## NAME

TAP::Parser::Aggregator - Aggregate TAP::Parser results

## VERSION

Version 3.17

## SYNOPSIS

```
    use TAP::Parser::Aggregator;

    my $aggregate = TAP::Parser::Aggregator->new;
    $aggregate->add( 't/00-load.t', $load_parser );
    $aggregate->add( 't/10-lex.t',  $lex_parser  );

    my $summary = <<'END_SUMMARY';
    Passed:  %s
    Failed:  %s
    Unexpectedly succeeded: %s
    END_SUMMARY
    printf $summary,
           scalar $aggregate->passed,
           scalar $aggregate->failed,
           scalar $aggregate->todo_passed;
```

## DESCRIPTION

`TAP::Parser::Aggregator` collects parser objects and allows reporting/querying their aggregate results.

## METHODS

### Class Methods

#### new

```
 my $aggregate = TAP::Parser::Aggregator->new;
```

Returns a new `TAP::Parser::Aggregator` object.

### Instance Methods

#### add

```
  $aggregate->add( $description => $parser );
```

The `$description` is usually a test file name (but only by convention.) It is used as a unique identifier (see e.g. *parsers*.) Reusing a description is a fatal error.

The `$parser` is a *TAP::Parser* object.

#### parsers

```
  my $count   = $aggregate->parsers;
  my @parsers = $aggregate->parsers;
  my @parsers = $aggregate->parsers(@descriptions);
```

In scalar context without arguments, this method returns the number of parsers aggregated. In list context without arguments, returns the parsers in the order they were added.

If `@descriptions` is given, these correspond to the keys used in each call to the add() method. Returns an array of the requested parsers (in the requested order) in list context or an array reference

in scalar context.

Requesting an unknown identifier is a fatal error.

**descriptions**

Get an array of descriptions in the order in which they were added to the aggregator.

**start**

Call `start` immediately before adding any results to the aggregator. Among other times it records the start time for the test run.

**stop**

Call `stop` immediately after adding all test results to the aggregator.

**elapsed**

Elapsed returns a *Benchmark* object that represents the running time of the aggregated tests. In order for `elapsed` to be valid you must call `start` before running the tests and `stop` immediately afterwards.

**elapsed_timestr**

Returns a formatted string representing the runtime returned by `elapsed()`. This lets the caller not worry about Benchmark.

**all_passed**

Return true if all the tests passed and no parse errors were detected.

**get_status**

Get a single word describing the status of the aggregated tests. Depending on the outcome of the tests returns 'PASS', 'FAIL' or 'NOTESTS'. This token is understood by *CPAN::Reporter*.

## Summary methods

Each of the following methods will return the total number of corresponding tests if called in scalar context. If called in list context, returns the descriptions of the parsers which contain the corresponding tests (see `add` for an explanation of description.

* failed
* parse_errors
* passed
* planned
* skipped
* todo
* todo_passed
* wait
* exit

For example, to find out how many tests unexpectedly succeeded (TODO tests which passed when they shouldn't):

```
my $count        = $aggregate->todo_passed;
my @descriptions = $aggregate->todo_passed;
```

Note that `wait` and `exit` are the totals of the wait and exit statuses of each of the tests. These values are totalled only to provide a true value if any of them are non-zero.

**total**

```
my $tests_run = $aggregate->total;
```

Returns the total number of tests run.

**has_problems**

```
if ( $parser->has_problems ) {
    ...
}
```

Identical to `has_errors`, but also returns true if any TODO tests unexpectedly succeeded. This is more akin to "warnings".

**has_errors**

```
if ( $parser->has_errors ) {
    ...
}
```

Returns true if *any* of the parsers failed. This includes:

* Failed tests

* Parse errors

* Bad exit or wait status

**todo_failed**

```
  # deprecated in favor of 'todo_passed'.  This method was horribly
misnamed.
```

This was a badly misnamed method. It indicates which TODO tests unexpectedly succeeded. Will now issue a warning and call `todo_passed`.

## See Also

*TAP::Parser*

*TAP::Harness*