NAME

Net::DBus::Test::MockMessage – Fake a message object when unit testing

SYNOPSIS

Sending a message

```
my $msg = new Net::DBus::Test::MockMessage;
my $iterator = $msg->iterator;

$iterator->append_byte(132);
$iterator->append_int32(14241);

$connection->send($msg);
```

DESCRIPTION

This module provides a ''mock'' counterpart to the Net::DBus::Binding::Message class. It is basically a pure Perl fake message object providing the same contract as the real message object. It is intended for use internally by the testing APIs.

METHODS

my `$call` = Net::DBus::Test::MockMessage->new_method_call(service_name => `$service`, object_path => `$object`, interface => `$interface`, method_name => `$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 `$msg` = Net::DBus::Test::MockMessage->new_method_return(replyto => `$method_call`);

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

my `$signal` = Net::DBus::Test::MockMessage->new_signal(object_path => `$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.

my `$msg` = Net::DBus::Test::MockMessage->new_error(replyto => `$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 `$type` = `$msg`->get_type

Retrieves the type code for this message. The returned value corresponds to one of the four `Net::DBus::Test::MockMessage::MESSAGE_TYPE_*` constants.

my `$name` = `$msg`->get_error_name

Returns the formal name of the error, as previously passed in via the `name` parameter in the constructor.

my `$interface` = `$msg`->get_interface

Retrieves the name of the interface targeted by this message, possibly an empty string if there is no applicable interface for this message.

my `$path` = `$msg`->get_path

Retrieves the object path associated with the message, possibly an empty string if there is no applicable object for this message.

my `$name` = `$msg`->get_destination

Retrieves the unique or well-known bus name for client intended to be the recipient of the message. Possibly returns an empty string if the message is being broadcast to all clients.

my `$name` = `$msg->`get_sender
: Retireves the unique name of the client sending the message

my `$serial` = `$msg->`get_serial
: Retrieves the unique serial number of this message. The number is guaranteed unique for as long as the connection over which the message was sent remains open. May return zero, if the message is yet to be sent.

my `$name` = `$msg->`get_member
: For method calls, retrieves the name of the method to be invoked, while for signals, retrieves the name of the signal.

`$msg->`set_sender($name)
: Set the name of the client sending the message. The name must be the unique name of the client.

`$msg->`set_destination($name)
: Set the name of the intended recipient of the message. This is typically used for signals to switch them from broadcast to unicast.

my `$iterator` = `$msg->`iterator;
: Retrieves an iterator which can be used for reading or writing fields of the message. The returned object is an instance of the `Net::DBus::Binding::Iterator` class.

`$boolean` = `$msg->`**get_no_reply()**
: Gets the flag indicating whether the message is expecting a reply to be sent.

`$msg->`set_no_reply($boolean)
: Toggles the flag indicating whether the message is expecting a reply to be sent. All method call messages expect a reply by default. By toggling this flag the communication latency is reduced by removing the need for the client to wait

my `@values` = `$msg->`get_args_list
: De-marshall all the values in the body of the message, using the message signature to identify data types. The values are returned as a list.

`$msg->`append_args_list(@values)
: Append a set of values to the body of the message. Values will be encoded as either a string, list or dictionary as appropriate to their Perl data type. For more specific data typing needs, the Net::DBus::Binding::Iterator object should be used instead.

my `$sig` = `$msg->`get_signature
: Retrieves a string representing the type signature of the values packed into the body of the message.

AUTHOR

Daniel P. Berrange

COPYRIGHT

SEE ALSO

Net::DBus::Binding::Message, Net::DBus::Test::MockConnection, Net::DBus::Test::MockIterator