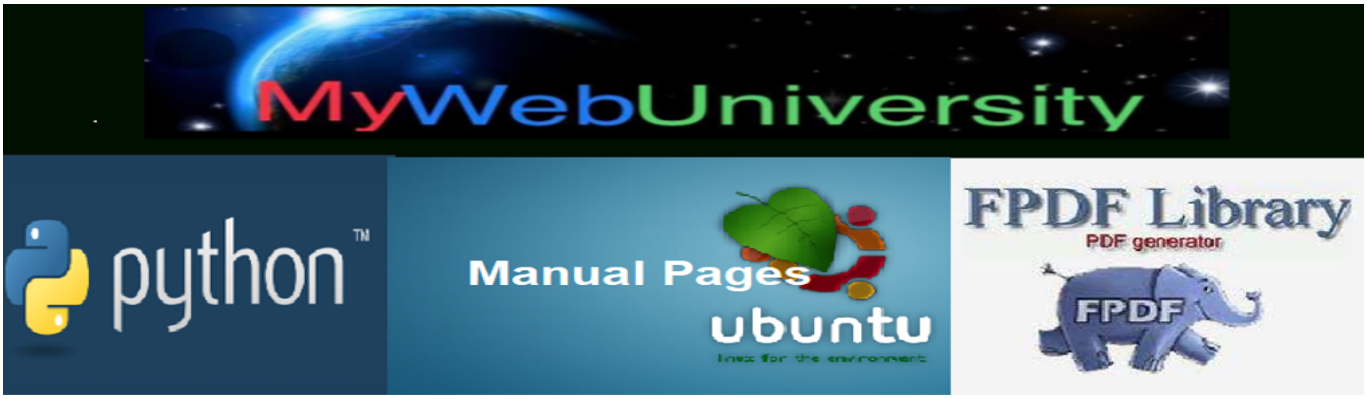

Rocky Enterprise Linux 9.2 Manual Pages on command 'Test2::API::InterceptResult::Event.3perl'

$ man Test2::API::InterceptResult::Event.3perl

Test2::API::InterceptResult::EvenPerleProgrammers RefereTest2::API::InterceptResult::Event(3perl)

NAME

Test2::API::InterceptResult::Event - Representation of an event for use in testing other test tools.

DESCRIPTION

"intercept { ... }" from Test2::API returns an instance of Test2::API::InterceptResult which is a blessed arrayref of Test2::API::InterceptResult::Event objects.

This POD documents the methods of these events, which are mainly provided for you to use when testing your test tools.

SYNOPSIS

```
use Test2::V0;
use Test2::API qw/intercept/;
my $events = intercept {
    ok(1, "A passing assertion");
    plan(1);
};
# This will convert all events into instances of
# Test2::API::InterceptResult::Event. Until we do this they are the
# original Test::Event::* instances
$events->upgrade(in_place => 1);
# Now we can get individual events in this form
my $assert = $events->[0];
my $plan   = $events->[1];
```

```
    # Or we can operate on all events at once:
    my $flattened = $events->flatten;
    is(
        $flattened,
        [
          {
            causes_failure => 0,
            name => 'A passing assertion',
            pass => 1,
            trace_file => 'xxx.t',
            trace_line => 5,
          },
          {
            causes_failure => 0,
            plan => 1,
            trace_file => 'xxx.t',
            trace_line => 6,
          },
        ],
        "Flattened both events and returned an arrayref of the results
    );
```

METHODS

!!! IMPORTANT NOTES ON DESIGN !!!

Please pay attention to what these return, many return a scalar when applicable or an empty list when not (as opposed to undef). Many also always return a list of 0 or more items. Some always return a scalar. Note that none of the methods care about context, their behavior is consistent regardless of scalar, list, or void context.

This was done because this class was specifically designed to be used in a list and generate more lists in bulk operations. Sometimes in a map you want nothing to show up for the event, and you do not want an undef in its place. In general single event instances are not going to be used alone, though that is allowed.

As a general rule any method prefixed with "the_" implies the event should have exactly 1 of the specified item, and and exception will be thrown if there are 0, or more than 1 of

the item.

ATTRIBUTES

$hashref = $event->facet_data

This will return the facet data hashref, which is all Test2 cares about for any given event.

$class = $event->result_class

This is normally Test2::API::InterceptResult. This is set at construction so that subtest results can be turned into instances of it on demand.

DUPLICATION

$copy = $event->clone

Create a deep copy of the event. Modifying either event will not effect the other.

CONDENSED MULTI-FACET DATA

$bool = $event->causes_failure

$bool = $event->causes_fail

These are both aliases of the same functionality.

This will always return either a true value, or a false value. This never returns a list.

