



**Full credit is given to the above companies including the Operating System (OS) that this PDF file was generated!**

***Rocky Enterprise Linux 9.2 Manual Pages on command 'Net::DBus::Object.3pm'***

***\$ man Net::DBus::Object.3pm***

Net::DBus::Object(3pm)      User Contributed Perl Documentation      Net::DBus::Object(3pm)

NAME

Net::DBus::Object - 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;
```

```
# Acquire a service 'org.demo.Hello'
```

```
my $service = $bus->export_service("org.demo.Hello");
```

```
# Export our object within the service
```

```
my $object = Demo::HelloWorld->new($service);
```

```
....rest of program...
```

```
# Define a new package for the object we're going
```

```

# to export
package Demo::HelloWorld;

# 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::Object);

# 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 $self = $class->SUPER::new($service, "/org/demo/HelloWorld");

    bless $self, $class;

    return $self;
}

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

    $self->emit_signal("Greeting", "Hello $name");
    return "Said hello to $name";
}

```

```

# 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");

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

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

```

## DESCRIPTION

This is the base for implementing objects which are directly exported to the bus. The methods implemented in a subclass are mapped to methods on the bus. By using this class, an application is directly tying the RPC functionality into its object model. Applications may thus prefer to use the "Net::DBus::ProxyObject" class which allows the RPC functionality to be maintained separately from the core object model, by proxying RPC method calls.

## METHODS

```
my $object = Net::DBus::Object->new($service, $path)
```

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.

```
my $object = Net::DBus::Object->new($parentobj, $subpath)
```

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.

#### AUTHOR

Daniel P. Berrange

#### COPYRIGHT

Copyright (C) 2005-2011 Daniel P. Berrange

#### SEE ALSO

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

perl v5.34.0

2022-02-06

Net::DBus::Object(3pm)