1 APR::Table -- A Perl API for manipulating opaque string-content table

1.1 SYNOPSIS

```
  use APR::Table;

  $table = make($pool, $nelts);
  $table_copy = $table->copy($pool);

  $table->clear();

  $table->set($key => $val);
  $table->unset($key);
  $table->add($key, $val);

  $val = $table->get($key);
  @val = $table->get($key);

  $table->merge($key => $val);
  overlap($table_a, $table_b, $flags);
  $new_table = overlay($table_base, $table_overlay, $pool);

  $table->do(sub {print "key $_[0], value $_[1]\n"}, @valid_keys);

  #Tied Interface
  $value = $table->{$key};
  $table->{$key} = $value;
  $table->{$key} = $value;
  exists $table->{$key};

  foreach my $key (keys %{$table}) {
      print "$key = $table->{$key}\n";
  }
```

1.2 DESCRIPTION

`APR::Table` allows its users to manipulate opaque string-content tables.

The table's structure is somewhat similar to the Perl's hash structure, but allows multiple values for the same key. An access to the records stored in the table always requires a key.

The key-value pairs are stored in the order they are added.

The keys are case-insensitive.

However as of the current implementation if more than value for the same key is requested, the whole table is lineary searched, which is very inefficient unless the table is very small.

`APR::Table` provides a TIE Interface.

See *apr/include/apr_tables.h* in ASF's *apr* project for low level details.

1.3 API

The variables used in the API definition have the following *"types"*:

- **APR::Table**

  `$table_*`

- **APR::Pool**

  `$pool`

- **scalars: unsigned integers only (SVIV) (as C expects them)**

  `$nelts`, `$flags`

- **scalars: (numerical (SVIV/SVNV) and strings (SVPV))**

  `$key`, `$val`

Function arguments (if any) and return values are shown in the function's synopsis.

- **make()**

  ```
  $table = make($pool, $nelts);
  ```

  Make a new table.

  param `$pool`: The pool to allocate the pool out of.

  param `$nelts`: The number of elements in the initial table.

  return: a new table.

  warning: This table can only store text data

- **copy()**

  ```
  $table_copy = $table->copy($pool);
  ```

  Create a new table and copy another table into it

  param `$pool`: The pool to allocate the new table out of

  param `$table`: The table to copy

  return: A copy of the table passed in

- **clear()**

```
$table->clear();
```

Delete all of the elements from a table.

param `$table`: A copy of the table passed in

- **set();**

  ```
  $table->set($key => $val);
  ```

  Add a key/value pair to a table, if another element already exists with the same key, this will over-write the old data.

  param `$table`: The table to add the data to.

  param `$key`: The key fo use.

  param `$val`: The value to add.

- **add()**

  ```
  $table->add($key, $val);
  ```

  Add data to a table, regardless of whether there is another element with the same key.

  param `$table`: The table to add to

  param `$key`: The key to use

  param `$val`: The value to add.

- **do()**

  ```
  $table->do(sub {[...]}, [@filter]);
  ```

  Iterate over all the elements of the table, invoking provided subroutine for each element. The subroutine gets passed as argument, a key-value pair.

  The subroutine can abort the iteration by returning 0 and should always return 1 otherwise.

  param `sub`: A subroutine reference or name to be called on each item in the table

  param `@filter`: Only keys matching one of the entries in the filter will be processed

- **get()**

  ```
  $val = $table->get($key);
  @val = $table->get($key);
  ```

  Get the value(s) associated with a given key.

After this call, the data is still in the table.

param `$table`: The table to search for the key

param `$key`: The key to search for

return: In the scalar context the first matching value returned. (The oldest in the table, if there is more than one value.) In the list context the whole table is traversed and all matching values are returned. If nothing matches *undef* is returned.

- **unset();**

  ```
  $table->unset($key);
  ```

  Remove data from the table

  param `$table`: The table to remove data from

  param `$key`: The key of the data being removed

- **merge()**

  ```
  $table->merge($key => $val);
  ```

  Add data to a table by merging the value with data that has already been stored

  param `$table`: The table to search for the data

  param `$key`: The key to merge data for

  param `$val`: The data to add

  remark: If the key is not found, then this function acts like add()

- **overlap()**

  ```
  overlap($table_a, $table_b, $flags);
  ```

  For each key/value pair in `$table_b`, add the data to `$table_a`. The definition of `$flags` explains how `$flags` define the overlapping method.

  param `$table_a`: The table to add the data to.

  param `$table_b`: The table to iterate over, adding its data to `%table_a`.

  param `$flags`: How to add the `$table_b` to `$table_a`.

  When `$flags == APR::OVERLAP_TABLES_SET`, if another element already exists with the same key, this will over-write the old data.

When `$flags == APR::OVERLAP_TABLES_MERGE`, the key/value pair from `$table_b` is added, regardless of whether there is another element with the same key in `$table_a`.

remark: This function is highly optimized, and uses less memory and CPU cycles than a function that just loops through table b calling other functions.

- **overlay()**

  ```
  $new_table = overlay($table_base, $table_overlay, $pool);
  ```

  Merge two tables into one new table. The resulting table may have more than one value for the same key.

  param `$pool`: The pool to use for the new table

  param `$table_overlay`: The first table to put in the new table

  param `$table_base`: The table to add at the end of the new table

  return: A new table containing all of the data from the two passed in

- **compress()**

  ```
  compress($table, $flag);
  ```

  Converts multi-valued keys in `$table` to single-valued keys. This function takes duplicate table entries and flattens them into a single entry. The flattening behavior is controlled by the (mandatory) flag.

  param `$table`: The table to add the data to.

  param `$flag`: How to compress `$table`.

  When `$flag` == `APR::OVERLAP_TABLES_SET`, each key will be set to the last value seen for that key. For example, given key/value pairs ’foo => bar’ and ’foo => baz’, ’foo’ would have a final value of ’baz’ after compression - the ’bar’ value would be lost.

  When `$flag` == `APR::OVERLAP_TABLES_MERGE`, multiple values for the same key are flattened into a comma-separated list. Given key/value pairs ’foo => bar’ and ’foo => baz’, ’foo’ would have a final value of ’bar, baz’ after compression.

1.3.1 TIE Interface

`APR::Table` also implements a tied interface, so you can work with the `$table` object as a hash reference.

The following tied-hash function are supported: FETCH, STORE, DELETE, CLEAR, EXISTS, FIRSTKEY, NEXTKEY and DESTROY.

remark: `APR::Table` can hold more than one key-value pair sharing the same key, so when using a table through the tied interface, the first entry found with the right key will be used, completely disregarding possible other entries with the same key. The only exception to this is if you iterate over the list with *each*, then you can access all key-value pairs that share the same key.

Table of Contents:

1 APR::Table -- A Perl API for manipulating opaque string-content table 1
1.1 SYNOPSIS 2
1.2 DESCRIPTION 2
1.3 API 3
1.3.1 TIE Interface 6