NAME

Net::DBus::BaseObject – base class for exporting objects to the bus

SYNOPSIS

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

# 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 _dispatch_object {
    my $self = shift;
    my $connection = shift;
    my $message = shift;

    if (....$message refers to a object's method ... ) {
       ...dispatch this object's interfaces/methods...
       return $reply;
    }
}
```

DESCRIPTION

This the base of all objects which are exported to the message bus. It provides the core support for type introspection required for objects exported to the message. When sub-classing this object, the `_dispatch` object should be implemented to handle processing of incoming messages. The Net::DBus::Exporter module is used to declare which methods (and signals) are being exported to the message bus.

All packages inheriting from this, will automatically have the interface `org.freedesktop.DBus.Introspectable` registered with Net::DBus::Exporter, and the `Introspect` method within this exported.

Application developers will rarely want to use this class directly, instead either Net::DBus::Object or `Net::DBus::ProxyObject` are the common choices. This class will only be used if wanting to write a new approach to dispatching incoming method calls.

METHODS

my `$object` = Net::DBus::BaseObject->new($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 $object = Net::DBus::BaseObject->new($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.

$object->**disconnect**();
: This method disconnects the object from the bus, such that it will no longer receive messages sent by other clients. Any child objects will be recursively disconnected too. After an object has been disconnected, it is possible for Perl to garbage collect the object instance. It will also make it possible to connect a newly created object to the same path.

my $bool = $object->is_connected
: Returns a true value if the object is connected to the bus, and thus capable of being accessed by remote clients. Returns false if the object is disconnected & thus ready for garbage collection. All objects start off in the connected state, and will only transition if the disconnect method is called.

my $service = $object->get_service
: Retrieves the Net::DBus::Service object within which this object is exported.

my $path = $object->get_object_path
: Retrieves the path under which this object is exported

$object->emit_signal_in($name, $interface, $client, @args);
: Emits a signal from the object, with a name of $name. If the $interface parameter is defined, the signal will be scoped within that interface. If the $client parameter is defined, the signal will be unicast to that client on the bus. The signal and the data types of the arguments @args must have been registered with Net::DBus::Exporter by calling the dbus_signal method.

$self->emit_signal_to($name, $client, @args);
: Emits a signal from the object, with a name of $name. The signal and the data types of the arguments @args must have been registered with Net::DBus::Exporter by calling the dbus_signal method. The signal will be sent only to the client named by the $client parameter.

$self->emit_signal($name, @args);
: Emits a signal from the object, with a name of $name. The signal and the data types of the arguments @args must have been registered with Net::DBus::Exporter by calling the dbus_signal method. The signal will be broadcast to all clients on the bus.

$object->connect_to_signal_in($name, $interface, $coderef);
: Connects a callback to a signal emitted by the object. The $name parameter is the name of the signal within the object, and $coderef is a reference to an anonymous subroutine. When the signal $name is emitted by the remote object, the subroutine $coderef will be invoked, and passed the parameters from the signal. The $interface parameter is used to specify the explicit interface defining the signal to connect to.

$object->connect_to_signal($name, $coderef);
: Connects a callback to a signal emitted by the object. The $name parameter is the name of the signal within the object, and $coderef is a reference to an anonymous subroutine. When the signal $name is emitted by the remote object, the subroutine $coderef will be invoked, and passed the parameters from the signal.

$reply = $object->_dispatch_object($connection, $message);
: The _dispatch_object method is to be used to handle dispatch of methods implemented by the object. The default implementation is a no-op and should be overridden by subclasses todo whatever processing is required. If the $message could be handled then another Net::DBus::Binding::Message instance should be returned for the reply. If undef is returned, then a generic error will be returned to the caller.

$currvalue = $object->_dispatch_property($name); =item $object->_dispatch_property($name, $newvalue);
: The _dispatch_property method is to be used to handle dispatch of property reads and writes.

The `$name` parameter is the name of the property being accessed. If `$newvalue` is supplied then the property is to be updated, otherwise the current value is to be returned. The default implementation will simply raise an error, so must be overridden in subclasses.

AUTHOR

Daniel P. Berrange

COPYRIGHT

SEE ALSO

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