cosNotification Application

version 1.0
Typeset in \LaTeX{} from SGML source using the DOCBUILDER 3.2.2 Document System.
Chapter 1

cosNotification User’s Guide

The cosNotification application is an Erlang implementation of the OMG CORBA Notification Service.

1.1 The cosNotification Application

1.1.1 Content Overview

The cosNotification documentation is divided into three sections:

- **PART ONE - The User’s Guide**
  Description of the cosNotification Application including services and a small tutorial demonstrating the development of a simple service.

- **PART TWO - Release Notes**
  A concise history of cosNotification.

- **PART THREE - The Reference Manual**
  A quick reference guide, including a brief description, to all the functions available in cosNotification.

1.1.2 Brief Description of the User’s Guide

The User’s Guide contains the following parts:

- cosNotification overview
- cosNotification installation
- A tutorial example

1.2 Introduction to cosNotification

1.2.1 Overview

The cosNotification application is a Notification Service compliant with the OMG Notification Service CosNotification.

\(^1\)URL: http://www.omg.org
Chapter 1: cosNotification User's Guide

Purpose and Dependencies

cosNotification is dependent on Orber-3.1.7 or later, which provides CORBA functionality in an Erlang environment, cosTime-1.0.1 or later and IDL-files to be compiled using IC-4.0.4 or later.

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 cosNotification

1.3.1 Installation Process

This chapter describes how to install cosNotificationApp [page 87] in an Erlang Environment.

Preparation

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

Configuration

When using the Notification Service the cosNotification application first must be installed using cosNotificationApp:install() or cosNotificationApp:install(Seconds), followed by cosNotificationApp:start().

Then the Event Channel Factory [page 42] must be started:

- cosNotificationApp:start_global_factory() - starts and returns a reference to a factory using default configuration parameters. This operation should be used for a multi-node Orber.
- cosNotificationApp:start_global_factory(Options) - starts and returns a reference to a factory using given configuration parameters. This operation should be used for a multi-node Orber.
- cosNotificationApp:start_factory() - starts and returns a reference to a factory using default configuration parameters.
- cosNotificationApp:start_factory(Options) - starts and returns a reference to a factory using given configuration parameters.

The following options exist:

- {pullInterval, Seconds} - determine how often Proxy Pull Consumers will check for new events with the client application. The default value is 20 seconds.
- {filterOp, OperationType} - determine which type of Administrator objects should be started, i.e., 'OR_OP' or 'AND_OP'. The default value is 'OR_OP'.
- {timeService, TimeServiceObj | 'undefined'} - to be able to use Start and/or Stop QoS this option must be used. See the function start_time_service/2 in the cosTime application. The default value is 'undefined'.

2 cosNotification Application
1.4: The Notification Service Components

- \{\text{filterOp}, \text{OperationType}\} - determine which type of Administrator objects should be started, i.e., 'OR\text{OP}' or 'AND\text{OP}'. The default value is 'OR\text{OP}'.
- \{\text{gcTime}, \text{Seconds}\} - this option determines how often, for example, proxies will garbage collect expired events. The default value is 60.
- \{\text{gcLimit}, \text{Amount}\} - determines how many events will be stored before, for example, proxies will garbage collect expired events. The default value is 50. This option is tightly coupled with the QoS property \text{MaxEventsPerConsumer}, i.e., the \text{gcLimit} should be less than \text{MaxEventsPerConsumer} and greater than 0.

The Factory is now ready to use. For a more detailed description see Examples [page 16].

1.4 The Notification Service Components

1.4.1 The Notification Service Components

This chapter describes the Notification Service Components and how they interact.

Components

There are seven components in the OMG Notification Service architecture. These are described below:
• **Event Channel**: acts as a factory for Administrator objects. Allows clients to set Administrative Properties.

• **Supplier Administrators**: acts as a factory for Proxy Consumers. Administrators are started as `AND_OP`- or `OR_OP`-type, which determines if events must be validated using both the Administrators associated Filter and/or its Proxy children Filters.

• **Consumer Administrators**: acts in the same way as Supplier Administrators but handle Proxy Suppliers.

• **Consumer Proxy**: is connected to a client application. Can be started as Pull or Push object. If the proxy is Push style the client application must push events to the Proxy, otherwise the Proxy is supposed to Pull events. The `CosNotification::AdminProperties` is used to set the pacing interval.

• **Supplier Proxy**: Acts in a similar way as the Consumer Proxy, but if started as a Push proxy it will push events to the client application.

• **Filters**: used to filter events. May be associated with Proxies and Administrators.
Mapping Filters: Used to override events Quality of Service settings. Can only be associated with Consumer Administrators and Proxy Suppliers.

When a Proxy is started it is set to accept CORBA::Any, CosNotification::StructuredEvent or CosNotification::EventBatch (a sequence of structured events).

If a Proxy is supposed to deliver structured events to a client application and receives an CORBA::Any event, the event is converted to a structured event with type name set to "%ANY" and the event is stored in remainder of body.

If a Proxy is supposed to deliver CORBA::Any events to a client application and receives a structured event, the event is stored in an Any type. The Any Type Code will be equal to the CosNotification::StructuredEvent Type Code.

1.5 Filters and the Constraint Language BNF
1.5.1 Filters and the Constraint Language BNF

This chapter describes the grammar supported by CosNotifyFilter [page 76] and CosNotifyFilterMappingFilter [page 83], and how to create and use filter objects.

How to create filter objects

To be able to filter events we must create a filter and associate it with one, or more, of the administrative or proxy objects. In the example below, we choose to associate the filter with a ConsumerAdmin object.

FilterFactory = cosNotificationApp:start_filter_factory(),
Filter = 'CosNotifyFilter_FilterFactory':
    create_filter(FilterFactory,"EXTENDED_TCL"),
ConstraintInfoSeq = 'CosNotifyFilter_Filter':
    add_constraints(Filter, ConstraintExpSeq),
FilterID = 'CosNotifyChannelAdmin_ConsumerAdmin':
    add_filter(AdminConsumer, Filter),

"EXTENDED_TCL" is the only grammar supported by Order Notification Service. Depending on which operation type the Admin object uses, i.e., 'AND.OP' or 'OR.OP', events will be tested using the associated filter. The operation properties are:

- 'AND.OP' - must be approved by the proxy's and its parent admin's filters. If all filters associated with an object (Admin or Proxy) return false the event will be discarded. In this situation it is pointless to try and verify with the other object's associated filters since the outcome still would be the same.
- 'OR.OP' - if one of the object's (Admin or Proxy) filters return true, the event will not be checked against any other filter associated with a proxy or its parent admin. If a object's associated filters all return false, the event will be forwarded to related proxies/admins, and tested against any associated filters.

Initially, filters are empty and will always return true. Hence, we must add constraints by using 'CosNotifyFilter_Filter':add_constraints/2. As input, the second argument must be a sequence of:
Chapter 1: cosNotification User's Guide

The `CosNotifyFilter_ConstraintExp`{
  event_types = [#'CosNotification_EventType'{
    domain_name = string(),
    type_name = string()}],
  constraint_expr = string()}

The event types describes which types of events that should be matched using the associated constraint_expr.

If a constraint expression is supposed to apply for all events, then the type_name can be set to the special event type %ALL in a constraint’s event type sequence. The domain_name should be "" or "*".

In the following sections we will take a closer look on how to write constraint expressions.

The CosNotification Constraint Language

The constraint language supported by the Notification Service is:

```xml
<constraint> := /* empty */
  | <bool>
  <bool> := <bool_or>
  <bool_or> := <bool_or> or <bool_and>
  | <bool_and>
  <bool_and> := <bool_and> and <bool_compare>
  | <bool_compare>
  <bool_compare> := <expr_in> == <expr_in>
  | <expr_in> != <expr_in>
  | <expr_in> < <expr_in>
  | <expr_in> <= <expr_in>
  | <expr_in> > <expr_in>
  | <expr_in> >= <expr_in>
  | <expr_in>
  <expr_in> := <expr_twiddle> in <Ident> /* sequence only */
  | <expr_twiddle>
  | <expr_twiddle> in $ <Component> /* sequence only */
  <expr_twiddle> := <expr> ~ <expr> /* string data types only */
  | <expr>
  <expr> := <expr> + <term>
  | <expr> - <term>
  | <term>
  <term> := <term> * <factor_not>
  | <term> / <factor_not>
  | <factor_not>
  <factor_not> := not <factor>
```

6 cosNotification Application
1.5: Filters and the Constraint Language BNF

| <factor> |
<factor> ::= ( <bool_or> )
| exist <Ident>
| <Ident>
| <Number>
| - <Number>
| <String>
| TRUE
| FALSE
| + <Number>
| exist $ <Component>
| $ <Component>
| default $ <Component> /* discriminated unions only */

<Component> ::= /* empty */
| . <CompDot>
| <CompArray>
| <CompAssoc>
| <Ident> <CompExt> /* run-time variable */

<CompExt> ::= /* empty */
| . <CompDot>
| <CompArray>
| <CompAssoc>

<CompDot> ::= <Ident> <CompExt>
| <CompPos>
| <UnionPos>
| _length /* only valid for arrays or sequences */
| _d /* discriminated unions only */
| _type_id /* only valid if possible to obtain */
| _repos_id /* only valid if possible to obtain */

<CompArray> ::= [ <Digits> ] <CompExt>

<CompAssoc> ::= ( <Ident> ) <CompExt>

<CompPos> ::= <Digits> <CompExt>

<UnionPos> ::= ( <UnionVal> ) <CompExt>

<UnionVal> ::= /* UnionVal */
| <Digits>
| - <Digits>
| + <Digits>
| <String>

/* Character set issues */
<Ident> ::=<Leader> <FollowSeq>
| \ < Leader> <FollowSeq>

<FollowSeq> ::= /* empty */
Chapter 1: cosNotification User's Guide

| <FollowSeq> <Follow>

<Number> ::= <Mantissa>
    | <Mantissa> <Exponent>

<Mantissa> ::= <Digits>
    | <Digits> .
    | . <Digits>
    | <Digits> . <Digits>

<Exponent> ::= <Exp> <Sign> <Digits>

<Sign> ::= +
    | -

<Exp> ::= E
    | e

<Digits> ::= <Digits> <Digit>
    | <Digit>

<String> ::= ' <TextChars> '

<TextChars> ::= /* <empty> */
    | <TextChars> <TextChar>

<TextChar> ::= <Alpha>
    | <Digit>
    | <Other>
    | <Special>

<Special> ::= \`
    | `'

<Leader> ::= <Alpha>

<Follow> ::= <Alpha>
    | <Digit>
    | _

<Alpha> is the set of alphabetic characters [A-Za-z]
<Digit> is the set of digits [0-9]
<Other> is the set of ASCII characters that are not <Alpha>, <Digit>, or <Special>

In the absence of parentheses, the following precedence relations hold:

1. (), exist, default, unary-sign
2. not
3. *, /
4. +, -
5. -
6. in
1.5: Filters and the Constraint Language BNF

7. ==, !=, <, <=, >, >=
8. and
9. or

The Constraint Language Data Types

The Notification Service Constraint Language defines how to write constraint expressions, which can be used to filter events. The representation does, however, differ slightly from ordinary Erlang terms. When creating a ConstraintExp, the field `constraint_expr` must be set to contain a string, e.g., "1 < 2". The Notification Service Constraint Language is designed to be able to filter structured and unstructured events using the same constraint expression. The Constraint Language Types and Operations can be divided into two sub-groups:

- Basic - arithmetics, strings, constants, numbers etc.
- Complex - accessing members of complex data types, such as unions.

Some of the basic types, e.g., integer, are self explanatory. Hence, they are not described further.

<table>
<thead>
<tr>
<th>Type/Operation</th>
<th>Examples</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>&quot;'MyString'&quot;</td>
<td>Strings are represented as a sequence of zero or more <code>&lt;TextChar&gt;</code>s enclosed in single quotes, e.g., 'string'.</td>
</tr>
<tr>
<td>&quot;</td>
<td>&quot;'String1' ~ 'String2'&quot;</td>
<td>The operator ~ is called the sub-string operator and mean &quot;String1 is contained within String2&quot;.</td>
</tr>
<tr>
<td>boolean</td>
<td>&quot;TRUE == (('lang' ~ 'Erlang' + 'fun' ~ 'functional') &gt;= 2)&quot;</td>
<td>Booleans may only be TRUE or FALSE, i.e., only capital letters. Expressions which evaluate to TRUE or FALSE can be summed up and matched, where TRUE equals 1 and FALSE 0.</td>
</tr>
<tr>
<td>sequence</td>
<td>&quot;myIntegerSequence[2]&quot;</td>
<td>The BNF use C/C++ notation, i.e., the example will return the third element.</td>
</tr>
<tr>
<td>length</td>
<td>&quot;myIntegerSequence.length&quot;</td>
<td>Returns the length of an sequence or array.</td>
</tr>
<tr>
<td>in</td>
<td>&quot;'Erlang' in $.FunctionalLanguages-StringSeq&quot;</td>
<td>Returns TRUE if a given element is found in the given sequence. The element must be of a simple type and the same as the sequence is defined to contain.</td>
</tr>
<tr>
<td>$</td>
<td>&quot;$ == 40&quot;</td>
<td>Denote the current event as well as any run-time variables. If the event is unstructured and its contained value 40, the example will return TRUE.</td>
</tr>
</tbody>
</table>

**continued ...**
The structure member operator . may be used to reference its members when the data refers to a named structure, discriminated union, or CORBA::Any data structure.

<table>
<thead>
<tr>
<th>Property</th>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>&quot;.MyStructMember == 40&quot;</td>
<td>Returns the unscoped IDL type name of the component. This operation is only valid if said information can be obtained.</td>
</tr>
<tr>
<td>.type_id</td>
<td>&quot;.type_id == 'MyStruct'&quot;</td>
<td>Returns the RepositoryId of the component. This operation is only valid if said information can be obtained.</td>
</tr>
<tr>
<td>.repos_id</td>
<td>&quot;.repos_id == 'IDL:MyModule/MyStruct:1.0'&quot;</td>
<td>May only be used when accessing discriminated unions and refers to the discriminator.</td>
</tr>
<tr>
<td>.d</td>
<td>&quot;.eventUnion..d&quot;</td>
<td>To avoid that a filtering of an event fails due to that, for example, we try to compare a union discriminator which does not exist, we can use this operator.</td>
</tr>
<tr>
<td>exist</td>
<td>&quot;exist $.eventUnion..d and $.eventUnion..d == 10&quot;</td>
<td>If the .d operation is in conjunction with the default operation, TRUE will be returned if the union has a default member that is active.</td>
</tr>
<tr>
<td>default</td>
<td>&quot;default $.eventUnion..d&quot;</td>
<td>When the component refers to a union, with one of the cases defined as case 0: short zero; we use 0 or 'zero'. The result of the example is TRUE if the union has a discriminator set to 0 and the value 5. If more than one case is defined to be 'zero', &quot;.('zero')&quot; accepts both; &quot;.(0)&quot; only returns TRUE if the discriminator is set to 0. Leaving out the identifier, i.e., &quot;.()&quot;, refers to the default value.</td>
</tr>
</tbody>
</table>

continued ...
Table 1.1: Table 1: Type & Operator Examples

In the next section we will take a closer look at how it is possible to write constraints using different types of notation etc.

Accessing Data In Events

To filter events, the supplied constraints must describe the contents of the events and desired values. We can, for example, state that we are only interested in receiving events which are of type CommunicationsAlarm. To be able to achieve this, the constraint must contain information that points out which fields to compare with. Figure one illustrates a conceptual overview of a structured event. The exact definition is found in the CosNotification.idl file.

![Figure 1: The structure of a structured event.](cosNotification Application)

The Notification Service supports different constraint expressions notation:

- Fully scoped, e.g., 
  
  \[
  {.header.fixed_header.event_type.type.name == 'CommunicationsAlarm'}
  \]

- Short hand, e.g., 
  
  \[
  $type.name == 'CommunicationsAlarm'
  \]

- Positional Notation, e.g., 
  
  \[
  $.0.0.1 == 'CommunicationsAlarm'
  \]
Note:
Which notation to use is up to the user, however, the fully scoped may be easier to understand, but in some cases, if received from an ORB that do not populate ID:s of named parts, the positional notation is the only option.

Note:
If a constraint, which access fields in a structured event structure, is supposed to handle unstructured events as well, the CORBA::Any must contain the same type of members.

How to filter against the fixed header fields, is described in the table below.

