NAME

Mon::Client − Methods for interaction with Mon client

SYNOPSIS

```
use Mon::Client;
```

DESCRIPTION

```
Mon::Client is used to interact with "mon" clients. It supports
a protocol−independent API for retrieving the status of the mon
server, and performing certain operations, such as disableing hosts
and service checks.
```

METHODS

new

Creates a new object. A hash can be supplied which sets the default values. An example which contains all of the variables that you can initialize:

```
$c = new Mon::Client (
    host => "monhost",
    port => 2583,
    username => "foo",
    password => "bar",
);
```

password (pw)

If *pw* is provided, sets the password. Otherwise, returns the currently set password.

host (host)

If *host* is provided, sets the mon host. Otherwise, returns the currently set mon host.

port (portnum)

If *portnum* is provided, sets the mon port number. Otherwise, returns the currently set port number.

username (user)

If *user* is provided, sets the user login. Otherwise, returns the currently set user login.

prot

If *protocol* is provided, sets the protocol, specified by a string which is of the form ‘‘1.2.3’’, where ‘‘1’’ is the major revision, ‘‘2’’ is the minor revision, and ‘‘3’’ is the sub-minor revision. If *protocol* is not provided, the currently set protocol is returned.

protid ([protocol])

Returns true if client and server protocol match, false otherwise. Implicitly called by **connect**. If protocol is specified as an integer, supplies that protocol version to the server for verification.

version

Returns the protocol version of the remote server.

error

Returns the error string from set by the last method, or undef if there was no error.

connected

Returns 0 (not connected) or 1 (connected).

connect (%args)

Connects to the server. If **host** and **port** have not been set, uses the defaults. Returns *undef* on error. If `$args`{‘‘skip_protid’’} is true, skip protocol identification upon connect.

disconnect

Disconnects from the server. Return *undef* on error.

login (`%hash`)

`%hash` is optional, but if specified, should contain two keys, **username** and **password**.

Performs the ‘‘login’’ command to authenticate the user to the server. Uses **username** and **password** if specified, otherwise uses the username and password previously set by those methods, respectively.

checkauth (command)

Checks to see if the specified command, as executed by the current user, is authorized by the server, without actually executing the command. Returns 1 (command is authorized) or 0 (command is not authorized).

disable_watch (watch)

Disables **watch**.

disable_service (watch, service)

Disables a service, as specified by **watch** and **service**.

disable_host (host)

Disables **host**.

enable_watch (watch)

Enables **watch**.

enable_service (watch, service)

Enables a service as specified by **watch** and **service**.

enable_host (host)

Enables **host**.

set (group, service, var, val)

Sets **var** in **group,service** to **val**. Returns undef on error.

get (group, service, var)

Gets variable **var** in **group,service** and returns it, or undef on error.

quit

Logs out of the server. This method should be followed by a call to the **disconnect** method.

list_descriptions

Returns a hash of service descriptions, indexed by watch and service. For example:

```
%desc = $mon->list_descriptions;
print "$desc{'watchname'}->{'servicename'}\n";
```

list_deps

Lists dependency expressions and their components for all services. If there is no dependency for a particular service, then the value will be ‘‘NONE’’.

```
%deps = $mon->list_deps;
foreach $watch (keys %deps) {
    foreach $service (keys %{$deps{$watch}}) {
        my $sref = \%{$deps{$watch}->{$service}};
        print "expr ($watch,$service) = $sref->{expression}\n";
        print "components ($watch,$service) = @{$sref->{components}}\n";
    }
}
```

list_group (hostgroup)

Lists members of **hostgroup**. Returns an array of each member.

list_watch

Returns an array of all the defined watch groups and services.

```
foreach $w ($mon->list_watch) {
    print "group=$w->[0] service=$w->[1]\n";
}
```

list_opstatus ([group1, service1], ...)

Returns a hash of per-service operational statuses, as indexed by watch and service. The list of anonymous arrays is optional, and if is not provided then the status of all groups and services will be queried.

```
%s = $mon−>list_opstatus;
foreach $watch (keys %s) {
    foreach $service (keys %{$s{$watch}}) {
        foreach $var (keys %{$s{$watch}{$service}}) {
            print "$watch $service $var=$s{$watch}{$service}{$var}\n";
        }
    }
}
```

list_failures

Returns a hash in the same manner as **list_opstatus**, but only the services which are in a failure state.

list_successes

Returns a hash in the same manner as **list_opstatus**, but only the services which are in a success state.

