Table Library Tutorial

The table library is explained in section 5.4 of the Reference Manual [1]. There are more details about tables in the TablesTutorial.

The manual is concise about the purpose of this library. We'll quote it here:

> Most functions in the table library assume that the table represents an array or a list. For those functions, an important concept is the size of the array. There are three ways to specify that size:
>
> - the field "`n`" - When the table has a field "`n`" with a numerical value, that value is assumed as its size.
> - `setn` - You can call the `table.setn()` function to explicitly set the size of a table.
> - implicit size - Otherwise, the size of the object is one less the first integer index with a `nil` value.
>
> For more details, see the descriptions of the `table.getn()` and `table.setn()` functions.

Note: The size of a table does not necessarily reflect the number of elements contained in the table. This may seem a little strange but it can be useful, for example, for maintaining none sequential lists.

table.concat(table [, sep [, i [, j]]])

Concatenate the elements of a table together to form a string. Each element must be able to be coerced into a string. A separator can be specified which is placed between concatenated elements. Additionally a range can be specified within the table, starting at the `i`-th element and finishing at the `j`-th element.

```
> = table.concat({ 1, 2, "three", 4, "five" })
12three4five
> = table.concat({ 1, 2, "three", 4, "five" }, ", ")
1, 2, three, 4, five
> = table.concat({ 1, 2, "three", 4, "five" }, ", ", 2)
2, three, 4, five
> = table.concat({ 1, 2, "three", 4, "five" }, ", ", 2, 4)
2, three, 4
```

We cannot join tables together because they cannot be coerced into strings. See the StringsTutorial for more information on coercion.

```
> = table.concat({ 1,2,{} })
```

```
stdin:1: bad argument #1 to `concat' (table contains non-strings)
stack traceback:
        [C]: in function `concat'
        stdin:1: in main chunk
        [C]: ?
```

table.foreach(table, f)

Apply the function `f` to the elements of the table passed. On each iteration the function `f` is passed the *key-value* pair of that element in the table.

```
> table.foreach({1,"two",3}, print) -- print the key-value pairs
1       1
2       two
3       3
> table.foreach({1,"two",3,"four"}, function(k,v) print(string.rep(v,k)) end)
1
twotwo
333
fourfourfourfour
```

If the function `f` returns a **non**-`nil` value the iteration loop terminates.

```
> table.foreach({1,"two",3}, function(k,v) print(k,v) return k<2 and nil end)
1       1
2       two
```

Tables can contain mixed *key-value* and *index-value* elements. `table.foreach()` will display all of the elements in a table. To only display the *index-value* elements see `table.foreachi()`. For more information about this subject see the TablesTutorial.

```
> t = { 1,2,"three"; pi=3.14159, banana="yellow" }
> table.foreach(t, print)
1       1
2       2
3       three
pi      3.14159
banana  yellow
```

table.foreachi(table, f)

Apply the function `f` to the elements of the table passed. On each iteration the function `f` is passed the *index-value* pair of that element in the table. This is similar to `table.foreach()` except that *index-value* pairs are passed, not *key-value* pairs. If the function `f` returns a **non**-`nil` value the iteration loop terminates.

```
> t = { 1,2,"three"; pi=3.14159, banana="yellow" }
> table.foreachi(t, print)
1       1
2       2
3       three
```

Note in the example only the indexed elements of the table are displayed. See the TablesTutorial for more information on *key-value* and *index-value* pairs.

table.getn(table)

This is used to determine the size of a table. The size of a table is discussed at the top of this page.

```
> = table.getn({1,2,3})         -- Lua will count the elements if no size is specified
3
> = table.getn({1,2,3; n=10})   -- note, n overrides counting the elements
10
> t = {1,2,3}
> table.setn(t, 10)              -- set our own size with setn()
> = table.getn(t)
10
> = table.getn({1,2,3,nil,5,6}) -- sequence ends at element 3 due to nil value at 4
3
```

table.sort(table [, comp])

Sort the elements of a table in-place (i.e. alter the table).

```
> t = { 3,2,5,1,4 }
> table.sort(t)
> = table.concat(t, ", ")  -- display sorted values
1, 2, 3, 4, 5
```

If the table has a specified size only the range specified is sorted, e.g.,

```
> t = { 3,2,5,1,4; n=3 }   -- construct a table with user size of 3
> table.sort(t)            -- sort will be limited by user size
> = table.concat(t, ", ")  -- only specified size is concatenated as well
2, 3, 5
```