<table>
<thead>
<tr>
<th>Field</th>
<th>Fully Scoped Constraint</th>
<th>Short Hand Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>type_name</td>
<td>&quot;$.header.fixed_header.event_type.type_name == 'Type'&quot;</td>
<td>&quot;$type_name == 'Type'&quot;</td>
</tr>
<tr>
<td>domain_name</td>
<td>&quot;$.header.fixed_header.event_type.domain_name == 'Domain'&quot;</td>
<td>&quot;$domain_name == 'Domain'&quot;</td>
</tr>
<tr>
<td>event_name</td>
<td>&quot;$.header.fixed_header.event_name == 'Event'&quot;</td>
<td>&quot;$event_name == 'Event'&quot;</td>
</tr>
</tbody>
</table>

Table 1.2: Table 2: Fixed Header Constraint Examples

If we are only interested in receiving events regarding 'Domain', 'Event' and 'Type', the constraint can look like "$domain_name == 'Domain' and $event_name == 'Event' and $type_name == 'Type'".

The variable event header consists of a sequence of name-value pairs. One way to filter on these are to use a constraint that looks like "$(.header.variable_header[1].name == 'priority' and $.header.variable_header[1].value > 0)". An easier way to accomplish the same result is to use a constraint that treats the name-value pair as an associative array, i.e., when given a name the corresponding value is returned. Hence, instead we can use "$.header.variable_header(priority) > 0".

Accessing the event body is done in the same way as for the event header fields. The user must, however, be aware of, that if a run-time variable ($variable) is used data in the event header may take precedence. The order of precedence is:

1. Reserved, e.g., $curtime
2. A simple-typed member of $.header.fixed_header.
3. Properties in $.header.variable_header.
4. Properties in $.filterable_data.
5. If no match is found it is translated to $.variable.
1.6: Quality Of Service and Admin Properties

Mapping Filters

Mapping Filters may only be associated with Consumer Administrators or Proxy Suppliers. The purpose of a Mapping Filter is to override Quality of Service settings. Initially, Mapping Filters are empty and will always return true. Hence, we must add constraints by using 'CosNotifyFilter_MAPPINGFILTER':add_mapping_constraints/2. If a constraint matches, the associated value will be used instead of the realted Quality of Service system settings.

As input, the second argument must be a sequence of:

`#'CosNotifyFilter.MappingConstraintPair'{
  constraint_expression = #'CosNotifyFilter.ConstraintExp'{
    event_types = [#'CosNotification.EventType'{
      domain_name = string(),
      type_name = string()},
    constraint_expr = string(),
    result_to_set = any()}
  result_to_set = any()}

1.6 Quality Of Service and Admin Properties

1.6.1 Quality Of Service and Admin Properties

This chapter explains the allowed properties for CosNotification_QoSAdmin [page 34] and CosNotification_AdminPropertiesAdmin [page 33].

Quality Of Service

The cosNotification application supports the following QoS settings:

<table>
<thead>
<tr>
<th>QoS</th>
<th>Range</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventReliability</td>
<td>BestEffort</td>
<td>BestEffort</td>
</tr>
<tr>
<td>ConnectionReliability</td>
<td>BestEffort/Persistent</td>
<td>BestEffort</td>
</tr>
<tr>
<td>Priority</td>
<td>+/-32767</td>
<td>0</td>
</tr>
<tr>
<td>OrderPolicy</td>
<td>Any-, Fifo-, Priority- and Deadline-Order</td>
<td>PriorityOrder</td>
</tr>
<tr>
<td>DiscardPolicy</td>
<td>RejectNewEvents, Any-, Fifo-, Lifo-, Priority- and Deadline-Order</td>
<td>RejectNewEvents</td>
</tr>
<tr>
<td>MaximumBatchSize</td>
<td>long() &gt; 0</td>
<td>100</td>
</tr>
<tr>
<td>PacingInterval</td>
<td>TimeBase::TimeT (see cosTime)</td>
<td>20 Seconds</td>
</tr>
<tr>
<td>StartTimeSupported</td>
<td>boolean</td>
<td>false</td>
</tr>
<tr>
<td>StopTimeSupported</td>
<td>boolean</td>
<td>false</td>
</tr>
<tr>
<td>MaxEventsPerConsumer</td>
<td>long() &gt; 0</td>
<td>100</td>
</tr>
<tr>
<td>Timeout</td>
<td>TimeBase::TimeT (see cosTime)</td>
<td>No timeout</td>
</tr>
</tbody>
</table>

Table 1.3: Table 1: Supported QoS settings
Comments on the table ‘Supported QoS settings’:

**EventReliability** To allow Persistent EventReliability, every event must be stored in a stable storage which would create a relatively huge overhead. Hence, only BestEffort is supported.

**ConnectionReliability** If this QoS is set to BestEffort and a client object returns anything other than ok to its associated Proxy, the Proxy will discard all events and terminate. Using Persistent and anything other than ok is returned, events will be dropped but the proxy will retry later when next delivery is due. A child may not have Persistent while its parent has BestEffort QoS set, e.g., Proxy vs. Admin.

**Priority** This QoS will treat all events as if they have the Priority equal to current value, unless the event itself contains a Priority setting, this event will be treated accordingly. Note: for this property to have any effect, the DiscardPolicy and/or OrderPolicy must be set to PriorityOrder.

**OrderPolicy** If set to PriorityOrder, events with the highest Priority will be delivered first. Deadline order will forward events with shortest expiry time first. If two events have the same priority, they will be delivered in FIFO-order.

**DiscardPolicy** If set to PriorityOrder and MaxEventsPerConsumer limit is reached, events with the lowest Priority will be discarded first. Deadline order will discard events with shortest expiry time first.

**MaximumBatchSize** Only valid if the object is supposed to handle a sequence of structured events and determines the largest amount of events that may be passed each time.

**PacingInterval** Determines how long an object will wait before forwarding a structured event sequence of length equal to, or less than MaximumBatchSize.

**StartTimeSupported** If set to true events which contains the QoS Property StartTime (TimeBase::UtcT - absolute time) will not be delivered until the StartTime value have been exceeded. See also the cosTime application.

**StopTimeSupported** If set to true, events which contain the QoS Properties StopTime (TimeBase::UtcT - absolute time) or Timeout (TimeBase::TimeT - relative time) will be discarded if the object has not been able to deliver the event in time. See also the cosTime application.

**MaxEventsPerConsumer** The maximum number of events the associated object may store before discarding events in the way described by the DiscardPolicy.

**Timeout** If this QoS property is not included in the event, and the Property StopTimeSupported equals true, this setting will be applied if events cannot be delivered within its time limit.

**Warning:**
Several of the above QoS Properties can be changed during run-time but we strongly advice not to since, if a relatively large amount of events are waiting for delivery, some of the QoS settings would require a total reorder of the events. The QoS property ConnectionReliability may never be updated during run-time since it may cause deadlock. Run-time, in this case, means activating the Channel by sending the first event.

**Setting Quality Of Service**
Assume we have a Consumer Admin object which we want to change the current Quality of Service. Typical usage:
1.6: Quality Of Service and Admin Properties

QoSPersistent =
[#'CosNotification_Property'
  {name='CosNotification':'ConnectionReliability'(),
   value=any:createCommand(order_tc:short(),
                     'CosNotification':'Persistent'())},
'CosNotification_QoSAdmin':set_qos(Ch, QoSPersistent),

If it is not possible to set the requested QoS the UnsupportedQoS exception is raised, which includes a sequence of PropertyError's describing which QoS, possible range and why is not allowed. The error codes are:

- UNSUPPORTED_PROPERTY - QoS not supported for this type of target object.
- UNAVAILABLE_PROPERTY - due to current QoS settings the given property is not allowed.
- UNSUPPORTED_VALUE - property value out of range; valid range is returned.
- UNAVAILABLE_VALUE - due to current QoS settings the given value is not allowed; valid range is returned.
- BAD_PROPERTY - unrecognized property.
- BAD_TYPE - type of supplied property is incorrect.
- BAD_VALUE - illegal value.

The CosNotification_QoSAdmin interface also supports an operation called validate_qos/2. The purpose of this operation is to check if a QoS setting is supported by the target object and if so, the operation returns additional properties which could be optionally added as well.

Admin Properties

The cosNotification application supports the following Admin Properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Range</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxQueueLength</td>
<td>0 or long()</td>
<td>0</td>
</tr>
<tr>
<td>MaxConsumers</td>
<td>long() &gt;= 0</td>
<td>0</td>
</tr>
<tr>
<td>MaxSuppliers</td>
<td>long() &gt;= 0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1.4: Table 2: Supported Admin Properties

According to the OMG specification the default values for Admin Properties is supposed to be 0, which means that no limit applies to these properties.

Note:
Admin Properties can only be set on a Channel Object level, i.e., they will not have an impact on any Admin or Proxy Objects. Currently, setting the Admin Property MaxQueueLength have no effect since we cannot discard events accordingly to the Quality of Service Property DiscardPolicy.
Chapter 1: cosNotification User’s Guide

1.7 cosNotification Examples

1.7.1 A Tutorial on How to Create a Simple Service

Interface Design

To use the cosNotification application clients must be implemented. There are twelve types of clients:

- Structured Push Consumer
- Sequence Push Consumer
- Any Push Consumer
- Structured Pull Consumer
- Sequence Pull Consumer
- Any Pull Consumer
- Structured Push Supplier
- Sequence Push Supplier
- Any Push Supplier
- Structured Pull Supplier
- Sequence Pull Supplier
- Any Pull Supplier

The interfaces for these participants are defined in CosNotification.idl and CosNotifyComm.idl.

Generating a Client Interface

We start by creating an interface which inherits from the correct interface, e.g., CosNotifyComm::SequencePushConsumer. Hence, we must also implement all operations defined in the SequencePushConsumer interface. The IDL file could look like:

```idl
#ifndef _MYCLIENT_IDL
#define _MYCLIENT_IDL
#include <CosNotification.idl>
#include <CosNotifyComm.idl>

module myClientImpl {

  interface ownInterface: CosNotifyComm::SequencePushConsumer {

    void ownFunctions(in any NeededArguments)
      raises(Systemexceptions,OwnExceptions);

  };
};

#endif
```
Run the IDL compiler on this file by calling the `ic:gen/1` function. This will produce the API named `myClientImpl OWNInterface.erl`. After generating the API stubs and the server skeletons it is time to implement the servers and if no special options are sent to the IDL compiler the file name is `myClientImpl OWNInterface_impl.erl`.

The callback module must contain the necessary functions inherited from `CosNotification.idl` and `CosNotifyComm.idl`.

**How to Run Everything**

Below is a short transcript on how to run `cosNotification`.

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

%% If cosEvent not installed before it is necessary to do it now.
cosNotificationApp:install_event(30),

%% Install cosNotification in the IFR.
cosNotificationApp:install(30),

%% Register the application specific Client implementations
%% in the IFR.
'oe_myClientImpl': 'oe_register',

%% Start the cosNotification application.
cosNotificationApp:start(),

%% Start a factory using the default configuration
ChFac = cosNotificationApp:start_factory(),
%% ... or use configuration parameters.
ChFac = cosNotificationApp:start_factory([],

%% Create a new event channel.
Ch = 'CosNotifyChannelAdmin_EventChannelFactory':
    create_channel(ChFac, DefaultQoS, DefaultAdmin),

%% Retrieve a SupplierAdmin and a Consumer Admin.
{AdminSupplier, ASID} =
    'CosNotifyChannelAdmin_EventChannel':new_for_suppliers(Ch, 'OR_OP'),
{AdminConsumer, ACID} =
    'CosNotifyChannelAdmin_EventChannel':new_for_consumers(Ch, 'OR_OP'),

