NAME

Net::DBus – Perl extension for the DBus message system

SYNOPSIS

```
    ####### Attaching to the bus ###########

    use Net::DBus;

    # Find the most appropriate bus
    my $bus = Net::DBus->find;

    # ... or explicitly go for the session bus
    my $bus = Net::DBus->session;

    # .... or explicitly go for the system bus
    my $bus = Net::DBus->system


    ######## Accessing remote services #########

    # Get a handle to the HAL service
    my $hal = $bus->get_service("org.freedesktop.Hal");

    # Get the device manager
    my $manager = $hal->get_object("/org/freedesktop/Hal/Manager",
                                   "org.freedesktop.Hal.Manager");

    # List devices
    foreach my $dev (@{$manager->GetAllDevices}) {
        print $dev, "\n";
    }


    ######### Providing services ##############

    # Register a service known as 'org.example.Jukebox'
    my $service = $bus->export_service("org.example.Jukebox");
```

DESCRIPTION

Net::DBus provides a Perl API for the DBus message system. The DBus Perl interface is currently operating against the 0.32 development version of DBus, but should work with later versions too, providing the API changes have not been too drastic.

Users of this package are either typically, service providers in which case the Net::DBus::Service and Net::DBus::Object modules are of most relevance, or are client consumers, in which case Net::DBus::RemoteService and Net::DBus::RemoteObject are of most relevance.

METHODS

my `$bus` = Net::DBus->find(%params);

Search for the most appropriate bus to connect to and return a connection to it. The heuristic used for the search is

- If DBUS_STARTER_BUS_TYPE is set to 'session' attach to the session bus

- Else If DBUS_STARTER_BUS_TYPE is set to 'system' attach to the system bus

- Else If DBUS_SESSION_BUS_ADDRESS is set attach to the session bus

- Else attach to the system bus

The optional `params` hash can contain be used to specify connection options. The only support option at this time is `nomainloop` which prevents the bus from being automatically attached to the main Net::DBus::Reactor event loop.

my `$bus` = Net::DBus->system(%params);
Return a handle for the system message bus. Note that the system message bus is locked down by default, so unless appropriate access control rules are added in /etc/dbus/system.d/, an application may access services, but won't be able to export services.

The optional `params` hash can be used to specify the following options:

nomainloop
If true, prevents the bus from being automatically attached to the main Net::DBus::Reactor event loop.

private
If true, the socket opened is private; any existing socket will be ignored and any future attempts to open the same bus will return a different existing socket or open a fresh one.

my `$bus` = Net::DBus->session(%params);
Return a handle for the session message bus.

The optional `params` hash can be used to specify the following options:

nomainloop
If true, prevents the bus from being automatically attached to the main Net::DBus::Reactor event loop.

private
If true, the socket opened is private; any existing socket will be ignored and any future attempts to open the same bus will return a different existing socket or open a fresh one.

my `$bus` = Net::DBus->test(%params);
Returns a handle for a virtual bus for use in unit tests. This bus does not make any network connections, but rather has an in-memory message pipeline. Consult Net::DBus::Test::MockConnection for further details of how to use this special bus.

my `$bus` = Net::DBus->new($address, `%params`);
Return a connection to a specific message bus. The `$address` parameter must contain the address of the message bus to connect to. An example address for a session bus might look like `unix:abstract=/tmp/dbus-PBFyyuUiVb,guid=191e0a43c3efc222e0818be556d67500`, while one for a system bus would look like `unix:/var/run/dbus/system_bus_socket`. The optional `params` hash can contain be used to specify connection options. The only support option at this time is `nomainloop` which prevents the bus from being automatically attached to the main Net::DBus::Reactor event loop.

my `$connection` = `$bus`->get_connection;
Return a handle to the underlying, low level connection object associated with this bus. The returned object will be an instance of the Net::DBus::Binding::Bus class. This method is not intended for use by (most!) application developers, so if you don't understand what this is for, then you don't need to be calling it!

my `$service` = `$bus`->get_service($name);
Retrieves a handle for the remote service identified by the service name `$name`. The returned object will be an instance of the Net::DBus::RemoteService class.

my $service = $bus->export_service($name);
Registers a service with the bus, returning a handle to the service. The returned object is an instance of the Net::DBus::Service class.

my $object = $bus->get_bus_object;
Retrieves a handle to the bus object, /org/freedesktop/DBus, provided by the service org.freedesktop.DBus. The returned object is an instance of Net::DBus::RemoteObject

my $name = $bus->get_unique_name;
Retrieves the unique name of this client's connection to the bus.

my $name = $bus->get_service_owner($service);
Retrieves the unique name of the client on the bus owning the service named by the $service parameter.

my $timeout = $bus->timeout(60 * 1000);
Sets or retrieves the timeout value which will be used for DBus requests belongs to this bus connection. The timeout should be specified in milliseconds, with the default value being 60 seconds.

DATA TYPING METHODS

These methods are not usually used, since most services provide introspection data to inform clients of their data typing requirements. If introspection data is incomplete, however, it may be necessary for a client to mark values with specific data types. In such a case, the following methods can be used. They are not, however, exported by default so must be requested at import time by specifying 'use Net::DBus qw(:typing)'

$typed_value = dbus_int16($value);
Mark a value as being a signed, 16−bit integer.

$typed_value = dbus_uint16($value);
Mark a value as being an unsigned, 16−bit integer.

$typed_value = dbus_int32($value);
Mark a value as being a signed, 32−bit integer.

$typed_value = dbus_uint32($value);
Mark a value as being an unsigned, 32−bit integer.

$typed_value = dbus_int64($value);
Mark a value as being an unsigned, 64−bit integer.

$typed_value = dbus_uint64($value);
Mark a value as being an unsigned, 64−bit integer.

$typed_value = dbus_double($value);
Mark a value as being a double precision IEEE floating point.

$typed_value = dbus_byte($value);
Mark a value as being an unsigned, byte.

$typed_value = dbus_string($value);
Mark a value as being a UTF−8 string. This is not usually required since 'string' is the default data type for any Perl scalar value.

$typed_value = dbus_signature($value);
Mark a value as being a UTF−8 string, whose contents is a valid type signature

$typed_value = dbus_object_path($value);
Mark a value as being a UTF−8 string, whose contents is a valid object path.

$typed_value = dbus_boolean($value);
Mark a value as being an boolean

`$typed_value` = dbus_array($value);
: Mark a value as being an array

`$typed_value` = dbus_struct($value);
: Mark a value as being a structure

`$typed_value` = dbus_dict($value);
: Mark a value as being a dictionary

`$typed_value` = dbus_variant($value);
: Mark a value as being a variant

`$typed_value` = dbus_unix_fd($value);
: Mark a value as being a unix file descriptor

SEE ALSO

Net::DBus, Net::DBus::RemoteService, Net::DBus::Service, Net::DBus::RemoteObject, Net::DBus::Object, Net::DBus::Exporter, Net::DBus::Dumper, Net::DBus::Reactor, `dbus-monitor(1)`, `dbus-daemon-1(1)`, `dbus-send(1)`, <http://dbus.freedesktop.org>,

AUTHOR

Daniel Berrange <dan@berrange.com>

COPYRIGHT