.

(**HARRAYSIZE** *HARRAY*) [Function]

Returns the number of slots in *HARRAY*. It's equivalent to (HARRAYPROP HARRAY 'SIZE).

(**REHASH** *OLDHARRAY NEWHARRAY*) [Function]

Hashes all hash keys and values in *OLDHARRAY* into *NEWHARRAY*. The two hash arrays do not have to be (and usually aren't) the same size. Returns *NEWHARRAY*.

(**MAPHASH** *HARRAY MAPHFN*) [Function]

MAPHFN is a function of two arguments. For each hash key in *HARRAY*, *MAPHFN* will be applied to the hash value, and the hash key. For example:

```
[MAPHASH A
   (FUNCTION (LAMBDA (VAL KEY)
        (if (LISTP KEY) then (PRINT VAL)]
```

will print the hash value for all hash keys that are lists. MAPHASH returns *HARRAY*.

Note: the argument order for CL:MAPHASH is *MAPHFN HARRAY*.

(**DMPHASH** *HARRAY HARRAY ... HARRAY*) [NLambda NoSpread Function]

Prints on the primary output file LOADable forms which will restore the hash-arrays contained as the values of the atoms *HARRAY* , *HARRAY* , ... *HARRAY* . Example: (DMPHASH SYSHASHARRAY) will dump the system hash-array.

All EQ identities except symbols and small integers are lost by dumping and loading because READ will create new structure for each item. Thus if two lists contain an EQ substructure, when they are dumped and loaded back in, the corresponding substructures while EQUAL are no longer EQ. The HORRIBLEVARS file package command (Chapter 17) provides a way of dumping hash tables such that these identities are preserved.

Hash Overflow

When a hash array becomes full, trying to add another hash key will cause the function HASHOVERFLOW to be called. This either enlarges the hash array, or causes the error Hash table full. How hash overflow is handled is determined by the value of the OVERFLOW property of the hash array (which can be accessed by HARRAYPROP). The possibilities for the overflow method are:

the symbol ERROR	The error Hash array full is generated when the hash array overflows. This is the default overflow behavior for hash arrays returned by HARRAY.
NIL	The array is automatically enlarged by at least a factor 1.5 every time it overflows. This is the default overflow behavior for hash arrays returned by HASHARRAY.
a positive integer *N*	The array is enlarged to include at least *N* more slots than it currently has.
a floating point number *F*	The array is changed to include *F* times the number of current slots.
a function or lambda expression *FN*	Upon hash overflow, *FN* is called with the hash array as its argument. If *FN* returns a number, that will become the size of the array. Otherwise, the new size defaults to 1.5 times its previous size. *FN* could be used to print a message, or perform some monitor function.

Note: For backwards compatibility, the hash array functions accept a list whose CAR is the hash array, and whose CDR is the overflow method. In this case, the overflow method specified in the list overrides the overflow method set in the hash array. Hash array functions perform with maximum efficiency only if a direct value of HASHARRAY is given.

Specifying Your Own Hashing Functions

In general terms, when a key is looked up in a hash array, it is converted to an integer, which is used to index into a linear array. If the key is not the same as the one found at that index, other indices are

tried until it the desired key is found. The value stored with that key is then returned (from GETHASH) or replaced (from PUTHASH).

To customize hash arrays, you'll need to supply the "hashing function" used to convert a key to an integer and the comparison function used to compare the key found in the array with the key being looked up. For hash arrays to work correctly, any two objects which are equal according to the comparison function must "hash" to equal integers.

By default, Medley uses a hashing function that computes an integer from the internal address of a key, and use EQ for comparing keys. This means that if non-atoms are used as hash keys, *the exact same object* (not a copy) must be used to retrieve the hash value.

There are some applications for which the EQ constraint is too restrictive. For example, it may be useful to use strings as hash keys, without the restriction that EQUAL but not EQ strings are considered to be different hash keys.

The user can override this default behavior for any hash array by specifying the functions used to compare keys and to "hash" a key to a number. This can be done by giving the *HASHBITSFN* and *EQUIVFN* arguments to HASHARRAY (see above).

The *EQUIVFN* argument is a function of two arguments that returns non-NIL when its arguments are considered equal. The *HASHBITSFN* argument is a function of one argument that produces a positive small integer (in the range [0..2^{16} - 1]) with the property that objects that are considered equal by the *EQUIVFN* produce the same hash bits.

For an existing hash array, the function HARRAYPROP (see above) can be used to examine the hashing and equivalence functions as the HASHBITSFN and EQUIVFN hash array properties. These properties are read-only for non-empty hash arrays, as it makes no sense to change the equivalence relationship once some keys have been hashed.

The following function is useful for creating hash arrays that take strings as hash keys:

(**STRINGHASHBITS** *STRING*) [Function]

Hashes the string *STRING* into an integer that can be used as a HASHBITSFN for a hash array. Strings which are STREQUAL hash to the same integer.

Example:

```
(HASHARRAY MINKEYS OVERFLOW 'STRINGHASHBITS 'STREQUAL)
```

creates a hash array where you can use strings as hash keys.

[This page intentionally left blank]