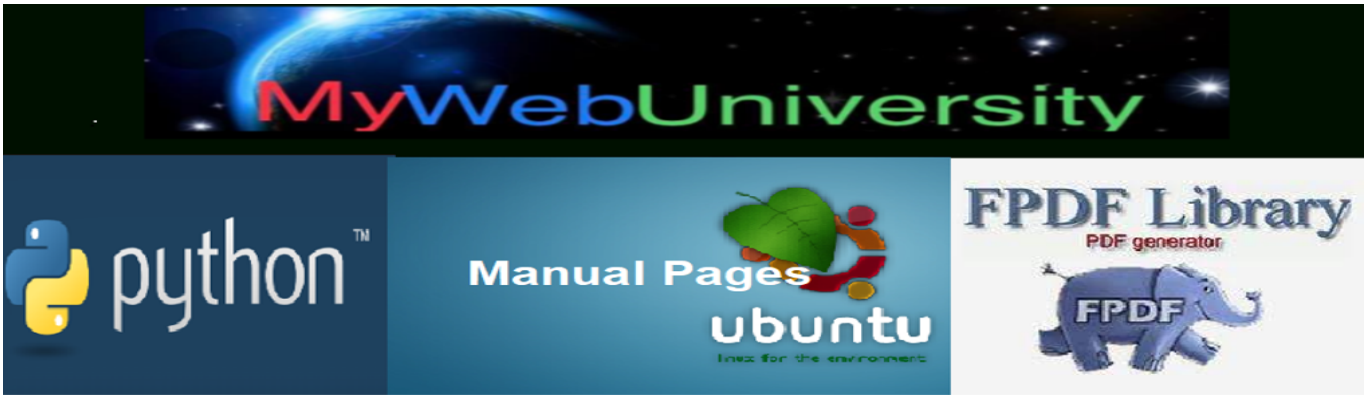

Rocky Enterprise Linux 9.2 Manual Pages on command 'Test2::Hub.3perl'

$ man Test2::Hub.3perl

Test2::Hub(3perl) Perl Programmers Reference Guide Test2::Hub(3perl)

NAME

Test2::Hub - The conduit through which all events flow.

SYNOPSIS

```
use Test2::Hub;
my $hub = Test2::Hub->new();
$hub->send(...);
```

DESCRIPTION

The hub is the place where all events get processed and handed off to the formatter. The hub also tracks test state, and provides several hooks into the event pipeline.

COMMON TASKS

SENDING EVENTS

```
$hub->send($event)
```

The "send()" method is used to issue an event to the hub. This method will handle thread/fork sync, filters, listeners, TAP output, etc.

ALTERING OR REMOVING EVENTS

You can use either "filter()" or "pre_filter()", depending on your needs. Both have identical syntax, so only "filter()" is shown here.

```
$hub->filter(sub {
    my ($hub, $event) = @_;
    my $action = get_action($event);
    # No action should be taken
    return $event if $action eq 'none';
```

```
        # You want your filter to remove the event
        return undef if $action eq 'delete';
        if ($action eq 'do_it') {
            my $new_event = copy_event($event);
            ... Change your copy of the event ...
            return $new_event;
        }
        die "Should not happen";
    });
```

By default, filters are not inherited by child hubs. That means if you start a subtest, the subtest will not inherit the filter. You can change this behavior with the "inherit" parameter:

```
    $hub->filter(sub { ... }, inherit => 1);
```

LISTENING FOR EVENTS

```
    $hub->listen(sub {
        my ($hub, $event, $number) = @_;
        ... do whatever you want with the event ...
        # return is ignored
    });
```

By default listeners are not inherited by child hubs. That means if you start a subtest, the subtest will not inherit the listener. You can change this behavior with the "inherit" parameter:

```
    $hub->listen(sub { ... }, inherit => 1);
```

POST-TEST BEHAVIORS

```
    $hub->follow_up(sub {
        my ($trace, $hub) = @_;
        ... do whatever you need to ...
        # Return is ignored
    });
```

follow_up subs are called only once, either when done_testing is called, or in an END block.

SETTING THE FORMATTER

By default an instance of Test2::Formatter::TAP is created and used.

```
my $old = $hub->format(My::Formatter->new);
```

Setting the formatter will REPLACE any existing formatter. You may set the formatter to undef to prevent output. The old formatter will be returned if one was already set. Only one formatter is allowed at a time.

METHODS

$hub->send($event)

This is where all events enter the hub for processing.

$hub->process($event)

This is called by send after it does any IPC handling. You can use this to bypass the IPC process, but in general you should avoid using this.

$old = $hub->format($formatter)

Replace the existing formatter instance with a new one. Formatters must be objects that implement a "$formatter->write($event)" method.

$sub = $hub->listen(sub { ... }, %optional_params)

You can use this to record all events AFTER they have been sent to the formatter. No changes made here will be meaningful, except possibly to other listeners.

```
$hub->listen(sub {
    my ($hub, $event, $number) = @_;
    ... do whatever you want with the event ...
    # return is ignored
});
```

Normally listeners are not inherited by child hubs such as subtests. You can add the "inherit => 1" parameter to allow a listener to be inherited.

