NAME

Net::DBus::Reactor – application event loop

SYNOPSIS

Create and run an event loop:

```
use Net::DBus::Reactor;
my $reactor = Net::DBus::Reactor->main();

$reactor->run();
```

Manage some file handlers

```
$reactor->add_read($fd,
                   Net::DBus::Callback->new(method => sub {
                      my $fd = shift;
                      ...read some data...
                   }, args => [$fd]));

$reactor->add_write($fd,
                    Net::DBus::Callback->new(method => sub {
                       my $fd = shift;
                       ...write some data...
                    }, args => [$fd]));
```

Temporarily (dis|en)able a handle

```
# Disable
$reactor->toggle_read($fd, 0);
# Enable
$reactor->toggle_read($fd, 1);
```

Permanently remove a handle

```
$reactor->remove_read($fd);
```

Manage a regular timeout every 100 milliseconds

```
my $timer = $reactor->add_timeout(100,
                                  Net::DBus::Callback->new(
            method => sub {
               ...process the alarm...
            }));
```

Temporarily (dis|en)able a timer

```
# Disable
$reactor->toggle_timeout($timer, 0);
# Enable
$reactor->toggle_timeout($timer, 1);
```

Permanently remove a timer

```
$reactor->remove_timeout($timer);
```

Add a post-dispatch hook

```
my $hook = $reactor->add_hook(Net::DBus::Callback->new(
      method => sub {
         ... do some work...
      }));
```

Remove a hook

```
$reactor->remove_hook($hook);
```

DESCRIPTION

This class provides a general purpose event loop for the purposes of multiplexing I/O events and timeouts in a single process. The underlying implementation is done using the select system call. File handles can be registered for monitoring on read, write and exception (out-of-band data) events. Timers can be registered to expire with a periodic frequency. These are implemented using the timeout parameter of the select system call. Since this parameter merely represents an upper bound on the amount of time the select system call is allowed to sleep, the actual period of the timers may vary. Under normal load this variance is typically 10 milliseconds. Finally, hooks may be registered which will be invoked on each iteration of the event loop (ie after processing the file events, or timeouts indicated by the select system call returning).

METHODS

my `$reactor` = Net::DBus::Reactor->**new**();

Creates a new event loop ready for monitoring file handles, or generating timeouts. Except in very unusual circumstances (examples of which I can't think up) it is not necessary or desriable to explicitly create new reactor instances. Instead call the main method to get a handle to the singleton instance.

`$reactor` = Net::DBus::Reactor->main;

Return a handle to the singleton instance of the reactor. This is the recommended way of getting hold of a reactor, since it removes the need for modules to pass around handles to their privately created reactors.

`$reactor`->manage($connection);
`$reactor`->manage($server);

Registers a `Net::DBus::Binding::Connection` or `Net::DBus::Binding::Server` object for management by the event loop. This basically involves hooking up the watch & timeout callbacks to the event loop. For connections it will also register a hook to invoke the `dispatch` method periodically.

`$reactor`->**run**();

Starts the event loop monitoring any registered file handles and timeouts. At least one file handle, or timer must have been registered prior to running the reactor, otherwise it will immediately exit. The reactor will run until all registered file handles, or timeouts have been removed, or disabled. The reactor can be explicitly stopped by calling the `shutdown` method.

`$reactor`->**shutdown**();

Explicitly shutdown the reactor after pending events have been processed.

`$reactor`->**step**();

Perform one iteration of the event loop, going to sleep until an event occurs on a registered file handle, or a timeout occurrs. This method is generally not required in day-to-day use.

`$reactor`->add_read($fd, `$callback`[, `$status`]);

Registers a file handle for monitoring of read events. The `$callback` parameter specifies either a code reference to a subroutine, or an instance of the `Net::DBus::Callback` object to invoke each time an event occurs. The optional `$status` parameter is a boolean value to specify whether the watch is initially enabled.

`$reactor`->add_write($fd, `$callback`[, `$status`]);

Registers a file handle for monitoring of write events. The `$callback` parameter specifies either a code reference to a subroutine, or an instance of the `Net::DBus::Callback` object to invoke each time an event occurs. The optional `$status` parameter is a boolean value to specify whether the watch is initially enabled.

`$reactor`->add_exception($fd, `$callback`[, `$status`]);

Registers a file handle for monitoring of exception events. The `$callback` parameter specifies either a code reference to a subroutine, or an instance of the `Net::DBus::Callback` object to invoke each time an event occurs. The optional `$status` parameter is a boolean value to specify whether the

watch is initially enabled.

my $id = $reactor->add_timeout($interval, $callback, $status);
: Registers a new timeout to expire every $interval milliseconds. The $callback parameter specifies either a code reference to a subroutine, or an instance of the Net::DBus::Callback object to invoke each time the timeout expires. The optional $status parameter is a boolean value to specify whether the timeout is initially enabled. The return parameter is a unique identifier which can be used to later remove or disable the timeout.

$reactor->remove_timeout($id);
: Removes a previously registered timeout specified by the $id parameter.

$reactor->toggle_timeout($id, $status[, $interval]);
: Updates the state of a previously registered timeout specified by the $id parameter. The $status parameter specifies whether the timeout is to be enabled or disabled, while the optional $interval parameter can be used to change the period of the timeout.

my $id = $reactor->add_hook($callback[, $status]);
: Registers a new hook to be fired on each iteration of the event loop. The $callback parameter specifies either a code reference to a subroutine, or an instance of the Net::DBus::Callback class to invoke. The $status parameter determines whether the hook is initially enabled, or disabled. The return parameter is a unique id which should be used to later remove, or disable the hook.

$reactor->remove_hook($id)
: Removes the previously registered hook identified by $id.

$reactor->toggle_hook($id, $status)
: Updates the status of the previously registered hook identified by $id. The $status parameter determines whether the hook is to be enabled or disabled.

$reactor->remove_read($fd);
$reactor->remove_write($fd);
$reactor->remove_exception($fd);
: Removes a watch on the file handle $fd.

$reactor->toggle_read($fd, $status);
$reactor->toggle_write($fd, $status);
$reactor->toggle_exception($fd, $status);
: Updates the status of a watch on the file handle $fd. The $status parameter species whether the watch is to be enabled or disabled.

SEE ALSO

Net::DBus::Callback, Net::DBus::Connection, Net::DBus::Server

AUTHOR

Daniel Berrange <dan@berrange.com>

COPYRIGHT