%% Use the corresponding Admin object to get access to wanted Proxies

%% Create a Push Consumer Proxie, i.e., the Client Push Supplier will
%% push events to this Proxy.
{StructuredProxyPushConsumer, ID11} =
    'CosNotifyChannelAdmin_SupplierAdmin':
    obtain_notification_push_consumer(AdminSupplier, 'STRUCTURED_EVENT'),
```

**cosNotification Application**
%% Create Push Suppliers Proxies, i.e., the Proxy will push events to the
%% registered Push Consumers.
{ProxyPushSupplier,I4D}='CosNotifyChannelAdmin_ConsumerAdmin':
  obtain_notification_push_supplier(AdminConsumer, 'ANY_EVENT'),
{StructuredProxyPushSupplier,ID5}='CosNotifyChannelAdmin_ConsumerAdmin':
  obtain_notification_push_supplier(AdminConsumer, 'STRUCTURED_EVENT'),
{SequenceProxyPushSupplier,ID6}='CosNotifyChannelAdmin_ConsumerAdmin':
  obtain_notification_push_supplier(AdminConsumer, 'SEQUENCE_EVENT'),

%% Create application Clients. We can, for example, start the Clients
%% our selves or look them up in the naming service. This is application
%% specific.
SupplierClient = ...
ConsumerClient1 = ...
ConsumerClient2 = ...
ConsumerClient3 = ...

%% Connect each Client to corresponding Proxy.
'CosNotifyChannelAdmin_StructuredProxyPushConsumer':
  connect_structured_push_supplier(StructuredProxyPushConsumer, SupplierClient),
'CosNotifyChannelAdmin_ProxyPushSupplier':
  connect_any_push_consumer(ProxyPushSupplier, ConsumerClient1),
'CosNotifyChannelAdmin_StructuredProxyPushSupplier':
  connect_structured_push_consumer(StructuredProxyPushSupplier, ConsumerClient2),
'CosNotifyChannelAdmin_SequenceProxyPushSupplier':
  connect_sequence_push_consumer(SequenceProxyPushSupplier, ConsumerClient3),

The example above, exemplifies a notification system where the SupplierClient in some way generates
event and pushes them to the proxy. The push supplier proxies will eventually push the events to each
ConsumerClient.

1.8  cosNotification Release Notes

1.8.1  cosNotification 1.0.4, Release Notes

Improvements and new features

- Fixed bugs and malfunctions
  - When passing event sequences, the PushSuppliers and PullSuppliers could crash if the objects had
    Filter objects associated and only a subset of the sequences where approved.
    Own Id: OTP-4099
  - SupplierAdmin’s did not filter any events, even though Filter objects had been attached to the
    SupplierAdmin.
    Own Id: OTP-4098
If one used the `get_default_supplier_admin`/1, exported by the CosNotifyChannelAdmin_EventChannel-module, it resulted in a loop which overloaded the channel. This is no longer the case.

Own Id: OTP-4086

If one used the `get_default_filter_factory`/1, exported by the CosNotifyChannelAdmin_EventChannel-module, a new instance was created each time. Now fixed.

Own Id: OTP-4092

Incompatibilities

- The include paths for `CosNotification.idl` have been changed. Hence, if you include this file in your own IDL-files you must update your paths to also point to where the `cosEvent` IDL-files are stored.

  Own Id: OTP-4093

Known bugs and problems

1.8.2 cosNotification 1.0.3, Release Notes

Improvements and new features

- It is now possible to start global channel factories.

  Own Id: OTP-4078

- The Orber, version 3.2.5 or later, configuration parameter orber_debug_level can now be used to generate reports when abnormal situations occurs. For more information consult the Orber User’s Guide. Note, it is not recommended to use this option for delivered systems since some of the reports is not to be considered as errors. The value of orber_debug_level must be 3, or higher, for reports to be generated.

  Own Id: OTP-4077, OTP-3962

Fixed bugs and malfunctions

- When using the cosEvent API accessing a cosNotification admins the objects returned by the functions obtain_push_supplier, obtain_pull_supplier, obtain_push_consumer and obtain_pull_consumer was not of the correct type. Due to the interface change it is not possible to upgrade during runtime.

  Own Id: OTP-4079

Incompatibilities

- 

Known bugs and problems

- 

1.8: cosNotification Release Notes

- 

cosNotification Application
1.8.3  cosNotification 1.0.2, Release Notes

Improvements and new features

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

Fixed bugs and malfunctions

- 

Incompatibilities

- 

Known bugs and problems

-
cosNotification Reference Manual

Short Summaries

- Erlang Module `CosNotification_AdminPropertiesAdmin` [page 33] – This module implements the OMG CosNotification::AdminPropertiesAdmin interface.
- Erlang Module `CosNotification_QoSAdmin` [page 34] – This module implements the OMG CosNotification::QoSAdmin interface.
- Erlang Module `CosNotifyChannelAdmin_ConsumerAdmin` [page 36] – This module implements the OMG CosNotifyChannelAdmin::ConsumerAdmin interface.
- Erlang Module `CosNotifyChannelAdmin_EventChannel` [page 39] – This module implements the OMG CosNotifyChannelAdmin::EventChannel interface.
- Erlang Module `CosNotifyChannelAdmin_EventChannelFactory` [page 42] – This module implements the OMG CosNotifyChannelAdmin::EventChannelFactory interface.
- Erlang Module `CosNotifyChannelAdmin_ProxyConsumer` [page 43] – This module implements the OMG CosNotifyChannelAdmin::ProxyConsumer interface.
- Erlang Module `CosNotifyChannelAdmin_ProxyPullConsumer` [page 45] – This module implements the OMG CosNotifyChannelAdmin::ProxyPullConsumer interface.
- Erlang Module `CosNotifyChannelAdmin_ProxyPullSupplier` [page 47] – This module implements the OMG CosNotifyChannelAdmin::ProxyPullSupplier interface.
- Erlang Module `CosNotifyChannelAdmin_ProxyPushConsumer` [page 49] – This module implements the OMG CosNotifyChannelAdmin::ProxyPushConsumer interface.
- Erlang Module `CosNotifyChannelAdmin_ProxyPushSupplier` [page 50] – This module implements the OMG CosNotifyChannelAdmin::ProxyPushSupplier interface.
- Erlang Module `CosNotifyChannelAdmin_ProxySupplier` [page 52] – This module implements the OMG CosNotifyChannelAdmin::ProxySupplier interface.
- Erlang Module `CosNotifyChannelAdmin_SequenceProxyPullConsumer` [page 55] – This module implements the OMG CosNotifyChannelAdmin::SequenceProxyPullConsumer interface.
Erlang Module CosNotifyChannelAdmin

- SequenceProxyPullSupplier [page 57] - This module implements the OMG CosNotifyChannelAdmin::SequenceProxyPullSupplier interface.
- SupplierAdmin [page 59] - This module implements the OMG CosNotifyChannelAdmin::SupplierAdmin interface.
- SequenceProxyPushConsumer [page 62] - This module implements the OMG CosNotifyChannelAdmin::SequenceProxyPushConsumer interface.
- SequenceProxyPushSupplier [page 64] - This module implements the OMG CosNotifyChannelAdmin::SequenceProxyPushSupplier interface.
- StructuredProxyPullConsumer [page 66] - This module implements the OMG CosNotifyChannelAdmin::StructuredProxyPullConsumer interface.
- StructuredProxyPullSupplier [page 68] - This module implements the OMG CosNotifyChannelAdmin::StructuredProxyPullSupplier interface.
- StructuredProxyPushConsumer [page 70] - This module implements the OMG CosNotifyChannelAdmin::StructuredProxyPushConsumer interface.
- StructuredProxyPushSupplier [page 72] - This module implements the OMG CosNotifyChannelAdmin::StructuredProxyPushSupplier interface.
- NotifyPublish [page 74] - This module implements the OMG CosNotifyComm::NotifyPublish interface.
- NotifySubscribe [page 75] - This module implements the OMG CosNotifyComm::NotifySubscribe interface.
- Filter [page 76] - This module implements the OMG CosNotifyFilter::Filter interface.
- FilterAdmin [page 80] - This module implements the OMG CosNotifyFilter::FilterAdmin interface.
- FilterFactory [page 82] - This module implements the OMG CosNotifyFilter::FilterFactory interface.
- MappingFilter [page 83] - This module implements the OMG CosNotifyFilter::MappingFilter interface.
- cosNotificationApp [page 87] - The main module of the cosNotification application.

CosNotification_AdminPropertiesAdmin

The following functions are exported:

- get_admin(Object) -> AdminProperties [page 33] - Return a list of AdminProperties associated with the target object.
- set_admin(Object, AdminProperties) -> Reply [page 33] - Update the AdminProperties for the target object.
CosNotification_QoSAdmin

The following functions are exported:

- `get_qos(Object) -> Reply`
  - [page 34] Return a list of name-value pairs which encapsulates the current QoS settings for the target object
- `set_qos(Object, QoS) -> Reply`
  - [page 34] Change the QoS settings for the target object
- `validate_qos(Object, QoS) -> Reply`
  - [page 35] Validate if the supplied QoS properties is valid for the target object

CosNotifyChannelAdmin_ConsumerAdmin

The following functions are exported:

- `get_MyID(ConsumerAdmin) -> AdminID`
  - [page 36] Return the target object’s Id
- `get_MyChannel(ConsumerAdmin) -> Channel`
  - [page 36] Return the ancestor channel
- `get_MyOperator(ConsumerAdmin) -> OpType`
  - [page 36] Return the filtering schema used by the target object
- `get_priority_filter(ConsumerAdmin) -> MappingFilter`
  - [page 36] Return the associated priority MappingFilter
- `set_priority_filter(ConsumerAdmin, MappingFilter) -> ok`
  - [page 37] Set the priority MappingFilter
- `get_lifetime_filter(ConsumerAdmin) -> MappingFilter`
  - [page 37] Return the associated lifetime MappingFilter
- `set_lifetime_filter(ConsumerAdmin, MappingFilter) -> ok`
  - [page 37] Set the lifetime MappingFilter
- `get_pull_suppliers(ConsumerAdmin) -> ProxyIDSeq`
  - [page 37] Return a list of all associated pull supplier Id:s
- `get_push_suppliers(ConsumerAdmin) -> ProxyIDSeq`
  - [page 37] Return a list of all associated push supplier Id:s
- `get_proxy_supplier(ConsumerAdmin, ProxyID) -> Reply`
  - [page 37] Return the proxy supplier with matching Id
- `obtain_notification_pull_supplier(ConsumerAdmin, ConsumerType) -> Reply`
  - [page 38] Create a supplier proxy
- `obtain_pull_supplier(ConsumerAdmin) -> Proxy`
  - [page 38] Create a supplier proxy
- `obtain_notification_push_supplier(ConsumerAdmin, ConsumerType) -> Reply`
  - [page 38] Create a supplier proxy
- `obtain_push_supplier(ConsumerAdmin) -> Proxy`
  - [page 38] Create a supplier proxy
- `destroy(ConsumerAdmin) -> ok`
  - [page 38] Terminate the target object and all its children
CosNotifyChannelAdmin_EventChannel

The following functions are exported:

- `getMyFactory(Channel) -> ChannelFactory`  
  [page 39] Return the factory object which created the target object

- `getDefaultConsumerAdmin(Channel) -> ConsumerAdmin`  
  [page 39] Return the default consumer admin associated with the target object

- `getDefaultSupplierAdmin(Channel) -> SupplierAdmin`  
  [page 39] Return the default supplier admin associated with the target object

- `getDefaultFilterFactory(Channel) -> FilterFactory`  
  [page 39] Return the default filter factory associated with the target object

- `newForConsumers(Channel, OpType) -> Return`  
  [page 40] Create a new ConsumerAdmin object

- `forConsumers(Channel) -> ConsumerAdmin`  
  [page 40] Create a new ConsumerAdmin object

- `newForSuppliers(Channel, OpType) -> Return`  
  [page 40] Create a new SupplierAdmin object

- `forSuppliers(Channel) -> SupplierAdmin`  
  [page 40] Create a new SupplierAdmin object

- `getConsumeradmin(Channel, AdminID) -> ConsumerAdmin`  
  [page 41] Return the ConsumerAdmin matching AdminID

- `getSupplieradmin(Channel, AdminID) -> SupplierAdmin`  
  [page 41] Return the SupplierAdmin matching AdminID

- `getAllConsumeradmins(Channel) -> Reply`  
  [page 41] Return a list of all ConsumerAdmins, currently active, Id:s

- `getAllSupplieradmins(Channel) -> Reply`  
  [page 41] Return a list of all SupplierAdmins, currently active, Id:s

- `destroy(Channel) -> ok`  
  [page 41] Terminate the channel and all its childrens

CosNotifyChannelAdmin_EventChannelFactory

The following functions are exported:

- `create_channel(ChannelFactory, InitialQoS, InitialAdmin) -> Return`  
  [page 42] Create a new channel

- `getAllChannels(ChannelFactory) -> ChannelIDSeq`  
  [page 42] Return all Id:s for channels, currently alive, created by the target object

- `getEventChannel(ChannelFactory, ChannelID) -> Return`  
  [page 42] Return the channel object associated with the given Id
CosNotifyChannelAdmin_ProxyConsumer

The following functions are exported:

- `get_my_type(ProxyConsumer) -> ProxyType`  
  [page 43] Return the proxy type
- `get_my_admin(ProxyConsumer) -> AdminObject`  
  [page 43] Return the associated Admin object
- `obtain_subscription_types(ProxyConsumer, ObtainInfoMode) -> EventTypeSeq`  
  [page 43] Administer subscription types
- `validate_event_qos(ProxyConsumer, QoSProperties) -> Reply`  
  [page 44] Check if certain Quality of Service properties can be added to events in the current context of the target object

CosNotifyChannelAdmin_ProxyPullConsumer

The following functions are exported:

- `connect_any_pull_supplier(ProxyPullConsumer, PullSupplier) -> Reply`  
  [page 45] Connect a supplier to the proxy
- `suspend_connection(ProxyPullConsumer) -> Reply`  
  [page 45] Suspend the connection between the client and the proxy
- `resume_connection(ProxyPullConsumer) -> Reply`  
  [page 45] Resume a previously suspended connection with the proxy
- `disconnect_pull_supplier(ProxyPullSupplier) -> ok`  
  [page 48] Close the connection and terminate the proxy

CosNotifyChannelAdmin_ProxyPullSupplier

The following functions are exported:

- `connect_any_pull_consumer(ProxyPullSupplier, PullConsumer) -> Reply`  
  [page 47] Connect a consumer to the proxy
- `pull(ProxyPullSupplier) -> Reply`  
  [page 47] Pull an Any event from the proxy
- `try_pull(ProxyPullSupplier) -> Reply`  
  [page 47] Try and pull an Any event from the proxy
- `disconnect_pull_supplier(ProxyPullSupplier) -> ok`  
  [page 48] Close the connection and terminate the proxy

CosNotifyChannelAdmin_ProxyPushConsumer

The following functions are exported:

- `connect_any_push_supplier(ProxyPushConsumer, PushSupplier) -> Reply`  
  [page 49] Connect a supplier to the proxy
- `push(ProxyPushConsumer, Event) -> Reply`  
  [page 49] Push an Any event to the proxy
- `disconnect_push_consumer(ProxyPushConsumer) -> ok`  
  [page 49] Close the connection and terminate the proxy
**CosNotifyChannelAdmin_ProxyPushSupplier**

The following functions are exported:

- `connect_{any_push_consumer}(ProxyPushSupplier, PushConsumer) -> Reply` [page 50] Connect a consumer to the proxy
- `suspend_connection(ProxyPushSupplier) -> Reply` [page 50] Suspend the connection between the proxy and the client
- `resume_connection(ProxyPushSupplier) -> Reply` [page 50] Resume a previously suspended connection with the proxy
- `disconnect_push_supplier(ProxyPushSupplier) -> ok` [page 51] Close the connection and terminate the proxy

**CosNotifyChannelAdmin_ProxySupplier**

The following functions are exported:

- `_get_{MyType}(ProxySupplier) -> ProxyType` [page 52] Return the proxy type
- `_get_{MyAdmin}(ProxySupplier) -> AdminObject` [page 52] Return the target object's associated Admin object
- `_get_priority_filter(ProxySupplier) -> MappingFilter` [page 52] Return the target object's associated priority MappingFilter
- `_set_priority_filter(ProxySupplier, MappingFilter) -> ok` [page 52] Set the target object's associated priority MappingFilter
- `_get_lifetime_filter(ProxySupplier) -> MappingFilter` [page 53] Return the target object's associated lifetime MappingFilter
- `_set_lifetime_filter(ProxySupplier, MappingFilter) -> ok` [page 53] Set the target object's associated lifetime MappingFilter
- `obtain_offered_types(ProxySupplier, ObtainInfoMode) -> EventTypeSeq` [page 53] Administer the type of events the proxy supplies
- `validate_event_qos(ProxySupplier, QoSProperties) -> Reply` [page 53] Check if the QoS properties can be set

**CosNotifyChannelAdmin_SequenceProxyPullConsumer**

The following functions are exported:

- `connect_sequence_pull_supplier(SequenceProxyPullConsumer, PullSupplier) -> Reply` [page 55] Connect a supplier to the proxy
- `suspend_connection(SequenceProxyPullConsumer) -> Reply` [page 55] Suspend the connection with the proxy
- `resume_connection(SequenceProxyPullConsumer) -> Reply` [page 55] Resume a previously suspended connection with the proxy
- `disconnect_sequence_pull_consumer(SequenceProxyPullConsumer) -> ok` [page 56] Close connection and terminate the proxy
CosNotifyChannelAdmin.SequenceProxyPullSupplier

The following functions are exported:

- `connect_sequence_pullconsumer(SequenceProxyPullSupplier, PullConsumer) -> Reply`  
  [page 57] Connect a consumer to the proxy
- `pull_structured_events(SequenceProxyPullSupplier, MaxEvents) -> Reply`  
  [page 57] Pull structured events from the proxy
- `try_pull_structured_events(SequenceProxyPullSupplier, MaxEvents) -> Reply`  
  [page 58] Try to pull structured events from the proxy
- `disconnect_sequence_pull_supplier(SequenceProxyPullSupplier) -> ok`  
  [page 58] Close the connection and terminate the proxy

CosNotifyChannelAdmin.SupplierAdmin

The following functions are exported:

- `get_MyID(SupplierAdmin) -> AdminID`  
  [page 59] Return the objects Id
- `get_MyChannel(SupplierAdmin) -> Channel`  
  [page 59] Return the objects associated channel
- `get_MyOperator(SupplierAdmin) -> OpType`  
  [page 59] Return the filter scheme
- `get_pull_consumers(SupplierAdmin) -> ProxyIDSeq`  
  [page 59] Return all associated pull consumers Id:s
- `get_push_consumers(SupplierAdmin) -> ProxyIDSeq`  
  [page 60] Return all associated push consumers Id:s
- `get_proxy_consumer(SupplierAdmin, ProxyID) -> Reply`  
  [page 60] Return the Proxy which corresponds to the given Id
- `obtain_notification_pull_consumer(SupplierAdmin, SupplierType) -> Reply`  
  [page 60] Create a new proxy
