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

1 APR::Table - Perl API for 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

`APR::Table` provides the following functions and/or methods:

1.3.1 `add`

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

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

- **arg1: `$t` (`APR::Table`)**

  The table to add to.

- **arg2: `$key` (string)**

  The key to use.

- **arg3: `$val` (string)**

  The value to add.

- **ret: no return value**

When adding data, this function makes a copy of both the key and the value.

1.3.2 `clear`

Delete all of the elements from a table.

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

- **arg1: `$t` (`APR::Table`)**

  The table to clear.

- **ret: no return value**

1.3.3 `compress`

Eliminate redundant entries in a table by either overwriting or merging duplicates:

```
$t->compress($flags);
```

- **arg1: `$t` (`APR::Table`)**

  The table to compress.

- **arg2: `$flags` (integer)**

```
APR::OVERLAP_TABLES_MERGE -- to merge
APR::OVERLAP_TABLES_SET   -- to overwrite
```

- **ret: no return value**

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

When `$flags` == `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 `$flags` == `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.

1.3.4 copy

Create a new table and copy another table into it.

```
$ret = $t->copy($p);
```

- **arg1: `$t` (`APR::Table`)**

  The table to copy.

- **arg2: `$p` (`APR::Pool`)**

  The pool to allocate the new table out of.

- **ret: `$ret` (`APR::Table`)**

  A copy of the table passed in.

1.3.5 do

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

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

- **arg1: `$p` (`APR::Table`)**

  The table to operate on.

- **arg2: `$sub` (CODE ref/string)**

  A subroutine reference or name to be called on each item in the table. The subroutine can abort the iteration by returning 0 and should always return 1 otherwise.

- **opt arg3: `@filter` (ARRAY)**

  If passed, only keys matching one of the entries in the `@filter` will be processed.

- **ret: no return value**

1.3.6 `get`

Get the value(s) associated with a given key. After this call, the data is still in the table.

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

- **arg1: `$t` (`APR::Table`)**

  The table to search for the key.

- **arg2: `$key` (string)**

  The key to search for.

- **ret: `$val` or `@val`**

  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.

1.3.7 `make`

Make a new table.

```
$ret = make($p, $nelts);
```

- **arg1: `$p` (`APR::Pool`)**

  The pool to allocate the pool out of.

- **arg2: `$nelts` (integer)**

  The number of elements in the initial table.

- **ret: `$ret` (`APR::Table`)**

  The new table.

Note: This table can only store text data.

1.3.8 `merge`

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

```
$t->merge($key, $val);
```

- **arg1: `$t` (`APR::Table`)**

  The table to search for the data.

- **arg2: `$key` (string)**

  The key to merge data for.

- **arg3: `$val` (string)**

  The data to add.

- **ret: no return value**

Note: if the key is not found, then this function acts like `add()`.

1.3.9 `overlap`

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

```
$t_a->overlap($t_b, $flags);
```

- **arg1: `$t_a` (`APR::Table`)**

  The table to add the data to.

- **arg2: `$t_b` (`APR::Table`)**

  The table to iterate over, adding its data to table `$t_a`

- **arg3: `$flags` (integer)**

  How to add the table to table `$t_a`.

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

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

- **ret: no return value**

This function is highly optimized, and uses less memory and CPU cycles than a function that just loops through table `$t_b` calling other functions.

Conceptually, `overlap()` does this:

```
apr_array_header_t *barr = apr_table_elts(b);
apr_table_entry_t *belt = (apr_table_entry_t *)barr-E<gt>elts;
int i;

for (i = 0; i E<lt> barr-E<gt>nelts; ++i) {
    if (flags & APR_OVERLAP_TABLES_MERGE) {
        apr_table_mergen(a, belt[i].key, belt[i].val);
    }
    else {
        apr_table_setn(a, belt[i].key, belt[i].val);
    }
}
```

Except that it is more efficient (less space and cpu-time) especially when `$t_b` has many elements.

Notice the assumptions on the keys and values in `$t_b` -- they must be in an ancestor of `$t_a`'s pool. In practice `$t_b` and `$t_a` are usually from the same pool.

1.3.10 overlay

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

```
$t = $t_base->overlay($t_overlay, $p);
```

- **arg1: `$t_base` (`APR::Table`)**

  The table to add at the end of the new table.

- **arg2: `$t_overlay` (`APR::Table`)**

  The first table to put in the new table.

- **arg3: `$p` (`APR::Pool`)**

  The pool to use for the new table.

- **ret: `$t` (`APR::Table`)**

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

1.3.11 set

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

```
$t->set($key, $val);
```

- **arg1: $t (APR::Table)**

  The table to add the data to.

- **arg2: $key (string)**

  The key to use.

- **arg3: $val (string)**

  The value to add.

- **ret: no return value**

When adding data, this function makes a copy of both the key and the value.

1.3.12 unset

Remove data from the table.

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

- **arg1: $t (APR::Table)**

  The table to remove data from.

- **arg2: $key (string)**

  The key of the data being removed.

- **ret: no return value**

1.4 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.

1.4.1 EXISTS

```
$ret = $t->EXISTS($key);
```

- **arg1: `$t` (`APR::Table`)**
- **arg2: `$key` (string)**
- **ret: `$ret` (integer)**

1.4.2 CLEAR

```
$t->CLEAR();
```

- **arg1: `$t` (`APR::Table`)**
- **ret: no return value**

1.4.3 STORE

```
$t->STORE($key, $value);
```

- **arg1: `$t` (`APR::Table`)**
- **arg2: `$key` (string)**
- **arg3: `$value` (string)**
- **ret: no return value**

1.4.4 DELETE

```
$t->DELETE($key);
```

- **arg1: `$t` (`APR::Table`)**
- **arg2: `$key` (string)**
- **ret: no return value**

1.4.5 FETCH

```
$ret = $t->FETCH($key);
```

- **arg1: `$t` (`APR::Table`)**
- **arg2: `$key` (string)**
- **ret: `$ret` (string)**

1.5 See Also

mod_perl 2.0 documentation.

1.6 Copyright

1.7 Authors

The mod_perl development team and numerous contributors.

Table of Contents: