

**NAME**

Net::DBus::Test::MockConnection – Fake a connection to the bus unit testing

**SYNOPSIS**

```
use Net::DBus;

my $bus = Net::DBus->test

# Register a service, and the object to be tested
use MyObject
my $service = $bus->export_service("org.example.MyService");
my $object = MyObject->new($service);

# Acquire the service & do tests
my $remote_service = $bus->get_service('org.example.MyService');
my $remote_object = $service->get_object("/org/example/MyObject");

# This traverses the mock connection, eventually
# invoking 'testSomething' on the $object above.
$remote_object->testSomething()
```

**DESCRIPTION**

This object provides a fake implementation of the Net::DBus::Binding::Connection enabling a pure 'in-memory' message bus to be mocked up. This is intended to facilitate creation of unit tests for services which would otherwise need to call out to other object on a live message bus. It is used as a companion to the Net::DBus::Test::MockObject module which is how fake objects are to be provided on the fake bus.

**METHODS**

```
my $con = Net::DBus::Test::MockConnection->new()
    Create a new mock connection object instance. It is not usually necessary to create instances of this
    object directly, instead the test method on the Net::DBus object can be used to get a handle to a test
    bus.

$con->send($message)
    Send a message over the mock connection. If the message is a method call, it will be dispatched
    straight to any corresponding mock object registered. If the message is an error or method return it will
    be made available as a return value for the send_with_reply_and_block method. If the
    message is a signal it will be queued up for processing by the dispatch method.

$con->request_name($service_name)
    Pretend to send a request to the bus registering the well known name specified in the
    $service_name parameter. In reality this is just a no-op giving the impression that the name was
    successfully registered.

my $reply = $con->send_with_reply_and_block($msg)
    Send a message over the mock connection and wait for a reply. The $msg should be an instance of
    Net::DBus::Binding::Message::MethodCall and the return $reply will be an instance
    of Net::DBus::Binding::Message::MethodReturn. It is also possible that an error will
    be thrown, with the thrown error being blessed into the Net::DBus::Error class.

$con->dispatch;
    Dispatches any pending messages in the incoming queue to their message handlers. This method
    should be called by test suites whenever they anticipate that there are pending signals to be dealt with.

$con->add_filter($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.
```

`$bus->add_match($rule)`

Register a signal match rule with the bus controller, allowing matching broadcast signals to be routed to this client. In reality this is just a no-op giving the impression that the match was successfully registered.

`$bus->remove_match($rule)`

Unregister a signal match rule with the bus controller, preventing further broadcast signals being routed to this client. In reality this is just a no-op giving the impression that the match was successfully unregistered.

`$con->register_object_path($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).

`$con->register_fallback($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).

`$con->unregister_object_path($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.

`my $msg = $con->make_error_message($replyto, $name, $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)`

Create a message representing a reply to the method call message passed in the `$replyto` parameter.

`my $msg = $con->make_signal_message($object_path, $interface, $signal_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.

## BUGS

It doesn't completely replicate the API of `Net::DBus::Binding::Connection`, merely enough to make the high level bindings work in a test scenario.

## AUTHOR

Daniel P. Berrange

## COPYRIGHT

Copyright (C) 2005–2009 Daniel P. Berrange

## SEE ALSO

`Net::DBus`, `Net::DBus::Test::MockObject`, `Net::DBus::Binding::Connection`,  
<<http://www.mockobjects.com/Faq.html>>