- `obtain_pull_consumer(SupplierAdmin) -> Proxy`  
  [page 60] Create a new proxy
- `obtain_notification_push_consumer(SupplierAdmin, SupplierType) -> Reply`  
  [page 60] Create a new proxy
- `obtain_push_consumer(SupplierAdmin) -> Proxy`  
  [page 61] Create a new proxy
- `destroy(SupplierAdmin) -> ok`  
  [page 61] Terminate the target object
CosNotifyChannelAdmin_SequenceProxyPushConsumer

The following functions are exported:

- `connect_sequence_push_supplier(SequenceProxyPushConsumer, PushSupplier) -> Reply`
  [page 62] Connect a supplier to the proxy
- `push_structured_events(SequenceProxyPushConsumer, EventBatch) -> Reply`
  [page 62] Push a structured event to the proxy
- `disconnect_sequence_push_consumer(SequenceProxyPushConsumer) -> ok`
  [page 63] Close connection and terminate the proxy

CosNotifyChannelAdmin_SequenceProxyPushSupplier

The following functions are exported:

- `connect_sequence_push_consumer(SequenceProxyPushSupplier, PushConsumer) -> Reply`
  [page 64] Connect a consumer to the proxy
- `suspend_connection(SequenceProxyPushSupplier) -> Reply`
  [page 64] Suspend the connection between the client and the target object
- `resume_connection(SequenceProxyPushSupplier) -> Reply`
  [page 64] Resume a previously suspended connection with the proxy
- `disconnect_sequence_push_supplier(SequenceProxyPushSupplier) -> ok`
  [page 65] Close the connection and terminate the proxy

CosNotifyChannelAdmin_StructuredProxyPullConsumer

The following functions are exported:

- `connect_structured_pull_supplier(StructuredProxyPullConsumer, PullSupplier) -> Reply`
  [page 66] Connect a supplier to the proxy
- `suspend_connection(StructuredProxyPullConsumer) -> Reply`
  [page 66] Suspend the connection between the target object and its client
- `resume_connection(StructuredProxyPullConsumer) -> Reply`
  [page 66] Resume a previously suspended connection with the proxy
- `disconnect_structured_pull_consumer(StructuredProxyPullConsumer) -> ok`
  [page 67] Close the connection and terminate the proxy
CosNotifyChannelAdmin_StructuredProxyPullSupplier

The following functions are exported:

- connect_structured_pull_consumer(StructuredProxyPullSupplier, PullConsumer) -> Reply
  [page 68] Connect a consumer to the proxy
- pull_structured_event(StructuredProxyPullSupplier) -> Reply
  [page 68] Pull a structured event from the proxy
- try_pull_structured_event(StructuredProxyPullSupplier) -> Reply
  [page 69] Try to pull a structured event from the proxy
- disconnect_structured_pull_supplier(StructuredProxyPullSupplier) -> 
  ok
  [page 69] Close connection and terminate the proxy

CosNotifyChannelAdmin_StructuredProxyPushConsumer

The following functions are exported:

- connect_structured_push_supplier(StructuredProxyPushConsumer, PushSupplier) -> Reply
  [page 70] Connect a supplier to the proxy
- push_structured_event(StructuredProxyPushConsumer, StructuredEvent) 
  -> Reply
  [page 70] Push a structured event to the proxy
- disconnect_structured_push_consumer(StructuredProxyPushConsumer) ->
  ok
  [page 71] Close the connection and terminate the proxy

CosNotifyChannelAdmin_StructuredProxyPushSupplier

The following functions are exported:

- connect_structured_push_consumer(StructuredProxyPushSupplier, PushConsumer) -> Reply
  [page 72] Connect a consumer to the proxy
- suspend_connection(StructuredProxyPushSupplier) -> Reply
  [page 72] Suspend the connection with the target object
- resume_connection(StructuredProxyPushSupplier) -> Reply
  [page 72] Resume a previously suspended connection
- disconnect_structured_push_supplier(StructuredProxyPushSupplier) ->
  ok
  [page 73] Close the connection and terminate the target object

CosNotifyComm_NotifyPublish

The following functions are exported:

- offer_change(Object, Added, Removed) -> Reply
  [page 74] Inform the target object which type of events the supplier will deliver
CosNotifyComm.NotifySubscribe

The following functions are exported:

- `subscription_change(Object, Added, Removed) -> Reply
  [page 75] Inform the target object which event types the client will and will not accept in the future`

CosNotifyFilter.Filter

The following functions are exported:

- `get_constraint_grammar(Filter) -> Grammar
  [page 76] Return which type of Grammar the Filter uses`
- `add_constraints(Filter, ConstraintExpSeq) -> Reply
  [page 76] Add new constraints to the filter`
- `modify_constraints(Filter, ConstraintIDSeq, ConstraintInfoSeq) -> Reply
  [page 76] Modify existing constraints`
- `get_constraints(Filter, ConstraintIDSeq) -> Reply
  [page 77] Return all constraints which match the supplied Ids`
- `get_all_constraints(Filter) -> ConstraintInfoSeq
  [page 77] Return all constraints associated with the target object`
- `remove_all_constraints(Filter) -> ok
  [page 77] Remove all constraints associated with the target object`
- `destroy(Filter) -> ok
  [page 78] Terminate the target object`
- `match(Filter, Event) -> Reply
  [page 78] Match the Any event if it satisfies at least one constraint`
- `match_structured(Filter, Event) -> Reply
  [page 78] Match the structured event if it satisfies at least one constraint`
- `attach_callback(Filter, NotifySubscribe) -> CallbackID
  [page 78] Connect NotifySubscribe object, which should be informed when the target object’s constraints are updated`
- `detach_callback(Filter, CallbackID) -> Reply
  [page 78] Disconnect the NotifySubscribe object with the given Id`
- `get_callbacks(Filter) -> CallbackIDSeq
  [page 79] Return all NotifySubscribe Id’s associated with the target object`

CosNotifyFilter.FilterAdmin

The following functions are exported:

- `add_filter(Object, Filter) -> FilterID
  [page 80] Add a new filter to the target object`
- `remove_filter(Object, FilterID) -> ok
  [page 80] Remove a filter associated with the target object`
- `get_filter(Object, FilterID) -> Reply
  [page 80] Return the filter with the given Id`
- `get_all_filters(Object) -> FilterIDSeq`
  [page 80] Return a list of all filter IDs associated with the target object
- `remove_all_filters(Object) -> ok`
  [page 81] Remove all filters from the target object

**CosNotifyFilter.Factory**

The following functions are exported:

- `create_filter(FilterFactory, Grammar) -> Reply`
  [page 82] Create a Filter object
- `create_mapping_filter(FilterFactory, Grammar) -> Reply`
  [page 82] Create a MappingFilter object

**CosNotifyFilter.MappingFilter**

The following functions are exported:

- `_get_constraint_grammar(MappingFilter) -> Grammar`
  [page 83] Return which type of Grammar the MappingFilter uses
- `_get_value_type(MappingFilter) -> CORBA::TypeCode`
  [page 83] Return the CORBA::TypeCode of the default value associated with the target object
- `_get_default_value(MappingFilter) -> #any`
  [page 83] Return the #any() default value associated with the target object
- `add_mapping_constraints(MappingFilter, MappingConstraintPairSeq) -> Reply`
  [page 83] Add new mapping constraints
- `modify_constraints(MappingFilter, ConstraintIDSeq, MappingConstraintInfoSeq) -> Reply`
  [page 84] Modify the constraints associated with the target object
- `get_mapping_constraints(MappingFilter, ConstraintIDSeq) -> Reply`
  [page 85] Return the target object’s associated constraints with given ID:s
- `get_all_mapping_constraints(MappingFilter) -> MappingConstraintInfoSeq`
  [page 85] Return the target object’s all associated constraints
- `remove_all_mapping_constraints(MappingFilter) -> ok`
  [page 85] Remove all constraints associated with the target object
- `destroy(MappingFilter) -> ok`
  [page 86] Terminate the target object
- `match(MappingFilter, Event) -> Reply`
  [page 86] Evaluate the given Any event with the Filter’s constraints
- `match_structured(MappingFilter, Event) -> Reply`
  [page 86] Evaluate the given structured event with the Filter’s constraints
The following functions are exported:

- **install()** -> Return
  [page 87] Install the cosNotification application

- **install(Seconds)** -> Return
  [page 87] Install the cosNotification application

- **install_event()** -> Return
  [page 87] Install the necessary cosEvent interfaces

- **install_event(Seconds)** -> Return
  [page 87] Install the necessary cosEvent interfaces

- **uninstall()** -> Return
  [page 87] Uninstall the cosNotification application

- **uninstall(Seconds)** -> Return
  [page 88] Uninstall the cosNotification application

- **uninstall_event()** -> Return
  [page 88] Uninstall the inherited cosEvent interfaces

- **uninstall_event(Seconds)** -> Return
  [page 88] Uninstall the inherited cosEvent interfaces

- **start()** -> Return
  [page 88] Start the cosNotification application

- **stop()** -> Return
  [page 88] Stop the cosNotification application

- **start_global_factory()** -> ChannelFactory
  [page 88] Start a global channel factory as default

- **start_global_factory(Options)** -> ChannelFactory
  [page 88] Start a global channel factory with options

- **start_factory()** -> ChannelFactory
  [page 89] Start a channel factory as default

- **start_factory(Options)** -> ChannelFactory
  [page 89] Start a channel factory with options

- **stop_factory(ChannelFactory)** -> Reply
  [page 89] Terminate the target object

- **start_filter_factory()** -> FilterFactory
  [page 89] Start a filter factory

- **stop_filter_factory(FilterFactory)** -> Reply
  [page 90] Terminate the target object
CosNotification -
AdminPropertiesAdmin

Erlang Module

To get access to the record definitions for the structures use:
-include_lib("cosNotification/include/*.hrl").
All objects, which inherit this interface, export functions described in this module.

Exports

get_admin(Object) -> AdminProperties
Types:
- Object = #objref
- AdminProperties = [AdminProperty]
- AdminProperty = #'CosNotification.Property'(name, value)
- name = string()
- value = #any
This operation returns sequence of name-value pairs which encapsulates the current administrative properties of the target object.

set_admin(Object, AdminProperties) -> Reply
Types:
- Object = #objref
- AdminProperties = [AdminProperty]
- AdminProperty = #'CosNotification.Property'(name, value)
- name = string()
- value = #any
- Reply = ok | {EXCEPTION, CosNotification.UnsupportedAdmin}
As input, this operation accepts a sequence of name-value pairs encapsulating the desired administrative settings for the target object. If it is not possible to set the given properties the exception UnsupportedAdmin will be raised.
CosNotification_QoSAadmin

Erlang Module

To get access to the record definitions for the structures use:
-include_lib("cosNotification/include/*.hrl").
All objects, which inherit this interface, export functions described in this module.

Exports

get_qos(Object) -> Reply
Types:
- Object = #objref
- Reply = [QoSProperty]
- QoSProperty = #CosNotification_Property{name, value}
- name = string()
- value = #any
This operation returns a list of name-value pairs which encapsulates the current QoS settings for the target object.

set_qos(Object, QoS) -> Reply
Types:
- Object = #objref
- QoS = [QoSProperty]
- QoSProperty = #CosNotification_Property{name, value}
- name = string()
- value = #any
- Reply = ok |
  {EXCEPTION, #CosNotification_UnsupportedQoS{qos_err}}
- qos_err = PropertyErrorSeq
  PropertyErrorSeq = [PropertyError]
- PropertyError = #CosNotification_PropertyError{name, available_range}
- code = 'UNSUPPORTED_PROPERTY' | 'UNAVAILABLE_PROPERTY' | 'UNSUPPORTED_VALUE' | 'UNAVAILABLE_VALUE' | 'BAD_PROPERTY' | 'BAD_TYPE' | 'BAD_VALUE'
- name = string()
- available_range = PropertyRange
  PropertyRange = #CosNotification_PropertyRange{low_val, high_val}
- low_val = high_val = #any
To alter the current QoS settings for the target object this function must be used. If it is not possible to set the requested QoS the UnsupportedQoS exception is raised, which includes a sequence of PropertyError’s describing which QoS, possible range and why is not allowed.

validate_qos(Object, QoS) -> Reply

Types:
- Object = #objref
- QoS = [QoSProperty]
- QoSProperty = #'Property'{name, value}
- name = string()
- value = #any
- Reply = {ok, NamedPropertyRangeSeq} | {'EXCEPTION', CosNotification_UnsupportedQoS{}}
- NamedPropertyRangeSeq = [NamedPropertyRange]
- NamedPropertyRange = #CosNotification_NamedPropertyRange{name, range}
- name = string()
- range = #CosNotification_PropertyRange{low_val, high_val}
- low_val = #any
- high_val = #any

The purpose of this operations is to check if a QoS setting is supported by the target object and if so, the operation returns additional properties which could be optionally added as well.
CosNotifyChannelAdmin - ConsumerAdmin

Erlang Module

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

This module also exports the functions described in:
- CosNotification_QoSAdmin [page 34]
- CosNotifyComm_NotifySubscribe [page 75]
- CosNotifyFilter_FilterAdmin [page 80]

Exports

-get_MyID(ConsumerAdmin) -> AdminID
Types:
- ConsumerAdmin = #objref
- AdminID = long()
The ID returned by the creating channel is equal to the value encapsulated by this readonly attribute.

-get_MyChannel(ConsumerAdmin) -> Channel
Types:
- ConsumerAdmin = #objref
- Channel = #objref
The creating channel’s reference is maintained by this readonly attribute.

-get_MyOperator(ConsumerAdmin) -> OpType
Types:
- ConsumerAdmin = #objref
- OpType = 'AND.OP' | 'OR.OP'
When ConsumerAdmin’s are created an operation type, i.e., 'AND.OP' or 'OR.OP', is supplied, which determines the semantics used by the target object concerning evaluation against any associated Filter objects.

-get_priority_filter(ConsumerAdmin) -> MappingFilter
Types:
- ConsumerAdmin = MappingFilter = #objref

If set, this operation returns the associated priority MappingFilter, otherwise a NIL object reference is returned.

_set_priority_filter(ConsumerAdmin, MappingFilter) -> ok
Types:
- ConsumerAdmin = MappingFilter = #objref

To associate a priority MappingFilter with the target object this operation must be used.

_get_lifetime_filter(ConsumerAdmin) -> MappingFilter
Types:
- ConsumerAdmin = MappingFilter = #objref

Unless a lifetime MappingFilter have been associated with the target object a NIL object reference is returned by this operation.

_set_lifetime_filter(ConsumerAdmin, MappingFilter) -> ok
Types:
- ConsumerAdmin = MappingFilter = #objref

This operation associate a lifetime MappingFilter with the target object.

_get_pull_suppliers(ConsumerAdmin) -> ProxyIDSeq
Types:
- ConsumerAdmin = #objref
- ProxyIDSeq = [ProxyID]
- ProxyID = long()

This readonly attribute maintains the Id’s for all PullProxies created by the target object and still alive.

_get_push_suppliers(ConsumerAdmin) -> ProxyIDSeq
Types:
- ConsumerAdmin = #objref
- ProxyIDSeq = [ProxyID]
- ProxyID = long()

This attribute is similar to the get_pull_suppliers attribute but maintains the Id’s for all PushProxies created by the target object and still alive.

get_proxy_supplier(ConsumerAdmin, ProxyID) -> Reply
Types:
- ConsumerAdmin = #objref
- ProxyID = long()
- Reply = Proxy | {'EXCEPTION', #CosNotifyChannelAdminProxyNotFound'}
If a proxy with the given Id exists the reference to the object is returned, but if the
object have terminated, or an incorrect Id is supplied, an exception is raised.

obtain_notification_pull_supplier(ConsumerAdmin, ConsumerType) -> Reply
Types:
  * ConsumerAdmin = #objref
  * ConsumerType = ‘ANY_EVENT’, ‘STRUCTURED_EVENT’, ‘SEQUENCE_EVENT’
  * Reply = {Proxy, ProxyID}
  * Proxy = #objref
  * ProxyID = long()

Determined by the parameter ConsumerType, a proxy which will accept events of the
defined type is created. Along with the object reference an Id is returned.

obtain_pull_supplier(ConsumerAdmin) -> Proxy
Types:
  * ConsumerAdmin = #objref
  * Proxy = #objref
This operation creates a new proxy which accepts #any[] events.

obtain_notification_push_supplier(ConsumerAdmin, ConsumerType) -> Reply
Types:
  * ConsumerAdmin = #objref
  * ConsumerType = ‘ANY_EVENT’, ‘STRUCTURED_EVENT’, ‘SEQUENCE_EVENT’
  * Reply = {Proxy, ProxyID}
  * Proxy = #objref
  * ProxyID = long()

A proxy which accepts events of the type described by the parameter ConsumerType is
created by this operation. A unique Id is returned as an out parameter.

obtain_push_supplier(ConsumerAdmin) -> Proxy
Types:
  * ConsumerAdmin = #objref
  * Proxy = #objref
The object created by this function is a proxy which accepts #any[] events.

destroy(ConsumerAdmin) -> ok
Types:
  * ConsumerAdmin = #objref
To terminate the target object this operation should be used. The associated Channel
will be notified.
CosNotifyChannelAdmin - EventChannel

Erlang Module

To get access to the record definitions for the structures use:
   -include_lib("cosNotification/include/*.hrl").
This module also exports the functions described in:

   - CosNotification_QoSAdmin [page 34]
   - CosNotification_AdminPropertiesAdmin [page 33]

Exports

$get_MyFactory(Channel) -> ChannelFactory
Types:
   - Channel = #objref
   - ChannelFactory = #objref
This readonly attribute maintains the reference of the event channel factory that
created the target channel.

$get_default_consumer_admin(Channel) -> ConsumerAdmin
Types:
   - Channel = #objref
   - ConsumerAdmin = #objref
This is a readonly attribute which maintains a reference to a default ConsumerAdmin
object associated with the target object.

$get_default_supplier_admin(Channel) -> SupplierAdmin
Types:
   - Channel = #objref
   - SupplierAdmin = #objref
This is a readonly attribute which maintains a reference to a default SupplierAdmin
object associated with the target object.

$get_default_filter_factory(Channel) -> FilterFactory
Types:
The default FilterFactory associated with the target channel is maintained by this readonly attribute.

```java
new_for_consumers(Channel, OpType) -> Return
```

Types:
- Channel = #objref
- OpType = 'AND\_OP' | 'OR\_OP'
- Return = {ConsumerAdmin, AdminID}
- ConsumerAdmin = #objref
- AdminID = long()

This operation creates a new instance of a ConsumerAdmin and supplies an Id which may be used when invoking other operations exported by this module. The returned object will inherit the Quality of Service properties of the target channel.

```java
for_consumers(Channel) -> ConsumerAdmin
```

Types:
- Channel = #objref
- ConsumerAdmin = #objref

A new new instance of a ConsumerAdmin object is created but no Id is returned. The returned object's operation type, i.e., 'AND\_OP' or 'OR\_OP', will be set to the value of the configuration parameter $\langle c\rangle$filterOp. The target object's Quality of Service properties will be inherited by the returned ConsumerAdmin.

```java
new_for_suppliers(Channel, OpType) -> Return
```

Types:
- Channel = #objref
- OpType = 'AND\_OP' | 'OR\_OP'
- Return = {SupplierAdmin, AdminID}
- SupplierAdmin = #objref
- AdminID = long()

Enables us to create a new instance of a SupplierAdmin. An Id, which may be used when invoking other operations exported by this module, is also returned. The current Quality of Service settings associated with the target object will be inherited by the SupplierAdmin.

```java
for_suppliers(Channel) -> SupplierAdmin
```

Types:
- Channel = #objref
- SupplierAdmin = #objref

To create a new SupplierAdmin with the target object's current Quality of Service settings we can use this function. The returned object's operation type ('AND\_OP' or 'OR\_OP') will be determined by the configuration variable $\langle c\rangle$filterOp.
get_consumeradmin(\texttt{Channel}, \texttt{AdminID}) \rightarrow \texttt{ConsumerAdmin}

Types:
- \texttt{Channel} = \#objref
- \texttt{AdminID} = \texttt{long()}
- \texttt{ConsumerAdmin} = \#objref | \{'EXCEPTION',
  \#CosNotifyChannelAdmin.AdminNotFound\}'

If the given \texttt{Id} is associated with a ConsumerAdmin the object reference is returned. If such association never existed or the ConsumerAdmin have terminated an exception is raised.

get_supplieradmin(\texttt{Channel}, \texttt{AdminID}) \rightarrow \texttt{SupplierAdmin}

Types:
- \texttt{Channel} = \#objref
- \texttt{AdminID} = \texttt{long()}
- \texttt{SupplierAdmin} = \#objref | \{'EXCEPTION',
  \#CosNotifyChannelAdmin_AdminNotFound\}'

Equal to the operation \texttt{get_consumeradmin}/2 but a reference to a SupplierAdmin is returned.

get_all_consumeradmins(\texttt{Channel}) \rightarrow \texttt{Reply}

Types:
- \texttt{Channel} = \#objref
- \texttt{Reply} = [\texttt{AdminID}]
- \texttt{AdminID} = \texttt{long()}

To get access to all ConsumerAdmin Id’s created by the target object, and still alive, this operation could be invoked.

get_all_supplieradmins(\texttt{Channel}) \rightarrow \texttt{Reply}

Types:
- \texttt{Channel} = \#objref
- \texttt{Reply} = [\texttt{AdminID}]
- \texttt{AdminID} = \texttt{long()}

Equal to the operation \texttt{get_all_consumeradmins}/1 but returns a list of all SupplierAdmin object ID’s.

\texttt{destroy}(\texttt{Channel}) \rightarrow \texttt{ok}

Types:
- \texttt{Channel} = \#objref

The \texttt{destroy} operation will terminate the target channel and all associated Admin objects.
CosNotifyChannelAdmin - EventChannelFactory

Erlang Module

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

Exports

create_channel(ChannelFactory, InitialQoS, InitialAdmin) -> Return
Types:
- ChannelFactory = #objref
- InitialQoS = CosNotification::QoSProperties
- InitialAdmin = CosNotification::AdminProperties
- Return = {EventChannel, ChannelID }
  - EventChannel = #objref
  - ChannelID = long()
This operation creates a new event channel. Along with the channel reference an id is
returned which can be used when invoking other operations exported by this module.
The Quality of Service argument supplied will be inherited by objects created by the
channel. For more information about QoS settings see the User’s Guide.

get_all_channels(ChannelFactory) -> ChannelIDSeq
Types:
- ChannelFactory = #objref
- ChannelIDSeq = [long()]
This operation returns a id sequence of all channel's created by this ChannelFactory.

get_event_channel(ChannelFactory, ChannelID) -> Return
Types:
- ChannelFactory = #objref
- ChannelID = long()
- Return = EventChannel | {'EXCEPTION',
  '#CosNotifyChannelAdmin.ChannelNotFound'}
  - EventChannel = #objref
This operation returns the EventChannel associated with the given id. If no channel is
associated with the id, i.e., never existed or have been terminated, an exception is raised.
CosNotifyChannelAdmin - ProxyConsumer

Erlang Module

To get access to the record definitions for the structures use:
-include_lib("cosNotification/include/*.hrl").
This module also exports the functions described in:

- CosNotification_QoSAdmin [page 34]
- CosNotifyFilter_FilterAdmin [page 80]

Exports

:get_MyType(ProxyConsumer) -> ProxyType
Types:
- ProxyConsumer = #objref
- ProxyType = 'PUSH_ANY' | 'PULL_ANY' | 'PUSH_STRUCTURED' | 'PULL_STRUCTURED' | 'PUSH_SEQUENCE' | 'PULL_SEQUENCE'
This readonly attribute maintains the enumerant describing the which type the target object is.

:get_MyAdmin(ProxyConsumer) -> AdminObject
Types:
- ProxyConsumer = AdminObject = #objref
This readonly attribute maintains the admin's reference which created the target object.

obtain_subscription_types(ProxyConsumer, ObtainInfoMode) -> EventTypeSeq
Types:
- ProxyConsumer = #objref
- ObtainInfoMode = 'ALL_NOW_UPDATES_OFF' | 'ALL_NOW_UPDATES_ON' | 'NONE_NOW_UPDATES_OFF' | 'NONE_NOW_UPDATES_ON'
- EventTypeSeq = [EventType]
- EventType = '#CosNotification.EventType'(domain_name, type_name)
- domain_name = type_name = string()
Depending on the input parameter ObtainInfoMode, this operation may return a sequence of the EventTypes the target object is interested in receiving. If 'ALL_NOW_UPDATES_OFF' or 'ALL_NOW_UPDATES_ON' is given a sequence will be returned, otherwise not. If 'ALL_NOW_UPDATES_OFF' or 'NONE_NOW_UPDATES_OFF' are issued the target object will not inform the associated NotifySubscribe object when an update occurs. 'ALL_NOW_UPDATES_ON' or 'NONE_NOW_UPDATES_ON' will result in that update information will be sent.

validate_event_qos(ProxyConsumer, QoSProperties) -> Reply

Types:
- ProxyConsumer = #objref
- QoSProperties = [QoSProperty]
- QoSProperty = '#CosNotification_Property'(name, value)
  - name = string()
  - value = #any
- Reply = {ok, NamedPropertyRangeSeq} | {'EXCEPTION',
  CosNotification_UnsupportedQoS{qos_err}}
  - NamedPropertyRangeSeq = [NamedPropertyRange]
  - NamedPropertyRange = '#CosNotification_NamedPropertyRange{name, range}
    - name = string()
    - range = '#CosNotification_PropertyRange{low_val, high_val}
    - low_val = #any
    - high_val = #any
- qos_err = PropertyErrorSeq
- PropertyErrorSeq = [PropertyError]
  - PropertyError = '#CosNotification_PropertyError{code, name, available_range}
    - code = 'UNSUPPORTED_PROPERTY' | 'UNAVAILABLE_PROPERTY' | 'UNSUPPORTED_VALUE' | 'UNAVAILABLE_VALUE' | 'BAD_PROPERTY' | 'BAD_TYPE' | 'BAD_VALUE'
    - name = string()
    - available_range = PropertyRange
  - PropertyRange = '#CosNotification_PropertyRange{low_val, high_val}
    - low_val = high_val = #any

To check if certain Quality of Service properties can be added to events in the current context of the target object this operation should be used. If we cannot support the required settings an exception describing why will be raised.
CosNotifyChannelAdmin - ProxyPullConsumer

Erlang Module

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

This module also exports the functions described in:
- CosNotifyComm_NotifyPublish [page 74]
- CosNotification_QoSAdmin [page 34]
- CosNotifyFilter_FilterAdmin [page 80]
- CosNotifyChannelAdmin_ProxyConsumer [page 43]

Exports

connect_any_pull_supplier(ProxyPullConsumer, PullSupplier) -> Reply

Types:
- ProxyPullConsumer = #objref
- PullSupplier = #objref
- Reply = ok | {'EXCEPTION', #CosEventChannelAdmin_AlreadyConnected'}
  | {'EXCEPTION', #CosEventChannelAdmin_TypeError'}

This operation connects the given PullSupplier to the target object. If a client is already connected the AlreadyConnected exception will be raised. The client must support the operations pull and try_pull, otherwise the TypeError exception is raised.

suspend_connection(ProxyPullConsumer) -> Reply

Types:
- ProxyPullConsumer = #objref
- Reply = ok | {'EXCEPTION',
  #CosNotifyChannelAdmin_ConnectionAlreadyInactive'}
  | {'EXCEPTION',
  #CosNotifyChannelAdmin_NotConnected'}

If we want to temporarily suspend the connection with the target object this operation must be sued. If the connection already have been suspended or no client have been connected an exception is raised.

resume_connection(ProxyPullConsumer) -> Reply

Types:
ProxyPullConsumer

- ProxyPullConsumer = #objref
- Reply = ok | {'EXCEPTION',
  #CosNotifyChannelAdmin_ConnectionAlreadyActive'} | {'EXCEPTION',
  #CosNotifyChannelAdmin_NotConnected'}

If the connection has been suspended earlier we can invoke this operation to reinstate
the connection. If the connection already is active or no client have been connected to
the target object an exception is raised.

disconnect_pull_consumer(ProxyPullConsumer) -> ok

Types:
- ProxyPullConsumer = #objref

Invoking this operation disconnects the client from the target object which then
terminates and inform its administrative parent.
CosNotifyChannelAdmin - ProxyPullSupplier

Erlang Module

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

This module also exports the functions described in:

- CosNotifyComm_NotifySubscribe [page 75]
- CosNotification_QoSAdmin [page 34]
- CosNotifyFilter_FilterAdmin [page 80]
- CosNotifyChannelAdmin_ProxySupplier [page 52]

Exports

connect_any_pull_consumer(ProxyPullSupplier, PullConsumer) -> Reply

Types:
- ProxyPullSupplier = #objref
- PullConsumer = #objref
- Reply = ok | {'EXCEPTION', '#CosEventChannelAdmin_AreadyConnected'[]}

This operation connects the given PullConsumer to the target object. If a connection already exists the AlreadyConnected exception is raised.

pull(ProxyPullSupplier) -> Reply

Types:
- ProxyPullSupplier = #objref
- Reply = #any | {'EXCEPTION', #CosEventChannelAdmin_Disconnected'[]}

This operation pulls next #any[] event, and blocks if the target object have no events to forward, until an event can be delivered. If no client have been connected the Disconnected exception is raised.

try_pull(ProxyPullSupplier) -> Reply

Types:
- ProxyPullSupplier = #objref
- Reply = {#any, HasEvent} | {'EXCEPTION', #CosEventChannelAdmin_Disconnected'[]}
- HasEvent = boolean()
This operation pulls next event, but do not block if the target object have no event to forward. If no client have been connected the Disconnected exception is raised.

\texttt{disconnect\_pull\_supplier(ProxyPullSupplier) \rightarrow ok}

Types:
- \texttt{ProxyPullSupplier = #objref}

Invoking this operation will cause the target object to close the connection and terminate.
CosNotifyChannelAdmin - ProxyPushConsumer

Erlang Module

To get access to the record definitions for the structures use:
\[-include_lib("cosNotification/include/*.hrl").\]

This module also exports the functions described in:
- CosNotifyComm_NotifyPublish [page 74]
- CosNotification_QoSAdmin [page 34]
- CosNotifyFilter_FilterAdmin [page 80]
- CosNotifyChannelAdmin_ProxyConsumer [page 43]

Exports

connect_any_push_supplier(ProxyPushConsumer, PushSupplier) -> Reply

Types:
- ProxyPushConsumer = #objref
- PushSupplier = #objref
- Reply = ok | {'EXCEPTION', #CosEventChannelAdmin_AlreadyConnected'}

This operation connects a PushSupplier to the target object. If a connection already exists the AlreadyConnected exception is raised.

push(ProxyPushConsumer, Event) -> Reply

Types:
- ProxyPushConsumer = #objref
- Event = #any
- Reply = ok | {'EXCEPTION', #CosEventChannelAdmin_Disconnected'}

This operation pushes an #any{} event to the target object. If no client have been connected the Disconnected exception is raised.

disconnect_push_consumer(ProxyPushConsumer) -> ok

Types:
- ProxyPushConsumer = #objref

Invoking this operation will cause the target object to close the connection and terminate.
CosNotifyChannelAdmin - ProxyPushSupplier

Erlang Module

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

This module also exports the functions described in:

- CosNotifyCommNotifySubscribe [page 75]
- CosNotification_QoSAdmin [page 34]
- CosNotifyFilterFilterAdmin [page 80]
- CosNotifyChannelAdminProxySupplier [page 52]

Exports

connect_any_push_consumer(ProxyPushSupplier, PushConsumer) -> Reply

Types:
- ProxyPushSupplier = #objref
- PushConsumer = #objref
- Reply = ok | {'EXCEPTION', 'CosEventChannelAdminAlreadyConnected'} | {'EXCEPTION', 'CosEventChannelAdminTypeError'}

This operation connects a PushConsumer to the target object. If a connection already exists or the given client does not support the operation push an exception, AlreadyConnected and TypeError respectively, is raised.

suspend_connection(ProxyPushSupplier) -> Reply

Types:
- ProxyPushSupplier = #objref
- Reply = ok | {'EXCEPTION', #CosNotifyChannelAdminConnectionAlreadyInactive'} | {'EXCEPTION', #CosNotifyChannelAdminNotConnected'}

This operation suspends the connection with the client object. If the connection already is suspended or no client have been associated an exception is raised.

resume_connection(ProxyPushSupplier) -> Reply

Types:
- ProxyPullConsumer = #objref
If a connection have been suspended earlier, calling this operation will resume the connection. If the connection already is active or no client have been connected an exception is raised.

disconnect_push_supplier(ProxyPushSupplier) -> ok

Types:
- ProxyPushSupplier = #objref

This operation cause the target object to close the connection and terminate.
CosNotifyChannelAdmin -
ProxySupplier

Erlang Module

To get access to the record definitions for the structures use:
-include_lib("cosNotification/include/*.hrl").
This module also exports the functions described in:

- CosNotification_QoSAdmin [page 34]
- CosNotifyFilter_FilterAdmin [page 80]

Exports

-get_TYPE(ProxySupplier) -> ProxyType
Types:
- ProxySupplier = #objref
- ProxyType = ‘PUSH_ANY’ | ‘PULL_ANY’ | ‘PUSH_STRUCTURED’ |
  ‘PULL_STRUCTURED’ | ‘PUSH_SEQUENCE’ | ‘PULL_SEQUENCE’
This readonly attribute maintains the enumerator describing the which type the target
object is.

-get_Admin(ProxySupplier) -> AdminObject
Types:
- ProxySupplier = #objref
- AdminObject = #objref
This readonly attribute maintains the admin’s reference which created the target object.

-get_priority_filter(ProxySupplier) -> MappingFilter
Types:
- ProxySupplier = #objref
- MappingFilter = #objref
This operation returns the associated priority MappingFilter. If no such object exist a
NIL reference is returned.

-set_priority_filter(ProxySupplier, MappingFilter) -> ok
Types:
- ProxySupplier = #objref
- MappingFilter = #objref

This operation associate a new priority MappingFilter with the target object.

_get_lifetime_filter(ProxySupplier) \rightarrow MappingFilter

Types:
- ProxySupplier = #objref
- MappingFilter = #objref

This operation returns the associated lifetime MappingFilter. If no such object exist a NIL reference is returned.

_set_lifetime_filter(ProxySupplier, MappingFilter) \rightarrow ok

Types:
- ProxySupplier = #objref
- MappingFilter = #objref

This operation associate a new lifetime MappingFilter with the target object.

obtain_offered_types(ProxySupplier, ObtainInfoMode) \rightarrow EventTypeSeq

Types:
- ProxySupplier = #objref
- ObtainInfoMode = 'ALL_NOW_UPDATES_OFF' | 'ALL_NOW_UPDATES_ON' | 'NONE_NOW_UPDATES_OFF' | 'NONE_NOW_UPDATES_ON'
- EventTypeSeq = [EventType]
- EventType = #CosNotification_EventType{domain_name, type_name}
  - domain_name = type_name = string()

Depending on the input parameter ObtainInfoMode, this operation may return a sequence of the EventTypes the target object is interested in receiving. If 'ALL_NOW_UPDATES_OFF' or 'ALL_NOW_UPDATES_ON' is given a sequence will be returned, otherwise not. If 'ALL_NOW_UPDATES_OFF' or 'NONE_NOW_UPDATES_OFF' are issued the target object will not inform the associated NotifySubscribe object when an update occurs. 'ALL_NOW_UPDATES_ON' or 'NONE_NOW_UPDATES_ON' will result in that update information will be sent.

validate_event_qos(ProxySupplier, QoSProperties) \rightarrow Reply

Types:
- ProxySupplier = #objref
- QoSProperties = [QoSProperty]
- QoSProperty = #CosNotification_Property{name, value}
  - name = string()
  - value = #any
- Reply = {ok, NamedPropertyRangeSeq} | {'EXCEPTION',
  CosNotification_UnsupportedQoS{qos_err}}
- NamedPropertyRangeSeq = [NamedPropertyRange]
- NamedPropertyRange = #CosNotification_NamedPropertyRange{name, range}
  - name = string()
To check if certain Quality of Service properties can be added to events in the current context of the target object this operation should be used. If we cannot support the required settings an exception describing why will be raised.
CosNotifyChannelAdmin - SequenceProxyPullConsumer
Erlang Module

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

This module also exports the functions described in:
- CosNotifyComm_NotifyPublish [page 74]
- CosNotification_QoSAdmin [page 34]
- CosNotifyFilter_FilterAdmin [page 80]
- CosNotifyChannelAdmin_ProxyConsumer [page 43]

Exports

connect_sequence_pull_supplier(SequenceProxyPullConsumer, PullSupplier) -> Reply
Types:
- SequenceProxyPullConsumer = #objref
- PullSupplier = #objref
- Reply = ok | {"EXCEPTION", #CosEventChannelAdmin_AlreadyConnected} | {"EXCEPTION", #CosEventChannelAdmin_TypeError}

This operation connects a PullSupplier to the target object. If a connection already exists or the supplied client does not support the functions pull_structured_events and try_pull_structured_events an exception is raised.

suspend_connection(SequenceProxyPullConsumer) -> Reply
Types:
- SequenceProxyPullConsumer = #objref
- Reply = ok | {"EXCEPTION", #CosNotifyChannelAdmin_ConnectionAlreadyInactive} | {"EXCEPTION", #CosNotifyChannelAdmin_NotConnected}

If a connection exist, invoking this operation will suspend the connection until instructed otherwise. Otherwise, no client have been connected or this operation already have been invoked an exception is raised.

resume_connection(SequenceProxyPullConsumer) -> Reply
Types:
- `SequenceProxyPullConsumer = #objref`
- `Reply = ok \| \{ 'EX CEP T I O N',
  #CosNotifyChannelAdmin_ConnectionAlreadyInactive'\} \| \{ 'EX CEP T I O N',
  #CosNotifyChannelAdmin_NotConnected'\}\`

If an connection have been suspended this operation must be used to resume the connection. If the connection already is active or no client have been connected an exception is raised.

