NAME

Net::DBus::ProxyObject – Implement objects to export to the bus

SYNOPSIS

```
# Connecting an object to the bus, under a service
package main;

use Net::DBus;

# Attach to the bus
my $bus = Net::DBus->find;

# Create our application's object instance
my $object = Demo::HelloWorld->new()

# Acquire a service 'org.demo.Hello'
my $service = $bus->export_service("org.demo.Hello");

# Finally export the object to the bus
my $proxy = Demo::HelloWorld::DBus->new($object);

....rest of program...


# Define a new package for the object we're going
# to export
package Demo::HelloWorld;

sub new {
    my $class = shift;
    my $service = shift;
    my $self = {};

    $self->{sighandler} = undef;

    bless $self, $class;

    return $self;
}

sub sighandler {
    my $self = shift;
    my $callback = shift;

    $self->[sighandler} = $callback;
}

sub Hello {
  my $self = shift;
  my $name = shift;

  &{$self->{sighandler}}("Greeting", "Hello $name");
  return "Said hello to $name";
}
```

```
sub Goodbye {
  my $self = shift;
  my $name = shift;

  &{$self->{sighandler}}("Greeting", "Goodbye $name");
  return "Said goodbye to $name";
}


# Define a new package for the object we're going
# to export
package Demo::HelloWorld::DBus;

# Specify the main interface provided by our object
use Net::DBus::Exporter qw(org.example.demo.Greeter);

# We're going to be a DBus object
use base qw(Net::DBus::ProxyObject);

# Export a 'Greeting' signal taking a stringl string parameter
dbus_signal("Greeting", ["string"]);

# Export 'Hello' as a method accepting a single string
# parameter, and returning a single string value
dbus_method("Hello", ["string"], ["string"]);

sub new {
    my $class = shift;
    my $service = shift;
    my $impl = shfit;
    my $self = $class->SUPER::new($service, "/org/demo/HelloWorld", $impl);

    bless $self, $class;

    $self->sighandler(sub {
        my $signame = shift;
        my $arg = shift;
        $self->emit_signal($signame, $arg);
    });

    return $self;
}

# Export 'Goodbye' as a method accepting a single string
# parameter, and returning a single string, but put it
# in the 'org.exaple.demo.Farewell' interface

dbus_method("Goodbye", ["string"], ["string"], "org.example.demo.Farewell");
```

DESCRIPTION

This the base for creating a proxy between a bus object and an application's object. It allows the application's object model to remain separate from the RPC object model. The proxy object will forward method calls from the bus, to the implementation object. The proxy object can also register callbacks against the application object, which it can use to then emit signals on the bus.

METHODS

my `$object` = Net::DBus::ProxyObject−>new($service, `$path`, `$impl`)

This creates a new DBus object with an path of `$path` registered within the service `$service`. The `$path` parameter should be a string complying with the usual DBus requirements for object paths, while the `$service` parameter should be an instance of Net::DBus::Service. The latter is typically obtained by calling the `export_service` method on the Net::DBus object. The `$impl` parameter is the application object which will implement the methods being exported to the bus.

my `$object` = Net::DBus::ProxyObject−>new($parentobj, `$subpath`, `$impl`)

This creates a new DBus child object with an path of `$subpath` relative to its parent `$parentobj`. The `$subpath` parameter should be a string complying with the usual DBus requirements for object paths, while the `$parentobj` parameter should be an instance of Net::DBus::BaseObject or a subclass. The `$impl` parameter is the application object which will implement the methods being exported to the bus.

AUTHOR

Daniel P. Berrange

COPYRIGHT

SEE ALSO

Net::DBus, Net::DBus::Service, Net::DBus::BaseObject, Net::DBus::ProxyObject, Net::DBus::Exporter, Net::DBus::RemoteObject