$hub->unlisten($sub)

You can use this to remove a listen callback. You must pass in the coderef returned by the "listen()" method.

$sub = $hub->filter(sub { ... }, %optional_params)

$sub = $hub->pre_filter(sub { ... }, %optional_params)

These can be used to add filters. Filters can modify, replace, or remove events before anything else can see them.

```
$hub->filter(
    sub {
        my ($hub, $event) = @_;
```

```
            return $event;    # No Changes
            return;           # Remove the event
            # Or you can modify an event before returning it.
            $event->modify;
            return $event;
        }
    );
```

If you are not using threads, forking, or IPC then the only difference between a "filter" and a "pre_filter" is that "pre_filter" subs run first. When you are using threads, forking, or IPC, pre_filters happen to events before they are sent to their destination proc/thread, ordinary filters happen only in the destination hub/thread. You cannot add a regular filter to a hub if the hub was created in another process or thread. You can always add a pre_filter.

`$hub->unfilter($sub)`

`$hub->pre_unfilter($sub)`

These can be used to remove filters and pre_filters. The $sub argument is the reference returned by "filter()" or "pre_filter()".

`$hub->follow_op(sub { ... })`

Use this to add behaviors that are called just before the hub is finalized. The only argument to your codeblock will be a Test2::EventFacet::Trace instance.

```
    $hub->follow_up(sub {
        my ($trace, $hub) = @_;
        ... do whatever you need to ...
        # Return is ignored
    });
```

follow_up subs are called only once, ether when done_testing is called, or in an END block.

`$sub = $hub->add_context_acquire(sub { ... });`

Add a callback that will be called every time someone tries to acquire a context. It gets a single argument, a reference of the hash of parameters being used the construct the context. This is your chance to change the parameters by directly altering the hash.

```
    test2_add_callback_context_acquire(sub {
```

```
        my $params = shift;
        $params->{level}++;
    });
```

This is a very scary API function. Please do not use this unless you need to. This is here for Test::Builder and backwards compatibility. This has you directly manipulate the hash instead of returning a new one for performance reasons.

Note Using this hook could have a huge performance impact.

The coderef you provide is returned and can be used to remove the hook later.

```
$hub->remove_context_acquire($sub);
```

This can be used to remove a context acquire hook.

```
$sub = $hub->add_context_init(sub { ... });
```

This allows you to add callbacks that will trigger every time a new context is created for the hub. The only argument to the sub will be the Test2::API::Context instance that was created.

Note Using this hook could have a huge performance impact.

The coderef you provide is returned and can be used to remove the hook later.

```
$hub->remove_context_init($sub);
```

This can be used to remove a context init hook.

```
$sub = $hub->add_context_release(sub { ... });
```

This allows you to add callbacks that will trigger every time a context for this hub is released. The only argument to the sub will be the Test2::API::Context instance that was released. These will run in reverse order.

Note Using this hook could have a huge performance impact.

The coderef you provide is returned and can be used to remove the hook later.

```
$hub->remove_context_release($sub);
```

This can be used to remove a context release hook.

```
$hub->cull()
```

Cull any IPC events (and process them).

```
$pid = $hub->pid()
```

Get the process id under which the hub was created.

```
$tid = $hub->tid()
```

Get the thread id under which the hub was created.

```
$hud = $hub->hid()
```

Get the identifier string of the hub.

$uuid = $hub->uuid()

If UUID tagging is enabled (see Test2::API) then the hub will have a UUID.

$ipc = $hub->ipc()

Get the IPC object used by the hub.

$hub->set_no_ending($bool)

$bool = $hub->no_ending

This can be used to disable auto-ending behavior for a hub. The auto-ending behavior is triggered by an end block and is used to cull IPC events, and output the final plan if the plan was 'NO PLAN'.

$bool = $hub->active

$hub->set_active($bool)

These are used to get/set the 'active' attribute. When true this attribute will force "hub->finalize()" to take action even if there is no plan, and no tests have been run. This flag is useful for plugins that add follow-up behaviors that need to run even if no events are seen.

STATE METHODS

$hub->reset_state()

Reset all state to the start. This sets the test count to 0, clears the plan, removes the failures, etc.

$num = $hub->count

Get the number of tests that have been run.

$num = $hub->failed

Get the number of failures (Not all failures come from a test fail, so this number can be larger than the count).

$bool = $hub->ended

True if the testing has ended. This MAY return the stack frame of the tool that ended the test, but that is not guaranteed.

$bool = $hub->is_passing

$hub->is_passing($bool)

Check if the overall test run is a failure. Can also be used to set the pass/fail status.

$hub->plan($plan)

$plan = $hub->plan

Get or set the plan. The plan must be an integer larger than 0, the string 'NO PLAN', or the string 'SKIP'.

$bool = $hub->check_plan

Check if the plan and counts match, but only if the tests have ended. If tests have not ended this will return undef, otherwise it will be a true/false.

THIRD PARTY META-DATA

This object consumes Test2::Util::ExternalMeta which provides a consistent way for you to attach meta-data to instances of this class. This is useful for tools, plugins, and other extensions.

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