1.   Hash tables and hash functions
1.1. Hash tables

1.1.1. Constructors

```
(make-eq-hash-table)
(make-eq-hash-table k)
```

Returns a newly allocated hash table that accepts arbitrary objects as keys, and compares those keys with `eq?`. If an argument is given, then the initial capacity of the hash table is set to k elements.

```
(make-eqv-hash-table)
(make-eqv-hash-table k)
```

Returns a newly allocated hash table that accepts arbitrary objects as keys, and compares those keys with `eqv?`. If an argument is given, then the initial capacity of the hash table is set to k elements.

```
(make-hash-table procedure₁  procedure₂)
(make-hash-table procedure₁  procedure₂  k)
```

Returns a newly allocated mutable hash table using $procedure_1$ as the hash function and $procedure_2$ as the procedure used to compare keys. The hash function must accept a key and return a non-negative exact integer. If a third argument is given, then the initial capacity of the hash table is set to k elements.

1.1.2. Procedures

```
(hash-table? hash-table)
```
Returns `#t` if *hash-table* was created by one of the hash table constructors, otherwise returns `#f`.

```
(hash-table-size hash-table)
```
Returns the number of keys contained in *hash-table* as an exact integer.

```
(hash-table-ref hash-table key default)
```
Returns the value in *hash-table* associated with *key*. If *hash-table* does not contain an association for *key*, then *default* is returned.

```
(hash-table-set! hash-table key value)
```
Changes *hash-table* to associate *key* with value, replacing any existing association for *key*. Returns the unspecified value.

(`hash-table-delete!` *hash-table key*)
Removes any association for *key* within *hash-table*. Returns the unspecified value.

(`hash-table-contains?` *hash-table key*)
Returns `#t` if *hash-table* contains an association for *key*, otherwise returns `#f`.

(`hash-table-update!` *hash-table key procedure default*)
Equivalent to, but potentially more efficient than:

```
(hash-table-set!
 hash-table key
 (procedure (hash-table-ref
              hash-table key default)))
```


If *hash-table* does not contain an association for *key*, then *default* is passed to *procedure*.

(`hash-table-update!/call` *hash-table key procedure f*)
Equivalent to, but potentially more efficient than:

```
(hash-table-set!
 hash-table key
 (procedure (hash-table-ref/call [TODO: non-existent procedure]
              hash-table key f)))
```


(`hash-table-fold` *hash-table procedure init*)
For every association in *hash-table*, calls *procedure* with three arguments: the association key, the association value, and an accumulated value. The accumulated value is *init* for the first invocation of *procedure*, and for subsequent invocations of *procedure*, it is the return value of the previous invocation of *procedure*. The order of the calls to *procedure* is indeterminate. The return value of `hash-table-fold` is the value of the last invocation of *procedure*. If any side effect is performed on the hash table while a `hash-table-fold` operation is in progress, then the behavior of `hash-table-fold` is unspecified.

(`hash-table-copy` *hash-table*)
(`hash-table-copy` *hash-table immutable*)

Returns a copy of *hash-table*. If the *immutable* argument is provided and is a true value, the returned hash table will be immutable, otherwise it will be mutable.

(`hash-table-clear!` *hash-table*)
Removes all associations from *hash-table*. Returns the unspecified value.

(`hash-table-for-each` *procedure hash-table*)
For every association in *hash-table*, calls *procedure* with two arguments: the association key and the association value. The *procedure* is called once for each association in hash-table. The order of these calls is indeterminate. If any side effect is performed on the hash table while a `hash-table-for-each` operation is in progress, then the behavior of `hash-table-for-each` is unspecified. The return value of `hash-table-for-each` is the unspecified value.

Equivalent to:

```
(hash-table-fold hash-table
                 (lambda (k v a)
                   (procedure k v))
                 (unspecified))
```


(`hash-table->alist` *hash-table*)
Equivalent to:

```
(hash-table-fold hash-table
                 (lambda (k v a)
                   (cons (cons k v) a))
                 '())
```


(`hash-table-keys` *hash-table*)
Equivalent to:

```
(hash-table-fold hash-table
                 (lambda (k v a) (cons k a))
                 '())
```


(`hash-table-values` *hash-table*)
Equivalent to:

```
(hash-table-fold hash-table
                 (lambda (k v a) (cons v a))
                 '())
```

1.1.3. Reflection

(`hash-table-equivalence-predicate` *hash-table*)
Returns the equivalence predicate used by *hash-table* to compare keys. For hash tables created with `make-eq-hash-table` and `make-eqv-hash-table`, returns `eq?` and `eqv?` respectively.

(`hash-table-hash-function` *hash-table*)
Returns the hash function used by hash-table. For hash tables created by `make-eq-hash-table` or `make-eqv-hash-table`, `#f` is returned.

(`hash-table-mutable?` *hash-table*)
Returns `#t` if *hash-table* is mutable, `#f` otherwise.

1.2. Hash functions

(`equal-hash` *obj*)
Returns an integer hash value for *obj*, based on its structure and current contents.

(`string-hash` *string*)
Returns an integer hash value for *string*, based on its current contents.

(`string-ci-hash` *string*)
Returns an integer hash value for *string* based on its current contents, ignoring case.

(`symbol-hash` *symbol*)
Returns an integer hash value for *symbol*.

1.3. Issues

1.3.1. Limit proposal scope

Should the specification be limited to eq and eqv hash tables, since those are the only kind which may not be implementable as a portable library? (Editors straw-poll: no)

1.3.2. Complexity

It may be appropriate to specify constraints on complexity, such as constant time for hash-table-size, or an appropriate constraint on accessor procedures.

1.3.3. Concurrency

R6RS does not deal with concurrency. Even if this proposal does not say anything about that, the issue should be considered. Any implementation that supports concurrency will have to implement some kind of mutual exclusion for operations that have side effects, and some will need mutual exclusion even for `hash-table-ref`. As specified, the updating operations are not atomic, so they create no new problems. The `hash-table-fold` and `hash-table-for-each` procedures already have a problem, even without concurrency.

1.3.4. hash-table-update!

Are the functional update procedures `hash-table-update!` and `hash-table-update!/call` justified?

1.3.5. Side-effects

There is a potential problem with the higher-order procedures, if a procedure argument mutates the hash table being operated on. This should be addressed somehow, if only by a statement that the behavior caused by such procedures is unspecified.

1.3.6. hash-table-map

This has been omitted because a single appropriate specification is not obvious, and any reasonable specification can easily be implemented in terms of `hash-table-fold`.

1.3.7. Names

Should the type name be `hash-table`, `hashtable`, or something else?