NAME

Net::DBus::Binding::Connection − A connection between client and server

SYNOPSIS

Creating a connection to a server and sending a message

```
use Net::DBus::Binding::Connection;

my $con = Net::DBus::Binding::Connection->new(address => "unix:path=/path/to/so

$con->send($message);
```

Registering message handlers

```
sub handle_something {
    my $con = shift;
    my $msg = shift;

    ... do something with the message...
}

$con->register_message_handler(
  "/some/object/path",
  \&handle_something);
```

Hooking up to an event loop:

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

$reactor->manage($con);

$reactor->run();
```

DESCRIPTION

An outgoing connection to a server, or an incoming connection from a client. The methods defined on this module have a close correspondence to the dbus_connection_XXX methods in the C API, so for further details on their behaviour, the C API documentation may be of use.

METHODS

my $con = Net::DBus::Binding::Connection−>new(address => ‘‘unix:path=/path/to/socket’’);

Creates a new connection to the remove server specified by the parameter `address`. If the `private` parameter is supplied, and set to a True value the connection opened is private; otherwise a shared connection is opened. A private connection must be explicitly shutdown with the `disconnect` method before the last reference to the object is released. A shared connection must never be explicitly disconnected.

$status = $con−>**is_connected()**;

Returns zero if the connection has been disconnected, otherwise a positive value is returned.

$status = $con−>**is_authenticated()**;

Returns zero if the connection has not yet successfully completed authentication, otherwise a positive value is returned.

$con−>**disconnect()**

Closes this connection to the remote host. This method is called automatically during garbage collection (ie in the DESTROY method) if the programmer forgets to explicitly disconnect.

$con−>**flush()**

Blocks execution until all data in the outgoing data stream has been sent. This method will not re-enter the application event loop.

`$con`->send($message)
: Queues a message up for sending to the remote host. The data will be sent asynchronously as the applications event loop determines there is space in the outgoing socket send buffer. To force immediate sending of the data, follow this method will a call to `flush`. This method will return the serial number of the message, which can be used to identify a subsequent reply (if any).

my `$reply` = `$con`->send_with_reply_and_block($msg, `$timeout`);
: Queues a message up for sending to the remote host and blocks until it has been sent, and a corresponding reply received. The return value of this method will be a `Net::DBus::Binding::Message::MethodReturn` or `Net::DBus::Binding::Message::Error` object.

my `$pending_call` = `$con`->send_with_reply($msg, `$timeout`);
: Queues a message up for sending to the remote host and returns immediately providing a reference to a `Net::DBus::Binding::PendingCall` object. This object can be used to wait / watch for a reply. This allows methods to be processed asynchronously.

`$con`->dispatch;
: Dispatches any pending messages in the incoming queue to their message handlers. This method is typically called on each iteration of the main application event loop where data has been read from the incoming socket.

`$message` = `$con`->borrow_message
: Temporarily removes the first message from the incoming message queue. No other thread may access the message while it is 'borrowed', so it should be replaced in the queue with the `return_message` method, or removed permanently with th `steal_message` method as soon as is practical.

`$con`->return_message($msg)
: Replaces a previously borrowed message in the incoming message queue for subsequent dispatch to registered message handlers.

`$con`->steal_message($msg)
: Permanently remove a borrowed message from the incoming message queue. No registered message handlers will now be run for this message.

`$msg` = `$con`->**pop_message()**;
: Permanently removes the first message on the incoming message queue, without running any registered message handlers. If you have hooked the connection up to an event loop (`Net::DBus::Binding::Reactor` for example), you probably don't want to be calling this method.

`$con`->set_watch_callbacks(\&add_watch, \&remove_watch, \&toggle_watch);
: Register a set of callbacks for adding, removing & updating watches in the application's event loop. Each parameter should be a code reference, which on each invocation, will be supplied with two parameters, the connection object and the watch object. If you are using a `Net::DBus::Binding::Reactor` object as the application event loop, then the 'manage' method on that object will call this on your behalf.

`$con`->set_timeout_callbacks(\&add_timeout, \&remove_timeout, \&toggle_timeout);
: Register a set of callbacks for adding, removing & updating timeouts in the application's event loop. Each parameter should be a code reference, which on each invocation, will be supplied with two parameters, the connection object and the timeout object. If you are using a `Net::DBus::Binding::Reactor` object as the application event loop, then the 'manage' method on that object will call this on your behalf.

`$con`->register_object_path($path, \&handler)
: Registers a handler for messages whose path matches that specified in the `$path` parameter. The supplied code reference will be invoked with two parameters, the connection object on which the message was received, and the message to be processed (an instance of the `Net::DBus::Binding::Message` class).

$con->unregister_object_path($path)
: Unregisters the handler associated with the object path `$path`. The handler would previously have been registered with the `register_object_path` or `register_fallback` methods.

$con->register_fallback($path, \&handler)
: Registers a handler for messages whose path starts with the prefix specified in the `$path` parameter. The supplied code reference will be invoked with two parameters, the connection object on which the message was received, and the message to be processed (an instance of the `Net::DBus::Binding::Message` class).

$con->set_max_message_size($bytes)
: Sets the maximum allowable size of a single incoming message. Messages over this size will be rejected prior to exceeding this threshold. The message size is specified in bytes.

$bytes = $con->**get_max_message_size()**;
: Retrieves the maximum allowable incoming message size. The returned size is measured in bytes.

$con->set_max_received_size($bytes)
: Sets the maximum size of the incoming message queue. Once this threshold is exceeded, no more messages will be read from wire before one or more of the existing messages are dispatched to their registered handlers. The implication is that the message queue can exceed this threshold by at most the size of a single message.

$bytes $con->**get_max_received_size()**
: Retrieves the maximum incoming message queue size. The returned size is measured in bytes.

$con->add_filter($coderef);
: Adds a filter to the connection which will be invoked whenever a message is received. The `$coderef` should be a reference to a subroutine, which returns a true value if the message should be filtered out, or a false value if the normal message dispatch should be performed.

my $msg = $con->make_raw_message($rawmsg)
: Creates a new message, initializing it from the low level C message object provided by the `$rawmsg` parameter. The returned object will be cast to the appropriate subclass of Net::DBus::Binding::Message.

my $msg = $con->make_error_message(replyto => `$method_call`, name => `$name`, description => `$description`);
: Creates a new message, representing an error which occurred during the handling of the method call object passed in as the `replyto` parameter. The `name` parameter is the formal name of the error condition, while the `description` is a short piece of text giving more specific information on the error.

my `$call` = $con->make_method_call_message(`$service_name`, `$object_path`, `$interface`, `$method_name`);
: Create a message representing a call on the object located at the path `$object_path` within the client owning the well-known name given by `$service_name`. The method to be invoked has the name `$method_name` within the interface specified by the `$interface` parameter.

my $msg = $con->make_method_return_message(replyto => `$method_call`);
: Create a message representing a reply to the method call passed in the `replyto` parameter.

my `$signal` = $con->make_signal_message(object_path => `$path`, interface => `$interface`, signal_name => `$name`);
: Creates a new message, representing a signal [to be] emitted by the object located under the path given by the `object_path` parameter. The name of the signal is given by the `signal_name` parameter, and is scoped to the interface given by the `interface` parameter.

AUTHOR

Daniel P. Berrange

COPYRIGHT

SEE ALSO

Net::DBus::Binding::Server, Net::DBus::Binding::Bus, Net::DBus::Binding::Message::Signal, Net::DBus::Binding::Message::MethodCall, Net::DBus::Binding::Message::MethodReturn, Net::DBus::Binding::Message::Error