```
disconnect_sequence_pull_consumer(SequenceProxyPullConsumer) -> ok
```

Types:
- `SequenceProxyPullConsumer = #objref`

This operation close the connection to the client and terminates the target object.
CosNotifyChannelAdmin - SequenceProxyPullSupplier

Erlang Module

To get access to the record definitions for the structures use:
\[-include_lib("cosNotification/include/*.hrl").\]
This module also exports the functions described in:

- \[CosNotifyComm\] NotifySubscribe [page 75]
- \[CosNotification\] QoSAdmin [page 34]
- \[CosNotifyFilter\] FilterAdmin [page 80]
- \[CosNotifyChannelAdmin\] ProxySupplier [page 52]

Exports

\[connect\_sequence\_pull\_consumer\(\text{SequenceProxyPullSupplier, PullConsumer}\) \rightarrow \text{Reply}\]

Types:
- SequenceProxyPullSupplier = \#objcRef
- PullConsumer = \#objcRef
- Reply = ok | \{'EXCEPTION', \#CosEventChannelAdmin\_AlreadyConnected'\}

This operation connects a PullConsumer to the target object. If a connection already exists an exception is raised.

\[pull\_structured\_events\(\text{SequenceProxyPullSupplier, MaxEvents}\) \rightarrow \text{Reply}\]

Types:
- SequenceProxyPullSupplier = \#objcRef
- MaxEvents = long()
- Reply = EventBatch | \{'EXCEPTION', \#CosEventChannelAdmin\_Disconnected'\}
- EventBatch = [StructuredEvent]
- StructuredEvent = \#CosNotification\_StructuredEvent'{header, filterable\_data, remainder\_of\_body}
- header = EventHeader
- filterable\_data = [ \#CosNotification\_Property'{name, value} ]
- name = string()
- value = \#any
- remainder\_of\_body = \#any
A client uses this operation to pull the next event sequence of maximum length \( \text{MaxEvents} \). This operation is blocking and will not reply until the requested amount of events can be delivered or the QoS property \( \text{PacingInterval} \) is reached. For more information see the User's Guide.

```java
try_pull_structured_events(SequenceProxyPullSupplier, MaxEvents) -> Reply
```

Types:
- \( \text{SequenceProxyPullSupplier} = \text{#objref} \)
- \( \text{MaxEvents} = \text{long()} \)
- \( \text{Reply} = [\text{EventBatch, HasEvent}] | \{\text{EXCEPTION}, #\text{CosEventChannelAdmin_Disconnected}\} \)
- \( \text{HasEvent} = \text{boolean()} \)
- \( \text{EventBatch} = [\text{StructuredEvent}] \)
- \( \text{StructuredEvent} = #\text{CosNotification_StructuredEvent}\{\text{header, filterable\_data, remainder\_of\_body}\} \)
- \( \text{header} = \text{EventHeader} \)
- \( \text{filterable\_data} = [\text{#CosNotification_Property}\{\text{name, value}\}] \)
- \( \text{name} = \text{string()} \)
- \( \text{value} = \text{#any} \)
- \( \text{remainder\_of\_body} = \text{#any} \)
- \( \text{EventHeader} = #\text{CosNotification_EventsHeader}\{\text{fixed\_header, variable\_header}\} \)
- \( \text{fixed\_header} = \text{FixedEventHeader} \)
- \( \text{variable\_header} = \text{OptionalHeaderFields} \)
- \( \text{FixedEventHeader} = #\text{CosNotification_FixedEventHeader}\{\text{event\_type, event\_name}\} \)
- \( \text{event\_type} = \text{EventType} \)
- \( \text{event\_name} = \text{string()} \)
- \( \text{EventType} = #\text{CosNotification_EventType}\{\text{domain\_name, type\_name}\} \)
- \( \text{domain\_name} = \text{type\_name} = \text{string()} \)
- \( \text{OptionalHeaderFields} = [\text{#CosNotification_Property}\{\text{name, value}\}] \)

This operation pulls an event sequence of the maximum length \( \text{MaxEvents} \), but does not block if the target object has no events to forward. The outparameter, \( \text{HasEvent} \) is true if the sequence contains any events.

```java
disconnect_sequence_pull_supplier(SequenceProxyPullSupplier) -> ok
```

Types:
- \( \text{SequenceProxyPullSupplier} = \text{#objref} \)

This operation causes the target object to close the connection and terminate.
CosNotifyChannelAdmin - SupplierAdmin

Erlang Module

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

This module also exports the functions described in:
- CosNotification_OsAadmin [page 34]
- CosNotifyComm_NotifyPublish [page 75]
- CosNotifyFilter_FilterAdmin [page 80]

Exports

get getMyID(SupplierAdmin) -> AdminID
Types:
- SupplierAdmin = #objref
- AdminID = long()
When a SupplierAdmin object is created it is given a unique Id by the creating channel. This readonly attribute maintains this Id.

get getMyChannel(SupplierAdmin) -> Channel
Types:
- SupplierAdmin = #objref
- Channel = #objref
The creating channel’s reference is maintained by this readonly attribute.

get getMyOperator(SupplierAdmin) -> OpType
Types:
- SupplierAdmin = #objref
- OpType = 'ANDOP' | 'OROP'
The Operation Type, which determines the semantics the target object will use for any associated Filters, is maintained by this readonly attribute.

get pull consumers(SupplierAdmin) -> ProxyIDSeq
Types:
- SupplierAdmin = #objref
- ProxyIDSeq = [ProxyID]
- ProxyID = long()

A sequence of all associated PullProxy Id's is maintained by this readonly attribute.

**_get_push_consumers(SupplierAdmin) -> ProxyIDSeq**

Types:
- SupplierAdmin = #objref
- ProxyIDSeq = [ProxyID]
- ProxyID = long()

This operation returns all PushProxy Id's created by the target object.

**get_proxy_consumer(SupplierAdmin, ProxyID) -> Reply**

Types:
- SupplierAdmin = #objref
- ProxyID = long()
- Reply = Proxy | {'EXCEPTION', '#CosNotifyChannelAdmin.ProxyNotFound'[]}
- Proxy = #objref

The Proxy which corresponds to the given Id is returned by this operation.

**obtain_notification_pull_consumer(SupplierAdmin, SupplierType) -> Reply**

Types:
- SupplierAdmin = #objref
- SupplierType = 'ANY_EVENT' | 'STRUCTURED_EVENT' | 'SEQUENCE_EVENT'
- Reply = {Proxy, ProxyID}
- Proxy = #objref
- ProxyID = long()

This operation creates a new proxy and returns its object reference along with its ID. The SupplierType parameter determines the event type accepted by the proxy.

**obtain_pull_consumer(SupplierAdmin) -> Proxy**

Types:
- SupplierAdmin = #objref
- Proxy = #objref

A proxy which accepts #any[] events is created by this operation.

**obtain_notification_push_consumer(SupplierAdmin, SupplierType) -> Reply**

Types:
- SupplierAdmin = #objref
- SupplierType = 'ANY_EVENT' | 'STRUCTURED_EVENT' | 'SEQUENCE_EVENT'
- Reply = {Proxy, ProxyID}
- Proxy = #objref
- ProxyID = long()
  Determined by the SupplierType parameter a compliant proxy is created and its object
  reference along with its Id is returned by this operation.

obtainPushConsumer(SupplierAdmin) -> Proxy
Types:
- SupplierAdmin = #objref
- Proxy = #objref
  A proxy which accepts #any() events is created by this operation.

destroy(SupplierAdmin) -> ok
Types:
- SupplierAdmin = #objref
  This operation terminates the SupplierAdmin object and notifies the creating channel
  that the target object no longer is active.
CosNotifyChannelAdmin - SequenceProxyPushConsumer

Erlang Module

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

This module also exports the functions described in:

- CosNotifyComm_CosNotifyPublish [page 74]
- CosNotification_QoSAdmin [page 34]
- CosNotifyFilter_FilterAdmin [page 80]
- CosNotifyChannelAdmin_ProxyConsumer [page 43]

Exports

connect_sequence_push_supplier(SequenceProxyPushConsumer, PushSupplier) -> Reply

Types:
- SequenceProxyPushConsumer = #objref
- PushSupplier = #objref
- Reply = ok | {EXCEPTION, #CosEventChannelAdmin_AlreadyConnected}

This operation connects a PushSupplier to the target object. If a connection already exists the AlreadyConnected exception is raised.

push_structured_events(SequenceProxyPushConsumer, EventBatch) -> Reply

Types:
- SequenceProxyPushConsumer = #objref
- EventBatch = [StructuredEvent]
- StructuredEvent = #CosNotification_StructuredEvent\{header, filterable_data, remainder_of_body\}
- header = EventHeader
- filterable_data = [{#CosNotification_Property\{name, value\}]
- name = string()
- value = #any
- remainder_of_body = #any
- EventHeader = #CosNotification_EventHeader\{fixed_header, variable_header\}
- fixed_header = FixedEventHeader
- variable_header = OptionalEventFields
- FixedEventHeader = #CosNotification_FixedEventHeader'(eventType, eventName)
  eventType = EventType
  eventName = string()
- EventType = #CosNotification_EventType'(domainName, typeName)
  domainName = typeName = string()
- OptionalHeaderFields = [#CosNotification_Property'(name, value)]
- Reply = ok | {'EXCEPTION', #CosEventChannelAdmin_Disconnected'}

A client must use this operation when it wishes to push a new sequence of events to the target object. If no connection exists the Disconnected exception is raised.

disconnect_sequence_push_consumer(SequenceProxyPushConsumer) -> ok

Types:
- SequenceProxyPushConsumer = #objref

This operation cause the target object to close the connection and terminate.
CosNotifyChannelAdmin - SequenceProxyPushSupplier

Erlang Module

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

This module also exports the functions described in:
- CosNotifyComm_Connect
- CosNotification_QoSAdmin
- CosNotifyFilter_FilterAdmin
- CosNotifyChannelAdmin_ProxySupplier

Exports

connect_sequence_push_consumer(SequenceProxyPushSupplier, PushConsumer) -> Reply

Types:
- SequenceProxyPushSupplier = #objref
- PushConsumer = #objref
- Reply = ok | {'EXCEPTION', #CosEventChannelAdmin_AlreadyConnected'} |
  {'EXCEPTION', #CosEventChannelAdmin_TypeError'}

This operation connects a PushConsumer to the target object. If a connection already exists or the function push_structured_events is not supported the exceptions AlreadyConnected or TypeError will be raised respectively.

suspend_connection(SequenceProxyPushSupplier) -> Reply

Types:
- SequenceProxyPushSupplier = #objref
- Reply = ok | {'EXCEPTION', #CosNotifyChannelAdmin_ConnectionAlreadyInactive'} |
  {'EXCEPTION', #CosNotifyChannelAdmin_NotConnected'}

This operation suspends the connection between the client and the target object. If no connection exists or the connection is already suspended an exception is raised.

resume_connection(SequenceProxyPushSupplier) -> Reply

Types:
- SequenceProxyPullConsumer = #objref
If the connection have previously been suspended this operation must used if we want to resume the connection. If no object have been connected or the connection already is active an exception is raised.

```
disconnect_sequence_push_supplier(SequenceProxyPushSupplier) -> ok
```

Types:
- `SequenceProxyPushSupplier = #objref`

This operation cause the target object to close the connection and terminate.
CosNotifyChannelAdmin - StructuredProxyPullConsumer

Erlang Module

To get access to the record definitions for the structures use:
-include_lib("cosNotification/include/*.hrl").
This module also exports the functions described in:

- CosNotifyCommNotifyPublish [page 74]
- CosNotificationQoSAdmin [page 34]
- CosNotifyFilterFilterAdmin [page 80]
- CosNotifyChannelAdminProxyConsumer [page 43]

Exports

connect StructuredProxyPullConsumer PullSupplier) -> Reply
Types:
- StructuredProxyPullConsumer = #objref
- PullSupplier = #objref
- Reply = ok | {EXCEPTION, 'CosNotifyChannelAdminAlreadyConnected'} |
  {EXCEPTION, 'CosNotifyChannelAdminTypeError'}

This operation connects a PullSupplier to the target object. If a connection already exists or the given client object does not support the functions pullStructuredEvent and tryPullStructuredEvent an exception is raised.

suspend_connection(StructuredProxyPullConsumer) -> Reply
Types:
- StructuredProxyPullConsumer = #objref
- Reply = ok | {EXCEPTION, 
  'CosNotifyChannelAdminConnectionAlreadyInactive'} | 
  'CosNotifyChannelAdminNotConnected'}

This operation suspends the connection between the target object and its client. If no connection exists or already suspended an exception is raised.

resume_connection(StructuredProxyPullConsumer) -> Reply
Types:
- StructuredProxyPullConsumer = #objref
If the connection have been suspended this operation must be used if we want to resume the connection. If the connection already are active or no connection have been created an exception is raised.