This method may be relatively slow (still super fast) because it determines pass or fail by creating an instance of Test2::Hub and asking it to process the event, and then asks the hub for its pass/fail state. This is slower than bulding in logic to do the check, but it is more reliable as it will always tell you what the hub thinks, so the logic will never be out of date relative to the Test2 logic that actually cares.

STRING_OR_EMPTY_LIST = $event->brief

Not all events have a brief, some events are not rendered by the formatter, others have no "brief" data worth seeing. When this is the case an empty list is returned.

This is done intentionally so it can be used in a map operation without having "undef" being included in the result.

When a brief can be generated it is always a single 1-line string, and is returned as-is, not in a list.

Possible briefs:

```
# From control facets
"BAILED OUT"
"BAILED OUT: $why"
```

```
        # From error facets
        "ERROR"
        "ERROR: $message"
        "ERROR: $partial_message [...]"
        "ERRORS: $first_error_message [...]"
        # From assert facets
        "PASS"
        "FAIL"
        "PASS with amnesty"
        "FAIL with amnesty"
        # From plan facets
        "PLAN $count"
        "NO PLAN"
        "SKIP ALL"
        "SKIP ALL: $why"
```

Note that only the first applicable brief is returned. This is essnetially a poor-mans TAP that only includes facets that could (but not necessarily do) cause a failure.

$hashref = $event->flatten

$hashref = $event->flatten(include_subevents => 1)

This ALWAYS returns a hashref. This puts all the most useful data for the most interesting facets into a single hashref for easy validation.

If there are no meaningful facets this will return an empty hashref.

If given the 'include_subevents' parameter it will also include subtest data:

Here is a list of EVERY possible field. If a field is not applicable it will not be present.

always present

```
        causes_failure => 1,    # Always present
```

Present if the event has a trace facet

```
        trace_line    => 42,
        trace_file    => 'Foo/Bar.pm',
        trace_details => 'Extra trace details',    # usually not present
```

If an assertion is present

```
        pass => 0,
```

```
        name => "1 + 1 = 2, so math works",
```

If a plan is present:

```
        plan => $count_or_SKIP_ALL_or_NO_PLAN,
```

If amnesty facets are present

    You get an array for each type that is present.

```
        todo => [    # Yes you could be under multiple todos, this will list them all.
            "I will fix this later",
            "I promise to fix these",
        ],
        skip => ["This will format the main drive, do not run"],
        ... => ["Other amnesty"]
```

If Info (note/diag) facets are present

    You get an arrayref for any that are present, the key is not defined if they are
    not present.

```
        diag => [
            "Test failed at Foo/Bar.pm line 42",
            "You forgot to tie your boots",
        ],
        note => ["Your boots are red"],
        ...  => ["Other info"],
```

If error facets are present

    Always an arrayref

```
        error => [
            "non fatal error (does not cause test failure, just an FYI",
            "FATAL: This is a fatal error (causes failure)",
        ],
        # Errors can have alternative tags, but in practice are always 'error',
        # listing this for completeness.
        ... => [ ... ]
```

Present if the event is a subtest

```
        subtest => {
            count      => 2,    # Number of assertions made
            failed     => 1,    # Number of test failures seen
```

```
            is_passing => 0,    # Boolean, true if the test would be passing
                                # after the events are processed.
            plan         => 2,  # Plan, either a number, undef, 'SKIP', or 'NO PLAN'
            follows_plan => 1,  # True if there is a plan and it was followed.
                                # False if the plan and assertions did not
                                # match, undef if no plan was present in the
                                # event list.
            bailed_out => "foo",    # if there was a bail-out in the
                                    # events in this will be a string explaining
                                    # why there was a bailout, if no reason was
                                    # given this will simply be set to true (1).
            skip_reason => "foo",   # If there was a skip_all this will give the
                                    # reason.
        },
```

if "(include_subtest => 1)" was provided as a parameter then the following will be included. This is the result of turning all subtest child events into an Test2::API::InterceptResult instance and calling the "flatten" method on it.

```
        subevents => Test2::API::InterceptResult->new(@child_events)->flatten(...),
```

If a bail-out is being requested

If no reason was given this will be set to 1.

```
        bailed_out => "reason",
```

$hashref = $event->summary()

This returns a limited summary. See "flatten()", which is usually a better option.

```
    {
        brief => $event->brief || '',
        causes_failure => $event->causes_failure,
        trace_line    => $event->trace_line,
        trace_file    => $event->trace_file,
        trace_tool    => $event->trace_subname,
        trace_details => $event->trace_details,
        facets => [ sort keys(%{$event->{+FACET_DATA}}) ],
    }
```

@list_of_facets = $event->facet($name)

This always returns a list of 0 or more items. This fetches the facet instances from the event. For facets like 'assert' this will always return 0 or 1 item. For events like 'info' (diags, notes) this will return 0 or more instances, once for each instance of the facet.

