cosProperty Application

version 1.0

Typeset in LaTeX from SGML source using the DOCBUILDER 3.2.2 Document System.

Contents

Chapter 1

cosProperty User's Guide

The cosProperty Application is an Erlang implementation of the OMG CORBA Property Service.

1.1 The cosProperty Application

1.1.1 Content Overview

The cosProperty documentation is divided into three sections:

- PART ONE - The User's Guide
  Description of the cosProperty Application including services and a small tutorial demonstrating the development of a simple service.
- PART TWO - Release Notes
  A concise history of cosProperty.
- PART THREE - The Reference Manual
  A quick reference guide, including a brief description, to all the functions available in cosProperty.

1.1.2 Brief description of the User's Guide

The User's Guide contains the following parts:

- cosProperty overview
- cosProperty installation
- A tutorial example

1.2 Introduction to cosProperty

1.2.1 Overview

The cosProperty application is compliant with the OMG[1] Service CosProperty Service.

[1] URL: `http://www.omg.org`

Purpose and Dependencies

cosProperty is dependent on *Orber*, which provides CORBA functionality in an Erlang environment.

Prerequisites

To fully understand the concepts presented in the documentation, it is recommended that the user is familiar with distributed programming, CORBA and the Orber application.

Recommended reading includes *CORBA, Fundamentals and Programming - Jon Siegel* and *Open Telecom Platform Documentation Set*. It is also helpful to have read *Concurrent Programming in Erlang*.

1.3 Installing cosProperty

1.3.1 Installation Process

This chapter describes how to install cosProperty [page 22] in an Erlang Environment.

Preparation

Before starting the installation process for cosProperty, the application Orber must be running.

Configuration

First the cosProperty application must be installed by using `cosProperty:install()` and, if requested, `cosProperty:install_db()`, followed by `cosProperty:start()`. Now we can start the desired Factory type by using either `cosProperty:start_SetFactory()` or `cosProperty:start_SetDefFactory()`.

1.4 cosProperty Examples

1.4.1 A tutorial on how to create a simple service

Initiate the application

To use the cosProperty application Orber must be running.

How to run everything

Below is a short transcript on how to run cosProperty.

```
%% Start Mnesia and Orber
mnesia:delete_schema([node()]),
mnesia:create_schema([node()]),
orber:install([node()]),
mnesia:start(),
orber:start(),

%% Install Property Service in the IFR.
cosProperty:install(),

%% Install Property Service in mnesia.
cosProperty:install_db(),

%% Now start the application.
cosProperty:start(),

%% To be able to create Property objects we must first a Factory
%% of our prefered type.
Fac = cosProperty:start_SetDefFactory(),

%% Now we can create a Property object.
'CosPropertyService_PropertySetDefFactory':
      create_propertysetdef(Fac),

%% Now we can create any allowed properties. There are many
%% options which are all described further in the documentation.
```

1.5 cosProperty Release Notes

1.5.1 cosProperty 1.0.1, Release Notes

Improvements and new features

- First release of the cosProperty application.
  Own Id: -

Fixed bugs and malfunctions

-

Incompatibilities

-

Known bugs and problems

-

cosProperty Reference Manual

Short Summaries

- Erlang Module CosPropertyService_PropertiesIterator [page 9] – This module implements the OMG CosPropertyService::PropertiesIterator interface.
- Erlang Module CosPropertyService_PropertyNamesIterator [page 11] – This module implements the OMG CosPropertyService::PropertyNamesIterator interface.
- Erlang Module CosPropertyService_PropertySet [page 12] – This module implements the OMG CosPropertyService::PropertySet interface.
- Erlang Module CosPropertyService_PropertySetDef [page 15] – This module implements the OMG CosPropertyService::PropertySetDef interface.
- Erlang Module CosPropertyService_PropertySetDefFactory [page 18] – This module implements the OMG CosPropertyService::PropertySetDefFactory interface.
- Erlang Module CosPropertyService_PropertySetFactory [page 20] – This module implements the OMG CosPropertyService::PropertySetFactory interface.
- Erlang Module cosProperty [page 22] – The main module of the cosProperty application

CosPropertyService_PropertiesIterator

The following functions are exported:

- `reset(Iterator) -> ok`
  [page 9] Reset the position to the first property
- `next_one(Iterator) -> Reply`
  [page 9] Return trueif a Property exists at the current position and the out parameter is a valid Property. Otherwise false and a non-valid property
- `next_n(Iterator, HowMany) -> Reply`
  [page 9] Return true if the requested number of properties can be delivered and there are additional properties. Otherwise false is returned and a sequence of max `HowMany` properties
- `destroy(Iterator) -> ok`
  [page 10] Terminate the target object

CosPropertyService_PropertyNamesIterator

The following functions are exported:

- `reset(Iterator) -> ok`
  [page 11] Reset the position to the first property name
- `next_one(Iterator) -> Reply`
  [page 11] Return true if a Property Name exists at the current position and the out parameter is a valid Property Name. Otherwise false and a non-valid Property Name
- `next_n(Iterator, HowMany) -> Reply`
  [page 11] Return `HowMany` Property Names and a boolean which is true if additional Property Names exists
- `destroy(Iterator) -> ok`
  [page 11] Terminate the target object

CosPropertyService_PropertySet

The following functions are exported:

- `define_property(PropertySet, Name, Value) -> Reply`
  [page 12] Add a new property to the target object
- `define_properties(PropertySet, Properties) -> Reply`
  [page 12] Add new properties to the target object
- `get_number_of_properties(PropertySet) -> ulong()`
  [page 13] Get the number of properties associated with the target object
- `get_all_property_names(PropertySet, Max) -> Reply`
  [page 13] Get `Max` property names. If the target object have additional associated properties they will be put in the returned Iterator
- `get_property_value(PropertySet, Name) -> Reply`
  [page 13] Return the property value associated with given name
- `get_properties(PropertySet, Names) -> Reply`
  [page 13] Return all properties associated with given names
- `get_all_properties(PropertySet, Max) -> Reply`
  [page 13] Return a list `Max` properties or less. If more properties are associated with the target object they will be put in the `PropertiesIterator`.
- `delete_property(PropertySet, Name) -> Reply`
  [page 14] Delete the property with given Name
- `delete_properties(PropertySet, Names) -> Reply`
  [page 14] Delete all properties with given Names
- `delete_all_properties(PropertySet) -> boolean()`
  [page 14] Delete all properties
- `is_property_defined(PropertySet, Name) -> Reply`
  [page 14] Return true if the target have an associated property with given name

CosPropertyService_PropertySetDef

The following functions are exported:

- `get_allowed_property_types(PropertySetDef) -> Reply`
  [page 15] Return allowed TypeCodes for the target object
- `get_allowed_properties(PropertySetDef) -> Reply`
  [page 15] Return a sequence of the allowed properties
- `define_property_with_mode(PropertySetDef, Name, Value, Mode) -> Reply`
  [page 15] Associate a new property with the target object
- `define_properties_with_modes(PropertySetDef, PropertyDefs) -> Reply`
  [page 16] Associate the given Property Definitions with the target object
- `get_property_mode(PropertySetDef, Name) -> Reply`
  [page 16] Return the mode of the given property
- `get_property_modes(PropertySetDef, Names) -> Reply`
  [page 16] Return the modes of the given properties
- `set_property_mode(PropertySetDef, Name, Mode) -> Reply`
  [page 17] Change the given property's mode
- `set_property_modes(PropertySetDef, PropertyModes) -> Reply`
  [page 17] Change the listed properties mode's

CosPropertyService_PropertySetDefFactory

The following functions are exported:

- `create_propertysetdef(Factory) ->`
  [page 18] Create a new `PropertySetDef` with no predefined settings
- `create_constrained_propertysetdef(Factory, PropertyTypes, PropertyDefs) -> Reply`
  [page 18] Create a new `PropertySetDef` with specified constraints
- `create_initial_propertysetdef(Factory, PropertyDefs) -> Reply`
  [page 18] Create a new `PropertySetDef` with specified initial properties

CosPropertyService_PropertySetFactory

The following functions are exported:

- `create_propertyset(Factory) -> PropertySet`
  [page 20] Create a new `PropertySet` with no predefined properties
- `create_constrained_propertyset(Factory, PropertyTypes, Properties) -> Reply`
  [page 20] Create a new `PropertySet` with specified constraints
- `create_initial_propertyset(Factory, Properties) -> Reply`
  [page 20] Create a new `PropertySet` with specified initial properties

cosProperty

The following functions are exported:

- `install() -> Return`
  [page 22] Install the cosProperty application in the IFR
- `install_db() -> Return`
  [page 22] Install data in mnesia necessary for running the cosProperty applicatio
- `uninstall() -> Return`
  [page 22] Remove all data in the IFR related to the cosProperty application
- `uninstall_db() -> Return`
  [page 22] Remove all data from mnesia related to the cosProperty application
- `start() -> Return`
  [page 22] Start the cosProperty application
- `start_SetDefFactory() -> Return`
  [page 22] Start a PropertySetDef Factory
- `start_SetFactory() -> Return`
  [page 23] Start a PropertySet Factory
- `stop_SetDefFactory(Factory) -> Return`
  [page 23] Stop the given PropertySetDef Factory
- `stop_SetFactory(Factory) -> Return`
  [page 23] Stop the given PropertySet Factory
- `stop() -> Return`
  [page 23] Stop the cosProperty application

CosPropertyService_-PropertiesIterator

Erlang Module

To get access to the record definitions for the structures use:
`-include_lib("cosProperty/include/CosPropertyService.hrl").`

Exports

`reset(Iterator) -> ok`

Types:

- Iterator = #objref

This operation resets the position to the first property.

`next_one(Iterator) -> Reply`

Types:

- Iterator = #objref
- Reply = {boolean(), #'CosPropertyService_Property'{property_name = Name, property_value = Value}}
- Name = string()
- Value = #any

This operation returns true . If false is returned the out parameter is a non-valid Property.

`next_n(Iterator, HowMany) -> Reply`

Types:

- Iterator = #objref
- HowMany = long()
- Reply = {boolean(), Properties}
- Properties = [#'CosPropertyService_Property'{property_name = Name, property_value = Value}]
- Name = string()
- Value = #any

This operation returns true if the requested number of properties can be delivered and there are additional properties. If false is returned and a sequence of max `HowMany` properties will be returned and no more properties can be delivered.

```
destroy(Iterator) -> ok
```

Types:

- Iterator = #objref

This operation will terminate the Iterator and all subsequent calls will fail.

CosPropertyService_-PropertyNamesIterator

Erlang Module

To get access to the record definitions for the structures use:
`-include_lib("cosProperty/include/CosPropertyService.hrl").`

Exports

`reset(Iterator) -> ok`

Types:

- Iterator = #objref

This operation resets the position to the first property name.

`next_one(Iterator) -> Reply`

Types:

- Iterator = #objref
- Reply = {boolean(), Name}
- Name = string()

This operation returns true if a Property Name exists at the current position and the out parameter is a valid Property Name. If false is returned the out parameter is a non-valid Property Name.

`next_n(Iterator, HowMany) -> Reply`

Types:

- Iterator = #objref
- HowMany = long()
- Reply = {boolean(), [Name]}
- Name = string()

This operation returns true if the requested number of Property Names can be delivered and there are additional property names. If false is returned a sequence of max `HowMany` property names will be returned and no more Property Names can be delivered.

`destroy(Iterator) -> ok`

Types:

- Iterator = #objref

This operation will terminate the Iterator and all subsequent calls will fail.

CosPropertyService_PropertySet

Erlang Module

To get access to the record definitions for the structures use:
`-include_lib("cosProperty/include/CosPropertyService.hrl").`

Exports

`define_property(PropertySet, Name, Value) -> Reply`

Types:

- PropertySet = #objref
- Name = non-empty string()
- Value = #any
- Reply = ok | {'EXCEPTION', #CosPropertyService_InvalidPropertyName{}} | {'EXCEPTION', #CosPropertyService_ConflictingProperty{}} | {'EXCEPTION', #CosPropertyService_UnsupportedTypeCode{}} | {'EXCEPTION', #CosPropertyService_UnsupportedProperty{}} | {'EXCEPTION', #CosPropertyService_ReadOnlyProperty{}}

This operation adds a new property to the given object. Depending on which initial arguments was supplied when starting the object several exceptions may be raised.

`define_properties(PropertySet, Properties) -> Reply`

Types:

- PropertySet = #objref
- Properties = [#'CosPropertyService_Property'{property_name = Name, property_value = Value}]
- Name = string()
- Value = #any
- Reply = ok | {'EXCEPTION', #CosPropertyService_MultipleExceptions{exceptions = Excs}}
- Excs = [#'CosPropertyService_PropertyException{reason = Reason, failing_property_name = Name}]
- Reason = invalid_property_name | conflicting_property | property_not_found | unsupported_type_code | unsupported_property | unsupported_mode | fixed_property | read_only_property

This operation adds several new properties to the given object. Depending on which initial arguments was supplied when starting the object an exceptions may be raised listing the properties failing.

`get_number_of_properties(PropertySet) -> ulong()`

Types:

- PropertySet = #objref

This operation returns the number of properties associated with the target object.

`get_all_property_names(PropertySet, Max) -> Reply`

Types:

- PropertySet = NamesIterator = #objref
- Max = ulong()
- Reply = {ok, Names, NamesIterator}
- Names = [string()]

This operation returns up to `Max` property names. If the target object have additional associated properties they will be put in the returned Iterator, otherwise the Iterator will be a `NIL` object.

`get_property_value(PropertySet, Name) -> Reply`

Types:

- PropertySet = #objref
- Name = string()
- Reply = #any | {'EXCEPTION', #CosPropertyService_PropertyNotFound{}} | {'EXCEPTION', #CosPropertyService_InvalidPropertyName{}}

This operation returns the property value associated with given name. If no such property exists or the given name is an empty string an exception will be raised.

`get_properties(PropertySet, Names) -> Reply`

Types:

- PropertySet = #objref
- Names = [string()]
- Reply = {boolean(), Properties}
- Properties = [#'CosPropertyService_Property'{property_name = Name, property_value = Value}]

This operation returns all properties associated with given names. If the boolean flag is true all properties where retrieved correctly, otherwise, all properties with the type `tk_void` was not found.

`get_all_properties(PropertySet, Max) -> Reply`

Types:

- PropertySet = PropertiesIterator = #objref
- Reply = {ok, Properties, PropertiesIterator}
- Properties = [#'CosPropertyService_Property'{property_name = Name, property_value = Value}]

This operation return a list `Max` properties or less. If more properties are associated with the target object they will be put in the `PropertiesIterator`. If the object had less than `Max` associated properties the Iterator will be a `NIL` object.

`delete_property(PropertySet, Name) -> Reply`

Types:

- PropertySet = #objref
- Name = string()
- Reply = ok | {'EXCEPTION', #CosPropertyService_FixedProperty{}} | {'EXCEPTION', #CosPropertyService_PropertyNotFound{}} | {'EXCEPTION', #CosPropertyService_InvalidPropertyName{}}

This operation tries to delete the property with given Name. An exception which indicates why it failed is raised if so needed.

`delete_properties(PropertySet, Names) -> Reply`

Types:

- PropertySet = #objref
- Names = [string()]
- Reply = ok | {'EXCEPTION', #CosPropertyService_MultipleExceptions{exceptions = Excs}}
- Excs = [#'CosPropertyService_PropertyException{reason = Reason, failing_property_name = Name}]
- Reason = invalid_property_name | conflicting_property | property_not_found | unsupported_type_code | unsupported_property | unsupported_mode | fixed_property | read_only_property

This operation tries to delete all given Properties. If one or more removal fails an exception is raised which describe why.

`delete_all_properties(PropertySet) -> boolean()`

Types:

- PropertySet = #objref

This operation deletes all properties. The boolean flag, if set to false, indicates that it was not possible to remove one or more properties, e.g., may be read only.

`is_property_defined(PropertySet, Name) -> Reply`

Types:

- PropertySet = #objref
- Name = non-empty string()
- Reply = boolean() | {'EXCEPTION', #CosPropertyService_InvalidPropertyName{}}

This operation returns true if the target have an associated property with given name.

CosPropertyService_-PropertySetDef

Erlang Module

To get access to the record definitions for the structures use:
`-include_lib("cosProperty/include/CosPropertyService.hrl").`

This module also exports the functions described in

CosPropertyService_PropertySet [page 12]

Exports

`get_allowed_property_types(PropertySetDef) -> Reply`

Types:

- PropertySetDef = #objref
- Reply = {ok, PropertyTypes}
- PropertyTypes = [CORBA::TypeCode]

This operation return the TypeCodes which we are allowed to use when adding new properties.

`get_allowed_properties(PropertySetDef) -> Reply`

Types:

- PropertySetDef = #objref
- Reply = {ok, PropertyDefs}
- PropertyDefs = [#'CosPropertyService_PropertyDef'{property_name = Name, property_value = Value, property_mode = Mode}]
- Name = string()
- Value = #any
- Mode = normal | read_only | fixed_normal | fixed_readonly | undefined

This operation a sequence of the allowed properties we may alter; depends on which mode associated with a certain property.

`define_property_with_mode(PropertySetDef, Name, Value, Mode) -> Reply`

Types:

- PropertySetDef = #objref
- Name = non-empty string()
- Value = #any

- Mode = normal | read_only | fixed_normal | fixed_readonly | undefined
- Reply = ok | {'EXCEPTION', #CosPropertyService_InvalidPropertyName{}} | {'EXCEPTION', #CosPropertyService_ConflictingProperty{}} | {'EXCEPTION', #CosPropertyService_UnsupportedTypeCode{}} | {'EXCEPTION', #CosPropertyService_UnsupportedProperty{}} | {'EXCEPTION', #CosPropertyService_UnsupportedMode{}} | {'EXCEPTION', #CosPropertyService_ReadOnlyProperty{}}

This operation attempts to associate a new property with the target object. If we fail to do so the appropriate exception is raised.

`define_properties_with_modes(PropertySetDef, PropertyDefs) -> Reply`

Types:

- PropertySetDef = #objref
- PropertyDefs = [#'CosPropertyService_PropertyDef'{property_name = Name, property_value = Value, property_mode = Mode}]
- Name = string()
- Value = #any
- Mode = normal | read_only | fixed_normal | fixed_readonly | undefined
- Reply = ok | {'EXCEPTION', #CosPropertyService_MultipleExceptions{exceptions = Excs}}
- Excs = [#'CosPropertyService_PropertyException{reason = Reason, failing_property_name = Name}]
- Reason = invalid_property_name | conflicting_property | property_not_found | unsupported_type_code | unsupported_property | unsupported_mode | fixed_property | read_only_property

This operation attempts to associate the given Property Definitions with the target object. If one or more attempts fail an exception is raised describing which properties we where not able to create.

`get_property_mode(PropertySetDef, Name) -> Reply`

Types:

- PropertySetDef = #objref
- Name = string()
- Reply = Mode | {'EXCEPTION', #CosPropertyService_InvalidPropertyName{}} | {'EXCEPTION', #CosPropertyService_PropertyNotFound{}}
- Mode = normal | read_only | fixed_normal | fixed_readonly | undefined

This operation returns the type of the given property.

`get_property_modes(PropertySetDef, Names) -> Reply`

Types:

- PropertySetDef = #objref
- Names = [string()]
- Reply = {boolean(), PropertyModes}
- PropertyModes = [#'CosPropertyService_PropertyMode'{property_name = Name, property_mode = Mode}]
- Name = string()

- Mode = normal | read_only | fixed_normal | fixed_readonly | undefined

This operation returns the modes of the listed properties. If the boolean flag is false, all properties with mode `undefined` this operation failed to comply.

`set_property_mode(PropertySetDef, Name, Mode) -> Reply`

Types:

- PropertySetDef = #objref
- Name = string()
- Mode = normal | read_only | fixed_normal | fixed_readonly | undefined
- Reply = ok | {'EXCEPTION', #CosPropertyService_InvalidPropertyName{}} | {'EXCEPTION', #CosPropertyService_UnsupportedMode{}} | {'EXCEPTION', #CosPropertyService_PropertyNotFound{}}

This operation changes the given property's mode. Return the approprioate exception if not able to fulfill the request.

`set_property_modes(PropertySetDef, PropertyModes) -> Reply`

Types:

- PropertySetDef = #objref
- PropertyModes = [#'CosPropertyService_PropertyMode'{property_name = Name, property_mode = Mode}]
- Name = string()
- Mode = normal | read_only | fixed_normal | fixed_readonly | undefined
- Reply = ok | {'EXCEPTION', #CosPropertyService_MultipleExceptions{exceptions = Excs}}
- Excs = [#'CosPropertyService_PropertyException{reason = Reason, failing_property_name = Name}]
- Reason = invalid_property_name | conflicting_property | property_not_found | unsupported_type_code | unsupported_property | unsupported_mode | fixed_property | read_only_property

This operation attempts to update the listed properties mode's. Rasises an exception which describe which and why an operation failed.

CosPropertyService_-PropertySetDefFactory

Erlang Module

To get access to the record definitions for the structures use:
`-include_lib("cosProperty/include/CosPropertyService.hrl").`

Exports

`create_propertysetdef(Factory) ->`

Types:

- Factory = PropertySetDef = #objref

This operation creates a new `PropertySetDef` with no predefined settings.

`create_constrained_propertysetdef(Factory, PropertyTypes, PropertyDefs) -> Reply`

Types:

- Factory = PropertySetDef = #objref
- PropertyTypes = [CORBA::TypeCode]
- PropertyDefs = [#'CosPropertyService_PropertyDef'{property_name = Name, property_value = Value, property_mode = Mode}]
- Name = string()
- Value = #any
- Mode = normal | read_only | fixed_normal | fixed_readonly | undefined
- Reply = {'EXCEPTION', #CosPropertyService_ConstraintNotSupported{}} | PropertySetDef
- PropertySetDef = #objref

This operation creates a new `PropertySetDef` with specific constraints. PropertyTypes states allowed TypeCode's and PropertyDefs valid `CosPropertyService::PropertyDef` data.

`create_initial_propertysetdef(Factory, PropertyDefs) -> Reply`

Types:

- Factory = PropertySetDef = #objref
- PropertyDefs = [#'CosPropertyService_PropertyDef'{property_name = Name, property_value = Value, property_mode = Mode}]
- Name = string()
- Value = #any

- Mode = normal | read_only | fixed_normal | fixed_readonly | undefined
- Reply = {'EXCEPTION', #CosPropertyService_MultipleExceptions{exceptions = Excs}} | PropertySetDef
- Excs = [#'CosPropertyService_PropertyException{reason = Reason, failing_property_name = Name}]
- Reason = invalid_property_name | conflicting_property | property_not_found | unsupported_type_code | unsupported_property | unsupported_mode | fixed_property | read_only_property
- PropertySetDef = #objref

This operation creates a new `PropertySetDef` with specific initial properties.

CosPropertyService_-PropertySetFactory

Erlang Module

To get access to the record definitions for the structures use:
`-include_lib("cosProperty/include/CosPropertyService.hrl").`

Exports

`create_propertyset(Factory) -> PropertySet`

Types:

- Factory = PropertySet = #objref

This operation creates a new `PropertySet` with no predefined properties.

`create_constrained_propertyset(Factory, PropertyTypes, Properties) -> Reply`

Types:

- Factory = #objref
- PropertyTypes = [CORBA::TypeCode]
- Properties = [#'CosPropertyService_Property'{property_name = Name, property_value = Value}]
- Name = string()
- Value = #any
- Reply = {'EXCEPTION', #CosPropertyService_ConstraintNotSupported{}} | PropertySet
- PropertySet = #objref

This operation creates a new `PropertySet` with specific constraints. PropertyTypes states allowed TypeCode's and Properties valid `CosPropertyService::Property` data.

`create_initial_propertyset(Factory, Properties) -> Reply`

Types:

- Factory = #objref
- Properties = [#'CosPropertyService_Property'{property_name = Name, property_value = Value}]
- Name = string()
- Value = #any
- Reply = {'EXCEPTION', #CosPropertyService_MultipleExceptions{exceptions = Excs}} | PropertySet

- Excs = [#'CosPropertyService_PropertyException{reason = Reason, failing_property_name = Name}]
- Reason = invalid_property_name | conflicting_property | property_not_found | unsupported_type_code | unsupported_property | unsupported_mode | fixed_property | read_only_property
- PropertySet = #objref

This operation creates a new `PropertySet` with specific initial properties.

cosProperty

Erlang Module

To get access to the record definitions for the structures use:
`-include_lib("cosProperty/include/*.hrl").`

This module contains the functions for starting and stopping the application.

Exports

`install() -> Return`

Types:

- Return = ok | {'EXIT', Reason}

This operation installs the cosProperty application in the IFR.

`install_db() -> Return`

Types:

- Return = ok | {'EXIT', Reason}

This operation installs data in mnesia necessary for running the cosProperty application.

`uninstall() -> Return`

Types:

- Return = ok | {'EXIT', Reason}

This operation removes all data in the IFR related to the cosProperty application.

`uninstall_db() -> Return`

Types:

- Return = ok | {'EXIT', Reason}

This operation removes all data from mnesia related to the cosProperty application.

`start() -> Return`

Types:

- Return = ok | {error, Reason}

This operation starts the cosProperty application.

`start_SetDefFactory() -> Return`

Types:

- Return = Factory | {'EXCEPTION', E}
- Factory = CosPropertyService::PropertySetDefFactory reference.

This operation starts a PropertySetDef Factory.

`start_SetFactory() -> Return`

Types:

- Return = Factory | {'EXCEPTION', E}
- Factory = CosPropertyService::PropertySetDefFactory reference.

This operation starts a PropertySet Factory.

`stop_SetDefFactory(Factory) -> Return`

Types:

- Factory = CosPropertyService::PropertySetDefFactory reference.
- Return = ok | {'EXCEPTION', E}

This operation stops the supplied PropertySetDef Factory.

`stop_SetFactory(Factory) -> Return`

Types:

- Factory = CosPropertyService::PropertySetFactory reference.
- Return = ok | {'EXCEPTION', E}

This operation stops the supplied PropertySet Factory.

`stop() -> Return`

Types:

- Return = ok | {error, Reason}

This operation stops the cosProperty application.

Index of Modules and Functions

Modules are typed in *this way*.
Functions are typed in `this way`.