```csharp
disconnect_structured_pull_consumer(StructuredProxyPullConsumer) => ok
```

Types:
- StructuredProxyPullConsumer = #objref

This operation cause the target object to close the connection and terminate.
CosNotifyChannelAdmin - StructuredProxyPullSupplier

Erlang Module

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

This module also exports the functions described in:
- CosNotifyComm NotifySubscribe [page 75]
- CosNotification_QoSAdmin [page 34]
- CosNotifyFilter_FilterAdmin [page 80]
- CosNotifyChannelAdmin_ProxySupplier [page 52]

Exports

connect_structured_pull_consumer(StructuredProxyPullSupplier, PullConsumer) -> Reply

Types:
- StructuredProxyPullSupplier = #objref
- PullConsumer = #objref
- Reply = ok | {'EXCEPTION', 'CosEventChannelAdmin_AlreadyConnected'}

This operation connects a PullConsumer to the target object. If a connection already exists, the AlreadyConnected exception is raised.

pull_structured_event(StructuredProxyPullSupplier) -> Reply

Types:
- StructuredProxyPullSupplier = #objref
- Reply = StructuredEvent | {'EXCEPTION', #CosEventChannelAdmin_Disconnected'}
- StructuredEvent = #CosNotification_StructuredEvent{header, filterable_data, remainder_of_body}
- header = EventHeader
- filterable_data = [
  #CosNotification_Property{name, value}]
- name = string()
- value = #any
- remainder_of_body = #any
- EventHeader = #CosNotification_EventHeader{fixed_header, variable_header}
- fixed_header = FixedEventHeader


- variable_header = OptionalHeaderFields
- FixedEventHeader = #CosNotification_FixedEventHeader\(\{\text{event_type}, \text{event_name}\}\)
- event_type = EventType
- event_name = string()
- EventType = #CosNotification_EventType\(\{\text{domain_name}, \text{type_name}\}\)
- domain_name = type_name = string()
- OptionalHeaderFields = [ #CosNotification_Property\{\text{name}, \text{value}\} ]

This operation pulls next event from the target object; if an event cannot be delivered this function blocks until an event arrives.

\[
\text{try}_\text{pull}_\text{structured}_\text{event}(\text{StructuredProxyPullSupplier}) \rightarrow \text{Reply}
\]

Types:
- StructuredProxyPullSupplier = #objref
- Reply = \{ StructuredEvent, HasEvent \} | \{'EXCEPTION', #CosEventChannelAdmin_Disconnected'\}
- HasEvent = boolean()
- StructuredEvent = #CosNotification_StructuredEvent\(\{\text{header, filterable_data, remainder_of_body}\}\)
  - header = EventHeader
  - filterable_data = [ #CosNotification_Property\{\text{name}, \text{value}\} ]
  - name = string()
  - value = #any
  - remainder_of_body = #any
- EventHeader = #CosNotification_EventHeader\(\{\text{fixed_header, variable_header}\}\)
  - fixed_header = FixedEventHeader
  - variable_header = OptionalHeaderFields
- FixedEventHeader = #CosNotification_FixedEventHeader\(\{\text{event_type}, \text{event_name}\}\)
  - event_type = EventType
  - event_name = string()
- Domain = #CosNotification_Domain\(\{\text{domain_name}, \text{type_name}\}\)
  - domain_name = type_name = string()
- OptionalHeaderFields = [ #CosNotification_Property\{\text{name}, \text{value}\} ]

This operation try to pull next event from the target object. If no event have arrived an empty event is returned and the out parameter HasEvent is set to false. Otherwise, the boolean flag is set to true and an valid event is returned.

\[
\text{disconnect}_\text{structured}_\text{pull}_\text{supplier}(\text{StructuredProxyPullSupplier}) \rightarrow \text{ok}
\]

Types:
- StructuredProxyPullSupplier = #objref

This operation cause the target object to close the connection and terminate.
CosNotifyChannelAdmin - StructuredProxyPushConsumer

Erlang Module

To get access to the record definitions for the structures use:

```
-include_lib("cosNotification/include/*.hrl").
```

This module also exports the functions described in:

- CosNotifyComm.NotifyPublish [page 74]
- CosNotification.QoSAdmin [page 34]
- CosNotifyFilter.FilterAdmin [page 80]
- CosNotifyChannelAdmin.ProxyConsumer [page 43]

Exports

```erlang
connect_structured_push_supplier(StructuredProxyPushConsumer, PushSupplier) -> Reply

Types:
  - StructuredProxyPushConsumer = #objref
  - PushSupplier = #objref
  - Reply = ok | {EXCEPTION, #CosEventChannelAdmin.AlreadyConnected}
```

This operation connects a PushSupplier to the target object. If a connection already exists an exception is raised.

```erlang
push_structured_event(StructuredProxyPushConsumer, StructuredEvent) -> Reply

Types:
  - StructuredProxyPushConsumer = #objref
  - StructuredEvent = #CosNotification.StructuredEvent{header, filterable_data, remainder_of_body}
  - header = EventHeader
  - filterable_data = [{#CosNotification.Property{name, value}}]
  - name = string()
  - value = #any
  - remainder_of_body = #any
  - EventHeader = #CosNotification.EventHeader{fixed_header, variable_header}
  - fixed_header = FixedEventHeader
  - variable_header = OptionalHeaderFields
  - FixedEventHeader = #CosNotification.FixedEventHeader{event_type, event_name}
```

cosNotification Application
- event_type = EventType
- event_name = string()
- EventType = #CosNotification_EventType'{domain_name, type_name}
- domain_name = type_name = string()
- OptionalHeaderFields = [ #CosNotification_Property'{name, value} ]
- Reply = ok | { 'EXCEPTION', #CosEventChannelAdmin_Disconnected'{} }

When a client wants to push a new event to the target object, this operation must be used.

disconnect_structured_push_consumer(StructuredProxyPushConsumer) -> ok

Types:
- StructuredProxyPushConsumer = #objref

This operation causes the target object to close the connection and terminate.
CosNotifyChannelAdmin - StructuredProxyPushSupplier

Erlang Module

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

This module also exports the functions described in:

- CosNotifyComm_NotifySubscribe [page 75]
- CosNotification_QosAdmin [page 34]
- CosNotifyFilter_FilterAdmin [page 80]
- CosNotifyChannelAdmin_ProxySupplier [page 52]

Exports

connect_structured_push_consumer(StructuredProxyPushSupplier, PushConsumer) -> Reply
Types:
- StructuredProxyPushSupplier = #objcRef
- PushConsumer = #objcRef
- Reply = ok | {EXCEPTION, #CosEventChannelAdminAlreadyConnected} | {EXCEPTION, #CosEventChannelAdminTypeError}

This operation connects a PushConsumer to the target object. If a connection already exists or the function push_structured_event is not supported by the client object an exception is raised.

suspend_connection(StructuredProxyPushSupplier) -> Reply
Types:
- StructuredProxyPushSupplier = #objcRef
- Reply = ok | {EXCEPTION, #CosNotifyChannelAdminConnectionAlreadyInactive} | {EXCEPTION, #CosNotifyChannelAdminNotConnected}

This operation suspends the connection with the target object. If no connection exists or the connection already is suspended an exception is raised.

resume_connection(StructuredProxyPushSupplier) -> Reply
Types:
- StructuredProxyPullConsumer = #objcRef

72  cosNotification Application
If the connection with the target object have been suspended this function must be used to resume the connection. If no client have been connected or the connection is active an exception is raised.

\[
\text{disconnect\_structured\_push\_supplier}(\text{StructuredProxyPushSupplier}) \rightarrow \text{ok}
\]

Types:
- \text{StructuredProxyPushSupplier} = \#\text{objref}

This operation cause the target object to close the connection and terminate.
CosNotifyComm_NotifyPublish

Erlang Module

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

All objects, which inherit this interface, export functions described in this module.

Exports

offer_change(Object, Added, Removed) -> Reply

Types:
- Object = #objref
- Added = Removed = EventTypeSeq
- EventTypeSeq = [type]
- Reply = ok | {'EXCEPTION', CosNotifyComm_InvalidEventType{type}}
- type = #CosNotification_EventType{domain_name, type_name}
- domain_name = type_name = string()

Objects supporting this interface can be informed by supplier objects about which type of events that will be delivered in the future. This operation accepts two parameters describing new and old event types respectively. If any of the supplied event type names is syntactically incorrect an exception is raised.
CosNotifyComm_NotifySubscribe

Erlang Module

To get access to the record definitions for the structures use:
-include_lib("cosNotification/include/*.hrl").
All objects, which inherit this interface, export functions described in this module.

Exports

subscription_change(Object, Added, Removed) -\rightarrow Reply

Types:
- Object = #objref
- Added = Removed = EventTypeSeq
- EventTypeSeq = [type]
- Reply = ok \| ['EXCEPTION', CosNotifyComm_INVALIDEventType{type}]
- type = '#CosNotification_EventType'(domain_name, type_name)
- domain_name = type_name = string()

This operation takes as input two sequences of event type names specifying events the client will and will not accept in the future respectively.
CosNotifyFilter_Filter

Erlang Module

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

Exports

_get_constraint_grammar(Filter) -> Grammar

Types:
  * Filter = #objref
  * Grammar = string()

This operation returns which type of Grammar the Filter uses. Currently, only "EXTENDED_TCL" is supported.

add_constraints(Filter, ConstraintExpSeq) -> Reply

Types:
  * Filter = #objref
  * ConstraintExpSeq = [Constraint]
  * ConstraintExp = #CosNotifyFilter_ConstraintExp{event_types, constraint_expr}
  * event_types = #CosNotification_EventTypeSeq{}
  * constraint_expr = string()
  * Reply = ConstraintInfoSeq | ['EXCEPTION', #CosNotifyFilter_InvalidConstraint{constr}]

Initially, Filters do not contain any constraints, hence, all events will be forwarded. The add_constraints/2 operation allow us to add constraints to the target object.

modify_constraints(Filter, ConstraintIDSeq, ConstraintInfoSeq) -> Reply

Types:
  * Filter = #objref
  * ConstraintIDSeq = [ConstraintID]
- ConstraintInfoSeq = [ConstraintInfo]
  - ConstraintInfo = #CosNotifyFilterConstraintInfo\{constraint_expression, constraintId\}
  - constraint_expression = ConstraintExp
  - constraintId = long()
  - Reply = ok | {‘EXCEPTION’, #CosNotifyFilterInvalidConstraint\{constr\}} | {‘EXCEPTION’, #CosNotifyFilterConstraintNotFound\{id\}}

- constr = ConstraintExp
- id = long()
- ConstraintExp = #CosNotifyFilterConstraintExp\{event_types, constraint_expr\}
- event_types = #CosNotificationEventTypeSeq\[]
- constraint_expr = string()

This operation is invoked by a client in order to modify the constraints associated with the target object. The constraints related to the Id's in the parameter sequence ConstraintIDSeq will, if all values are valid, be deleted. The ConstraintInfoSeq parameter contains Id-Expression pairs and a constraint matching one of the unique Id's will, if all input values are correct, be updated. If the parameters contain incorrect data an exception will be raised.

get_constraints(Filter, ConstraintIDSeq) -> Reply

Types:
- Filter = #objref
- ConstraintIDSeq = [ConstraintID]
- ConstraintID = long()
- Reply = ConstraintInfoSeq | {‘EXCEPTION’, #CosNotifyFilterConstraintNotFound\{id\}}
- ConstraintInfoSeq = [ConstraintInfo]
- ConstraintInfo = #CosNotifyFilterConstraintInfo\{constraint_expression, constraintId\}
- constraint_expression = ConstraintExp
- constraintId = id = long()

This operation return a sequence of ConstraintInfo's, related to the given ConstraintID's, associated with the target object.

get_all_constraints(Filter) -> ConstraintInfoSeq

Types:
- Filter = #objref
- ConstraintInfoSeq = [ConstraintInfo]
- ConstraintInfo = #CosNotifyFilterConstraintInfo\{constraint_expression, constraintId\}
- constraint_expression = ConstraintExp
- constraintId = long()

All constraints, and their unique Id, associated with the target object will be returned by this operation.

remove_all_constraints(Filter) -> ok
Types:
- Filter = #objref

All constraints associated with the target object are removed by this operation and, since the target object no longer contain any constraints, true will always be the result of any match operation.

destroy(Filter) -> ok
Types:
- Filter = #objref

This operation terminates the target object.

match(Filter, Event) -> Reply
Types:
- Filter = #objref
- Event = #any
- Reply = boolean() | {'EXCEPTION', #CosNotifyFilter.UnsupportedFilterableData'}

This operation accepts an #any event and returns true if it satisfies at least one constraint. If the event contains data of the wrong type, e.g., should be a string() but in fact it's a short(), an exception is raised.

match_structured(Filter, Event) -> Reply
Types:
- Filter = #objref
- Event = #'CosNotification.StructuredEvent'
- Reply = boolean() | {'EXCEPTION', #CosNotifyFilter.UnsupportedFilterableData'}

This operation is similar to the matchy operation but accepts structured events instead.

attach_callback(Filter, NotifySubscribe) -> CallbackID
Types:
- Filter = #objref
- NotifySubscribe = #objref
- CallbackID = long()

This operation connects a NotifySubscribe object, which should be informed when the target object’s constraints are updated. A unique ID is returned which must be stored if we ever want to detach the callback object in the future.

detach_callback(Filter, CallbackID) -> Reply
Types:
- Filter = #objref
- CallbackID = long()
- Reply = ok | {'EXCEPTION', #CosNotifyFilter.CallbackNotFound'}
If the target object has an associated callback that matches the supplied Id it will be removed and longer informed of any updates. If no object with a matching Id is found an exception is raised.

get_callbacks(Filter) -> CallbackIDSeq

Types:
- Filter = #objref
- CallbackIDSeq = [CallbackID]
- CallbackID = long()

This operation returns a sequence of all connected NotifySubscribe object Id’s. If no callbacks are associated with the target object the list will be empty.
CosNotifyFilter_FilterAdmin

Erlang Module

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

All objects, which inherit this interface, export functions described in this module.

Exports

```erlang
add_filter(Object, Filter) -> FilterID
Types:
  • Object = #objref
  • Filter = #objref
  • FilterID = long()
This operation connects a new Filter to the target object. This Filter will, together
with other associated Filters, be used to select events to forward. A unique Id is
returned and should be used if we no longer want to consult the given Filter.
```

```erlang
remove_filter(Object, FilterID) -> ok
Types:
  • Object = #objref
  • FilterID = long()
If a certain Filter no longer should be associated with the target object this operation
must be used. Events will no longer be tested against the Filter associated with the
given Id.
```

```erlang
get_filter(Object, FilterID) -> Reply
Types:
  • Object = #objref
  • FilterID = long()
  • Reply = Filter | {EXCEPTION, #CosNotifyFilter_FilterNotFound'{}
  • Filter = #objref
If the target object is associated with a Filter matching the given Id the reference will
be returned. If no such Filter is known by the target object an exception is raised.
```

```erlang
get_all_filters(Object) -> FilterIDSeq
Types:
```
- Object = #objref
- FilterID Seq = [FilterID]
- FilterID = long()

Id’s for all Filter objects associated with the target object is returned by this operation.

remove_all_filters(Object) -> ok

Types:
- Object = #objref

If we want to remove all Filters associated with the target object we can use this function.
CosNotifyFilter_FilterFactory

Erlang Module

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

Exports

create_filter(FilterFactory, Grammar) -> Reply
Types:
- FilterFactory = #objref
- Grammar = string()
- Reply = Filter | {'EXCEPTION', #CosNotifyFilter::InvalidGrammar'[]}
- Filter = #objref
This operation creates a new Filter object, under the condition that Grammar given is supported. Currently, only "EXTENDED_TCL" is supported.

create_mapping_filter(FilterFactory, Grammar) -> Reply
Types:
- FilterFactory = #objref
- Grammar = string()
- Reply = MappingFilter | {'EXCEPTION', #CosNotifyFilter::InvalidGrammar'[]}
- Filter = #objref
This operation creates a new MappingFilter object, under the condition that Grammar given is supported. Currently, only "EXTENDED_TCL" is supported.
CosNotifyFilter_MappingFilter

Erlang Module

The main purpose of this module is to match events against associated constraints and return the value for the first constraint that returns true for the given event. If all constraints return false the default value will be returned.

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

Exports

$get_constraint_grammar(MappingFilter) -> Grammar

Types:
- MappingFilter = #objref
- Grammar = string()

This operation returns which type of Grammar the MappingFilter uses. Currently, only "EXTENDED_TCL" is supported.

$get_value_type(MappingFilter) -> CORBA::TypeCode

Types:
- MappingFilter = #objref

This readonly attribute maintains the CORBA::TypeCode of the default value associated with the target object.

$get_default_value(MappingFilter) -> #any

Types:
- MappingFilter = #objref

This readonly attribute maintains the #any{} defualt value associated with the target object.

add_mapping_constraints(MappingFilter, MappingConstraintPairSeq) -> Reply

Types:
- MappingFilter = #objref
- MappingConstraintPairSeq = [MappingConstraintPair]
- MappingConstraintPair = 
  #CosNotifyFilter_MappingConstraintPair\{constraint_expression, result_to_set\}
- constraint_expression = #CosNotifyFilter_ConstraintExp\{event_types, constraint_expr\}
This operation adds new mapping constraints, which will be used when trying to override Quality of Service settings defined in the given event. If a constraint returns true the associated value will be returned, otherwise the default value.