These will be blessed into the proper Test2::EventFacet subclass. If no subclass can be found it will be blessed as an Test2::API::InterceptResult::Facet generic facet class.

$undef_or_facet = $event->the_facet($name)

If you know you will have exactly 1 instance of a facet you can call this.

If you are correct and there is exactly one instance of the facet it will always return the hashref.

If there are 0 instances of the facet this will reutrn undef, not an empty list.

If there are more than 1 instance this will throw an exception because your assumption was incorrect.

TRACE FACET

@list_of_facets = $event->trace

TODO

$undef_or_hashref = $event->the_trace

This returns the trace hashref, or undef if it is not present.

$undef_or_arrayref = $event->frame

If a trace is present, and has a caller frame, this will be an arrayref:

[$package, $file, $line, $subname]

If the trace is not present, or has no caller frame this will return undef.

$undef_or_string = $event->trace_details

This is usually undef, but occasionally has a string that overrides the file/line number debugging a trace usually provides on test failure.

$undef_or_string = $event->trace_package

Same as "(caller())[0]", the first element of the trace frame.

Will be undef if not present.

$undef_or_string = $event->trace_file

Same as "(caller())[1]", the second element of the trace frame.

Will be undef if not present.

$undef_or_integer = $event->trace_line

Same as "(caller())[2]", the third element of the trace frame.

Will be undef if not present.

$undef_or_string = $event->trace_subname

$undef_or_string = $event->trace_tool

Aliases for the same thing

Same as "(caller($level))[4]", the fourth element of the trace frame.

Will be undef if not present.

$undef_or_string = $event->trace_signature

A string that is a unique signature for the trace. If a single context generates multiple events they will all have the same signature. This can be used to tie assertions and diagnostics sent as seperate events together after the fact.

ASSERT FACET

$bool = $event->has_assert

Returns true if the event has an assert facet, false if it does not.

$undef_or_hashref = $event->the_assert

Returns the assert facet if present, undef if it is not.

@list_of_facets = $event->assert

TODO

EMPTY_LIST_OR_STRING = $event->assert_brief

Returns a string giving a brief of the assertion if an assertion is present. Returns an empty list if no assertion is present.

SUBTESTS (PARENT FACET)

$bool = $event->has_subtest

True if a subetest is present in this event.

$undef_or_hashref = $event->the_subtest

Get the one subtest if present, otherwise undef.

@list_of_facets = $event->subtest

TODO

EMPTY_LIST_OR_OBJECT = $event->subtest_result

Returns an empty list if there is no subtest.

Get an instance of Test2::API::InterceptResult representing the subtest.

CONTROL FACET (BAILOUT, ENCODING)

$bool = $event->has_bailout

True if there was a bailout

$undef_hashref = $event->the_bailout

Return the control facet if it requested a bailout.

EMPTY_LIST_OR_HASHREF = $event->bailout

Get a list of 0 or 1 hashrefs. The hashref will be the control facet if a bail-out was requested.

EMPTY_LIST_OR_STRING = $event->bailout_brief

Get the brief of the balout if present.

EMPTY_LIST_OR_STRING = $event->bailout_reason

Get the reason for the bailout, an empty string if no reason was provided, or an empty list if there was no bailout.

PLAN FACET

TODO

$bool = $event->has_plan

$undef_or_hashref = $event->the_plan

@list_if_hashrefs = $event->plan

EMPTY_LIST_OR_STRING $event->plan_brief

AMNESTY FACET (TODO AND SKIP)

TODO

$event->has_amnesty

$event->the_amnesty

$event->amnesty

$event->amnesty_reasons

$event->has_todos

$event->todos

$event->todo_reasons

$event->has_skips

$event->skips

$event->skip_reasons

$event->has_other_amnesty

$event->other_amnesty

$event->other_amnesty_reasons

ERROR FACET (CAPTURED EXCEPTIONS)

TODO

$event->has_errors

$event->the_errors

$event->errors

$event->error_messages

$event->error_brief

INFO FACET (DIAG, NOTE)

TODO

$event->has_info

$event->the_info

$event->info

$event->info_messages

$event->has_diags

$event->diags

$event->diag_messages

$event->has_notes

$event->notes

$event->note_messages

$event->has_other_info

$event->other_info

$event->other_info_messages

SOURCE

The source code repository for Test2 can be found at

http://github.com/Test-More/test-more/.

MAINTAINERS

Chad Granum <exodist@cpan.org>

AUTHORS

Chad Granum <exodist@cpan.org>

COPYRIGHT

See http://dev.perl.org/licenses/