A comparison function can be provided to customise the element sorting. The comparison function must return a boolean value specifying whether the first argument should be before the second argument in the sequence. The default behaviour is for the < comparison to be made. For example, the following behaves

the same as no function being supplied:

```
> t = { 3,2,5,1,4 }
> table.sort(t, function(a,b) return a<b end)
> = table.concat(t, ", ")
1, 2, 3, 4, 5
```

We can see if we reverse the comparison the sequence order is reversed.

```
> table.sort(t, function(a,b) return a>b end)
> = table.concat(t, ", ")
5, 4, 3, 2, 1
```

table.insert(table, [pos,] value)

Insert a given value into a table. If a position is given insert the value before the element currently at that position:

```
> t = { 1,3,"four" }
> table.insert(t, 2, "two")  -- insert "two" at position before element 2
> = table.concat(t, ", ")
1, two, 3, four
```

If no position is specified we append the value to the end of the table:

```
> table.insert(t, 5)         -- no position given so append to end
> = table.concat(t, ", ")
1, two, 3, four, 5
```

When a table has an element inserted both the size of the table and the element indices are updated:

```
> t = { 1,"two",3 }               -- create a table
> = table.getn(t)                 -- find current size
3
> table.foreach(t, print)         -- display the table contents
1       1
2       two
3       3
> table.insert(t, 1, "inserted")  -- insert an element at the start
> = table.concat(t, ", ")         -- see what we have
inserted, 1, two, 3
> = table.getn(t)                 -- find the size
```

```
4
> table.foreach(t, print)         -- the indexes have been updated
1       inserted
2       1
3       two
4       3
```

When no position is specified the element is inserted at the end of the table according to the calculated size. The size of a table may be user specified and not reflect the number of elements, e.g.,

```
> t = { 1,"two",3; n=10 }  -- create a table with user size
> table.insert(t, "end")   -- insert with no position inserts at "end"
> table.foreach(t, print)  -- display the table contents
1       1
2       two
3       3
11      end
n       11
```

table.remove(table [, pos])

Remove an element from a table. If a position is specified the element at that the position is removed. The remaining elements are reindexed sequentially and the size of the table is updated to reflect the change. The element removed is returned by this function. E.g.,

```
> t = { 1,"two",3,"four" }    -- create a table
> = table.getn(t)             -- find the size
4
> table.foreach(t, print)     -- have a look at the elements
1       1
2       two
3       3
4       four
> = table.remove(t,2)         -- remove element number 2 and display it
two
> table.foreach(t, print)     -- display the updated table contents
1       1
2       3
3       four
> = table.getn(t)             -- find the size
3
```

If no position is given remove the last element in the table which is specified by the size of the table. E.g.,

```
> t = { 1,"two","three" }
> = table.getn(t)           -- find the table size (which is removed)
3
> table.foreach(t, print)   -- display contents
1       1
2       two
3       three
> = table.remove(t)         -- remove the element at position "n"
three
> table.foreach(t, print)   -- display updated contents
1       1
2       two
> = table.getn(t)           -- display new size
2
```

If the size of the table does not reflect the number of elements nothing is removed, e.g.,

```
> t = {1,2,3}
> table.setn(t,10)          -- set user size
> table.foreach(t, print)   -- display table contents, note size "n" is stored internally
1       1
2       2
3       3
> = table.getn(t)           -- find the size
10
> = table.remove(t)         -- remove last element
nil
> = table.getn(t)           -- find the updated size
9
> table.foreach(t, print)   -- display elements
1       1
2       2
3       3
```

table.setn(table, n)

Set the size of a table (see notes on table size and top of this page). If the table has a value "`n`" it is updated, e.g.,

```
> t = { 1,"two","three"; n=10 }  -- create a table which has a user size specified
> = table.getn(t)                -- read the size
10
> table.foreach(t, print)        -- display the elements of the table
1       1
```

```
2       two
3       three
n       10
> table.setn(t, 12)              -- use setn to change the size
> = table.getn(t)               -- display the size
12
> table.foreach(t, print)       -- display the table contents
1       1
2       two
3       three
n       12
```

If the table has no element with the key `n` the size of the table is held internally.

```
> t = { 1,"two",3,4 }       -- no "n"
> = table.getn(t)           -- calculate the size of the table
4
> table.setn(t, 10)         -- set the size to a user defined value
> = table.getn(t)           -- find the size
10
> table.foreach(t, print)   -- look at the table contents
1       1
2       two
3       3
4       4
```

FindPage · RecentChanges · preferences
edit · history
Last edited June 15, 2003 12:54 pm PDT (diff)