list_disabled

Returns a hash of disabled watches, services, and hosts.

```
%d = $mon−>list_disabled;

foreach $group (keys %{$d{"hosts"}}) {
    foreach $host (keys %{$d{"hosts"}{$group}}) {
        print "host $group/$host disabled\n";
    }
}

foreach $watch (keys %{$d{"services"}}) {
    foreach $service (keys %{$d{"services"}{$watch}}) {
        print "service $watch/$service disabled\n";
    }
}

for (keys %{$d{"watches"}}) {
    print "watch $_ disabled\n";
}
```

list_alerthist

Returns an array of hash references containing the alert history.

```
@a = $mon−>list_alerthist;

for (@a) {
    print join (" ",
        $_−>{"type"},
        $_−>{"watch"},
        $_−>{"service"},
        $_−>{"time"},
        $_−>{"alert"},
        $_−>{"args"},
        $_−>{"summary"},
        "\n",
    );
}
```

list_dtlog

Returns an array of hash references containing the downtime log.

`@a` = `$mon`−>list_dtlog

```
 for (@a) {
   print join (" ",
       $_->{"timeup"},
       $_->{"group"},
       $_->{"service"},
       $_->{"failtime"},
       $_->{"downtime"},
       $_->{"interval"},
       $_->{"summary"},
       "\n",
   );
 }
```

list_failurehist

Returns an array of hash references containing the failure history.

```
@f = $mon->list_failurehist;

for (@f) {
    print join (" ",
        $_->{"watch"},
        $_->{"service"},
        $_->{"time"},
        $_->{"summary"},
        "\n",
    );
}
```

list_pids

Returns an array of hash references containing the list of process IDs of currently active monitors run by the server.

```
@p = $mon->list_pids;

$server = shift @p;

for (@p) {
    print join (" ",
        $_->{"watch"},
        $_->{"service"},
        $_->{"pid"},
        "\n",
    );
}
```

list_state

Lists the state of the scheduler. Returns a two-element array. The first element of the array is 0 if the scheduler is stopped, and 1 if the scheduler is currently running. The second element of the array returned is the string ‘‘scheduler running’’ if the scheduler is currently running, and if the scheduler is stopped, the second element is the *time* (2) that the scheduler was stopped.

```
@s = $mon->list_state;

if ($s[0] == 0) {
    print "scheduler stopped since " . localtime ($s[1]) . "\n";
}
```

start

Starts the scheduler.

stop

Stops the scheduler.

reset

Resets the server.

reload (what)

Causes the server to reload its configuration. **what** is an optional argument, and currently the only supported option is **auth**, which reloads the authorization file.

term

Terminates the server.

set_maxkeep

Sets the maximum number of history entries to store in memory.

get_maxkeep

Returns the maximum number of history entries to store in memory.

test (test, group, service [, exitval, period])

Schedules a service test to run immediately, or tests an alert for a given period. **test** must be **monitor**, **alert**, **startupalert**, or **upalert**. To test alerts, the **exitval** and **period** must be supplied. Periods are identified by their label in the mon config file. If there are no period tags, then the actual period string must be used, exactly as it is listed in the config file.

test_config

Tests the syntax of the configuration file. Returns a two-element array. The first element of the array is 0 if the syntax of the config file is invalid, and 1 if the syntax of the config file is OK. The second element of the array returned is the failure message, if the config file has invalid syntax, and the result code if the config file syntax is OK. This function returns undef if it cannot get a connection or a response from the mon server.

Config file checking stops as soon as an error is found, so you will need to run this command more than once if you have multiple errors in your config file in order to find them all.

```
@s = $mon->test_config;

if ($s[0] == 0) {
    print "error in config file:\n" . $s[1] . "\n";
}
```

ack (group, service, text)

When **group/service** is in a failure state, acknowledges this with **text**, and disables all further alerts during this failure period.

loadstate (state)

Loads **state**.

savestate (state)

Saves **state**.

servertime

Returns the time on the server using the same output as the *time*(2) system call.

send_trap (`%vars`)

Sends a trap to a remote mon server. Here is an example:

```
$mon->send_trap (
    group           => "remote-group",
    service         => "remote-service",
    retval          => 1,
    opstatus        => "fail",
    summary         => "hosta hostb hostc",
    detail          => "hosta hostb and hostc are unresponsive",
);
```

retval must be a nonnegative integer.

opstatus must be one of *fail*, *ok*, *coldstart*, *warmstart*, *linkdown*, *unknown*, *timeout*, *untested*.

Returns *undef* on error.