



**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::Binding::Introspector.3pm'***

***\$ man Net::DBus::Binding::Introspector.3pm***

Net::DBus::Binding::IntrospectoUsermContributed Perl DocumenNet::DBus::Binding::Introspector(3pm)

### NAME

Net::DBus::Binding::Introspector - Handler for object introspection data

### SYNOPSIS

```
# Create an object populating with info from an
# XML doc containing introspection data.
my $ins = Net::DBus::Binding::Introspector->new(xml => $data);
# Create an object, defining introspection data
# programmatically
my $ins = Net::DBus::Binding::Introspector->new(object_path => $object->get_object_path);
$ins->add_method("DoSomething", ["string"], [], "org.example.MyObject");
$ins->add_method("TestSomething", ["int32"], [], "org.example.MyObject");
```

### DESCRIPTION

This class is responsible for managing introspection data, and answering questions about it. This is not intended for use by application developers, whom should instead consult the higher level API in Net::DBus::Exporter.

### METHODS

```
my $ins = Net::DBus::Binding::Introspector->new(object_path => $object_path, xml => $xml);
```

Creates a new introspection data manager for the object registered at the path specified for the "object\_path" parameter. The optional "xml" parameter can be used to pre-load the manager with introspection metadata from an XML document.

```
$ins->add_interface($name)
```

Register the object as providing an interface with the name \$name

```
my $bool = $ins->has_interface($name)
```

Return a true value if the object is registered as providing an interface with the name \$name; returns false otherwise.

```
my @interfaces = $ins->has_method($name, [$interface])
```

Return a list of all interfaces provided by the object, which contain a method called \$name. This may be an empty list. The optional \$interface parameter can restrict the check to just that one interface.

```
my $boolean = $ins->is_method_allowed($name[, $interface])
```

Checks according to whether the remote caller is allowed to invoke the method \$name on the object associated with this introspector. If this object has 'strict exports' enabled, then only explicitly exported methods will be allowed. The optional \$interface parameter can restrict the check to just that one interface. Returns a non-zero value if the method should be allowed.

```
my @interfaces = $ins->has_signal($name)
```

Return a list of all interfaces provided by the object, which contain a signal called \$name. This may be an empty list.

```
my @interfaces = $ins->has_property($name)
```

Return a list of all interfaces provided by the object, which contain a property called \$name. This may be an empty list. The optional \$interface parameter can restrict the check to just that one interface.

```
$ins->add_method($name, $params, $returns, $interface, $attributes, $paramnames,  
$returnnames);
```

Register the object as providing a method called \$name accepting parameters whose types are declared by \$params and returning values whose type are declared by \$returns. The method will be scoped to the interface named by \$interface. The \$attributes parameter is a hash reference for annotating the method. The \$paramnames and \$returnnames parameters are a list of argument and return value names.

```
$ins->add_signal($name, $params, $interface, $attributes);
```

Register the object as providing a signal called \$name with parameters whose types are declared by \$params. The signal will be scoped to the interface named by \$interface.

The \$attributes parameter is a hash reference for annotating the signal.

```
$ins->add_property($name, $type, $access, $interface, $attributes);
```

Register the object as providing a property called \$name with a type of \$type. The

\$access parameter can be one of "read", "write", or "readwrite". The property will be scoped to the interface named by \$interface. The \$attributes parameter is a hash reference for annotating the signal.

```
my $boolean = $ins->is_method_deprecated($name, $interface)
```

Returns a true value if the method called \$name in the interface \$interface is marked as deprecated

```
my $boolean = $ins->is_signal_deprecated($name, $interface)
```

Returns a true value if the signal called \$name in the interface \$interface is marked as deprecated

```
my $boolean = $ins->is_property_deprecated($name, $interface)
```

Returns a true value if the property called \$name in the interface \$interface is marked as deprecated

```
my $boolean = $ins->does_method_reply($name, $interface)
```

Returns a true value if the method called \$name in the interface \$interface will generate a reply. Returns a false value otherwise.

```
my $boolean = $ins->method_has_strict_exceptions($name, $interface)
```

Returns true if the method called \$name in the interface \$interface has the strict\_exceptions attribute; that is any exceptions which aren't Net::DBus::Error objects should not be caught and allowed to travel up the stack.

```
my @names = $ins->list_interfaces
```

Returns a list of all interfaces registered as being provided by the object.

```
my @names = $ins->list_methods($interface)
```

Returns a list of all methods registered as being provided by the object, within the interface \$interface.

```
my @names = $ins->list_signals($interface)
```

Returns a list of all signals registered as being provided by the object, within the interface \$interface.

```
my @names = $ins->list_properties($interface)
```

Returns a list of all properties registered as being provided by the object, within the interface \$interface.

```
my @paths = $self->list_children;
```

Returns a list of object paths representing all the children of this node.

```
my $path = $ins->get_object_path
```

Returns the path of the object associated with this introspection data

```
my @types = $ins->get_method_params($interface, $name)
```

Returns a list of declared data types for parameters of the method called \$name within the interface \$interface.

```
my @types = $ins->get_method_param_names($interface, $name)
```

Returns a list of declared names for parameters of the method called \$name within the interface \$interface.

```
my @types = $ins->get_method_returns($interface, $name)
```

Returns a list of declared data types for return values of the method called \$name within the interface \$interface.

```
my @types = $ins->get_method_return_names($interface, $name)
```

Returns a list of declared names for return values of the method called \$name within the interface \$interface.

```
my @types = $ins->get_signal_params($interface, $name)
```

Returns a list of declared data types for values associated with the signal called \$name within the interface \$interface.

```
my @types = $ins->get_signal_param_names($interface, $name)
```

Returns a list of declared names for values associated with the signal called \$name within the interface \$interface.

```
my $type = $ins->get_property_type($interface, $name)
```

Returns the declared data type for property called \$name within the interface \$interface.

```
my $bool = $ins->is_property_readable($interface, $name);
```

Returns a true value if the property called \$name within the interface \$interface can have its value read.

```
my $bool = $ins->is_property_writable($interface, $name);
```

Returns a true value if the property called \$name within the interface \$interface can have its value written to.

```
my $xml = $ins->format([$obj])
```

Return a string containing an XML document representing the state of the introspection data. The optional \$obj parameter can be an instance of Net::DBus::Object to include object specific information in the XML (eg child nodes).

```
my $xml_fragment = $ins->to_xml
```

Returns a string containing an XML fragment representing the state of the introspection data. This is basically the same as the "format" method, but without the leading doctype declaration.

```
$type = $ins->to_xml_type($type)
```

Takes a text-based representation of a data type and returns the compact representation used in XML introspection data.

```
$ins->encode($message, $type, $name, $direction, @args)
```

Append a set of values <@args> to a message object \$message. The \$type parameter is either "signal" or "method" and \$direction is either "params" or "returns". The introspection data will be queried to obtain the declared data types & the argument marshalling accordingly.

```
my @args = $ins->decode($message, $type, $name, $direction)
```

Unmarshalls the contents of a message object \$message. The \$type parameter is either "signal" or "method" and \$direction is either "params" or "returns". The introspection data will be queried to obtain the declared data types & the arguments unmarshalled accordingly.

#### AUTHOR

Daniel P. Berrange

#### COPYRIGHT

Copyright (C) 2004-2011 Daniel P. Berrange

#### SEE ALSO

Net::DBus::Exporter, Net::DBus::Binding::Message

perl v5.34.0

2022-02-06

Net::DBus::Binding::Introspector(3pm)