```plaintext
modify_constraints(MappingFilter, ConstraintIDSeq, MappingConstraintInfoSeq) -> Reply
```

**Types:**
- `MappingFilter` = #objref
- `ConstraintIDSeq` = [ConstraintID]
- `ConstraintID` = long()
- `MappingConstraintInfoSeq` = [MappingConstraintInfo]
- `MappingConstraintInfo` = #CosNotifyFilter_MappingConstraintInfo`{constraint_expression, constraint_id, value}
- `constraint_expression` = ConstraintExp
- `constraint_id` = long()
- `value` = #any
- `ConstraintInfoSeq` = [ConstraintInfo]
- `ConstraintInfo` = #CosNotifyFilter_ConstraintInfo`{constraint_expression, constraint_id}
- `constraint_expression` = ConstraintExp
- `constraint_id` = long()
- `Reply` = ok | {'EXCEPTION', #CosNotifyFilter_InvalidConstraint'{constr}} | {'EXCEPTION', #CosNotifyFilter_InvalidValue'{constr, value}}
- `constr` = ConstraintExp
- `id` = long()
- `value` = #any
- `ConstraintExp` = #CosNotifyFilter_ConstraintExp`{event_types, constraint_expr}
- `event_types` = #CosNotification_EventTypeSeq`
- `constraint_expr` = string()
The ConstraintIDSeq supplied should relate to constraints the caller wishes to remove. If any of the supplied Id's are not found an exception will be raised. This operation also accepts a sequence of MappingConstraintInfo which will be added. If the target object cannot modify the constraints as requested an exception is raised describing which constraint, and why, could not be updated.

get_mapping_constraints(MappingFilter, ConstraintIDSeq) -> Reply

Types:
- MappingFilter = #objref
- ConstraintIDSeq = [ConstraintID]
- ConstraintID = long()
- Reply = MappingConstraintInfoSeq | {'EXCEPTION', #CosNotifyFilter.ConstraintNotFound\{id\}}
- MappingConstraintInfoSeq = [MappingConstraintInfo]
- MappingConstraintInfo = #CosNotifyFilter.MappingConstraintInfo\{constraint_expression, constraint_id, value\}
- constraint_expression = ConstraintExp
- ConstraintExp = #CosNotifyFilter.ConstraintExp\{event_types, constraint_expr\}
- event_types = #CosNotification.EventTypeSeq\{}
- constraint_expr = string()
- constraint_id = id = long()
- value = #any

When adding a new constraint a unique Id is returned, which is accepted as input for this operation. The associated constraint is returned, but if no such Id exists an exception is raised.

get_all_mapping_constraints(MappingFilter) -> MappingConstraintInfoSeq

Types:
- MappingFilter = #objref
- MappingConstraintInfoSeq = [MappingConstraintInfo]
- MappingConstraintInfo = #CosNotifyFilter.MappingConstraintInfo\{constraint_expression, constraint_id, value\}
- constraint_expression = ConstraintExp
- ConstraintExp = #CosNotifyFilter.ConstraintExp\{event_types, constraint_expr\}
- event_types = #CosNotification.EventTypeSeq\{}
- constraint_expr = string()
- constraint_id = id = long()
- value = #any

This operation returns a sequence of all unique Id's associated with the target object. If no constraint have been added the sequence will be empty.

remove_all_mapping_constraints(MappingFilter) -> ok

Types:
- MappingFilter = #objref
This operation removes all constraints associated with the target object.

\[
\text{destroy(MappingFilter)} \rightarrow \text{ok}
\]

Types:
- \text{MappingFilter} = \#objref

This operation terminates the target object. Remember to remove this Filter from the objects it have been associated with.

\[
\text{match(MappingFilter, Event)} \rightarrow \text{Reply}
\]

Types:
- \text{MappingFilter} = \#objref
- \text{Event} = \#any
- \text{Reply} = \{\text{boolean()}, \#any\} \cup \{\text{EXCEPTION}, \text{'CosNotifyFilter\_UnsupportedFilterableData'}\}

This operation evaluates \text{any} events with the Filter's constraints, and returns the value to use. The value is the default value if all constraints returns false and the value associated with the first constraint returning true.

\[
\text{match\_structured(MappingFilter, Event)} \rightarrow \text{Reply}
\]

Types:
- \text{MappingFilter} = \#objref
- \text{Event} = \#\text{CosNotification\_StructuredEvent}
- \text{Reply} = \{\text{boolean()}, \#any\} \cup \{\text{EXCEPTION}, \text{'CosNotifyFilter\_UnsupportedFilterableData'}\}

Similar to \text{match/2} but accepts a structured event as input.
cosNotificationApp

Erlang Module

To get access to the record definitions for the structures use:
-include_lib("cosNotification/include/*.hrl").
This module contains the functions for starting and stopping the application.

Exports

install() -> Return
Types:
  • Return = ok | 'EXCEPTION', E
This operation installs the cosNotification application.

install(Seconds) -> Return
Types:
  • Return = ok | 'EXCEPTION', E
This operation installs the cosNotification application using Seconds delay between each block, currently 6, of IFR-registrations. This approach spreads the IFR database access over a period of time to allow other applications to run smoother.

install_event() -> Return
Types:
  • Return = ok | 'EXCEPTION', E
This operation installs the necessary cosEvent interfaces. If cosEvent is already installed this operation may not be used.

install_event(Seconds) -> Return
Types:
  • Return = ok | 'EXCEPTION', E
This operation installs the necessary cosEvent interfaces using Seconds delay between each block of IFR-registrations. If cosEvent is already installed this operation may not be used.

uninstall() -> Return
Types:
  • Return = ok | 'EXCEPTION', E
This operation uninstalls the cosNotification application.

uninstall(Seconds) -> Return

Types:
- Return = ok | {'EXCEPTION', E}
This operation uninstalls the cosNotification application using Seconds delay between each block, currently 6, of IFR-unregistrations. This approach spreads the IFR database access over a period of time to allow other applications to run smoother.

uninstall_event() -> Return

Types:
- Return = ok | {'EXCEPTION', E}
This operation uninstalls the inherited cosEvent interfaces. If cosEvent is in use this function may not be used.

uninstall_event(Seconds) -> Return

Types:
- Return = ok | {'EXCEPTION', E}
This operation uninstalls the inherited cosEvent interfaces, using Seconds delay between each block of IFR-unregistrations. If cosEvent is in use this function may not be used.

start() -> Return

Types:
- Return = ok | {error, Reason}
This operation starts the cosNotification application.

stop() -> Return

Types:
- Return = ok | {error, Reason}
This operation stops the cosNotification application.

start_global_factory() -> ChannelFactory

Types:
- ChannelFactory = #objref
This operation creates a Event Channel Factory [page 42] should be used for a multi-node Orber. The Factory is used to create a new channel [page 39].

start_global_factory(Options) -> ChannelFactory

Types:
- Options = [Option]
- Option = {pullInterval, Seconds} | {filterOp, Op} | {gcTime, Seconds} | {gcLimit, Amount} | {timeService, #objref}
This operation creates a Event Channel Factory [page 42] and should be used for a multi-node Orber. The Factory is used to create a new channel [page 39].

- \{pullInterval, Seconds\} - determine how often Proxy Pull Consumers will check for new events with the client application. The default value is 20 seconds.
- \{filterOp, OperationType\} - determine which type of Administrator objects should be started, i.e., 'OR\_OP' or 'AND\_OP'. The default value is 'OR\_OP'.
- \{timeService, TimeServiceObj | 'undefined'\} - to be able to use Start and/or Stop QoS this option must be used. See the function start\_time\_service/2 in the cosTime application. The default value is 'undefined'.
- \{filterOp, OperationType\} - determine which type of Administrator objects should be started, i.e., 'OR\_OP' or 'AND\_OP'. The default value is 'OR\_OP'.
- \{gcTime, Seconds\} - this option determines how often, for example, proxies will garbage collect expired events. The default value is 60.
- \{gcLimit, Amount\} - determines how many events will be stored before, for example, proxies will garbage collect expired events. The default value is 50. This option is tightly coupled with the QoS property MaxEventsPerConsumer, i.e., the gcLimit should be less than MaxEventsPerConsumer and greater than 0.

`start\_factory()` \rightarrow \texttt{ChannelFactory}

Types:
- \texttt{ChannelFactory = #objref}

This operation creates a Event Channel Factory [page 42]. The Factory is used to create a new channel [page 39].

`start\_factory(\texttt{Options})` \rightarrow \texttt{ChannelFactory}

Types:
- \texttt{Options = [Option]}
- \texttt{Option = \{pullInterval, Seconds\} | \{filterOp, O p\} | \{gcTime, Seconds\} | \{gcLimit, Amount\} | \{timeService, #objref\}}
- \texttt{ChannelFactory = #objref}

This operation creates a Event Channel Factory [page 42]. The Factory is used to create a new channel [page 39].

`stop\_factory(\texttt{ChannelFactory})` \rightarrow \texttt{Reply}

Types:
- \texttt{ChannelFactory = #objref}
- \texttt{Reply = ok | \{'EXCEPTION', E\}}

This operation stop the target channel factory.

`start\_filter\_factory()` \rightarrow \texttt{FilterFactory}

Types:
- \texttt{FilterFactory = #objref}
This operation creates a Filter Factory [page 82]. The Factory is used to create a new Filter's [page 76] and MappingFilter's [page 83].

stop_filter_factory(FilterFactory) -> Reply

Types:
  • FilterFactory = #objref
  • Reply = ok | {'EXCEPTION', E}

This operation stop the target filter factory.
List of Figures

1.1 Figure 1: The Notification Service Components ......................................... 4
1.2 Figure 1: The structure of a structured event. ............................................... 11
List of Tables

1.1 Table 1: Type & Operator Examples ........................................ 11
1.2 Table 2: Fixed Header Constraint Examples ................................. 12
1.3 Table 1: Supported QoS settings .............................................. 13
1.4 Table 2: Supported Admin Properties ......................................... 15
Index of Modules and Functions

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

_get_MyAdmin/1
  CosNotifyChannelAdmin_ProxyConsumer, 43
  CosNotifyChannelAdmin_ProxySupplier, 52

_get_MyChannel/1
  CosNotifyChannelAdmin_-_ConsumerAdmin, 39
  CosNotifyChannelAdminSupplierAdmin, 59

_get_MyFactory/1
  CosNotifyChannelAdmin_EventChannel, 39

_get_MyID/1
  CosNotifyChannelAdmin_-_ConsumerAdmin, 36
  CosNotifyChannelAdminSupplierAdmin, 59

_get_MyOperator/1
  CosNotifyChannelAdmin_-_ConsumerAdmin, 36
  CosNotifyChannelAdminSupplierAdmin, 59

_get_MyType/1
  CosNotifyChannelAdmin_ProxyConsumer, 43
  CosNotifyChannelAdmin_ProxySupplier, 52

_get_constraint_grammar/1
  CosNotifyFilter_Filter, 76
  CosNotifyFilter_MappingFilter, 83

_get_default_consumer_admin/1
  CosNotifyChannelAdmin_EventChannel, 39

_get_default_filter_factory/1
  CosNotifyChannelAdmin_EventChannel, 39

_get_default_supplier_admin/1
  CosNotifyChannelAdmin_EventChannel, 39

_get_default_value/1
  CosNotifyFilter_MappingFilter, 83

_get_lifetime_filter/1
  CosNotifyChannelAdmin_-_ConsumerAdmin, 37
  CosNotifyChannelAdmin_ProxySupplier, 53

_get_priority_filter/1
  CosNotifyChannelAdmin_-_ConsumerAdmin, 36
  CosNotifyChannelAdmin_ProxySupplier, 52

_get_pull_consumers/1
  CosNotifyChannelAdminSupplierAdmin, 59

_get_pull_suppliers/1
  CosNotifyChannelAdmin_-_ConsumerAdmin, 37

_get_push_consumers/1
  CosNotifyChannelAdminSupplierAdmin, 60

_get_push_suppliers/1
  CosNotifyChannelAdmin_-_ConsumerAdmin, 37

_get_value_type/1
  CosNotifyFilter_MappingFilter, 83
Index of Modules and Functions

_set_lifetime_filter/2
CosNotifyChannelAdmin-
ConsumerAdmin, 37
CosNotifyChannelAdmin_ProxySupplier, 53

_set_priority_filter/2
CosNotifyChannelAdmin-
ConsumerAdmin, 37
CosNotifyChannelAdmin_ProxySupplier, 52

add_constraints/2
CosNotifyFilter_Filter, 76

add_filter/2
CosNotifyFilter_FilterAdmin, 80

add_mapping_constraints/2
CosNotifyFilter_MappingFilter, 83

attach_callback/2
CosNotifyFilter_Filter, 78

connect_any_pull_consumer/2
CosNotifyChannelAdmin-
ProxyPullSupplier, 47

connect_any_pull_supplier/2
CosNotifyChannelAdmin-
ProxyPullConsumer, 45

connect_any_push_consumer/2
CosNotifyChannelAdmin-
ProxyPushSupplier, 50

connect_any_push_supplier/2
CosNotifyChannelAdmin-
ProxyPushConsumer, 49

connect_sequence_pull_consumer/2
CosNotifyChannelAdmin-
SequenceProxyPullSupplier, 57

connect_sequence_pull_supplier/2
CosNotifyChannelAdmin-
SequenceProxyPullConsumer, 62

connect_sequence_push_consumer/2
CosNotifyChannelAdmin-
SequenceProxyPushSupplier, 68

connect_sequence_push_supplier/2
CosNotifyChannelAdmin-
SequenceProxyPushConsumer, 70

connect_structured_pull_consumer/2
CosNotifyChannelAdmin-
StructuredProxyPullSupplier, 64

connect_structured_pull_supplier/2
CosNotifyChannelAdmin-
StructuredProxyPullConsumer, 66

connect_structured_push_consumer/2
CosNotifyChannelAdmin-
StructuredProxyPushSupplier, 72

connect_structured_push_supplier/2
CosNotifyChannelAdmin-
StructuredProxyPushConsumer, 70

CosNotification_AdminPropertiesAdmin
    get_admin/1, 33
    set_admin/2, 33

CosNotification_QoSAdmin
    get_qos/1, 34
    set_qos/2, 34
    validate_qos/2, 35

CosNotification_Application
    install/0, 87
    install/1, 87
    install_event/0, 87
    install_event/1, 87
    start/0, 88
    start_factory/0, 89
    start_factory/1, 89
    start_filter_factory/0, 89
    start_global_factory/0, 88
    start_global_factory/1, 88
    stop/0, 88
    stop_factory/1, 89
    stop_filter_factory/1, 90
    uninstall/0, 87
    uninstall/1, 88
    uninstall_event/0, 88
    uninstall_event/1, 88

CosNotifyChannelAdmin_ConsumerAdmin
Index of Modules and Functions

CosNotifyChannelAdmin

_CosNotifyChannelAdmin_  
_EventChannel_  
_get_MyChannel/1, 36  
_get_MyID/1, 36  
_get_MyOperator/1, 36  
_get_lifetime_filter/1, 37  
_get_priority_filter/1, 36  
_get_pull_suppliers/1, 37  
_get_push_suppliers/1, 37  
_set_lifetime_filter/2, 37  
_set_priority_filter/2, 37  
destroy/1, 38  
get_proxy_supplier/2, 37  
obtain_notification_pull_supplier/2, 38  
obtain_notification_push_supplier/2, 38  
obtain_pull_supplier/1, 38  
obtain_push_supplier/1, 38  

CosNotifyChannelAdmin

_EventChannelFactory_  
_create_channel/3, 42  
get_all_channels/1, 42  
get_event_channel/2, 42  

CosNotifyChannelAdmin

_ProxyConsumer_  
_get_MyAdmin/1, 43  
_get_MyType/1, 43  
obtain_subscriptions_types/2, 43  
validate_event_qos/2, 44  

CosNotifyChannelAdmin

_ProxyPullConsumer_  
connect_any_pull_supplier/2, 45  
disconnect_pull_consumer/1, 46  
resume_connection/1, 45  
suspend_connection/1, 45  

CosNotifyChannelAdmin

_ProxyPullSupplier_  
connect_any_push_consumer/2, 49  
disconnect_push_consumer/1, 49  
push/2, 49  

CosNotifyChannelAdmin

_ProxyPushConsumer_  
connect_any_pull_supplier/2, 49  
disconnect_sequence_pull_consumer/1, 49  
push/2, 49  

CosNotifyChannelAdmin

_ProxyPushSupplier_  
connect_any_push_consumer/2, 50  
disconnect_push_supplier/1, 51  
resume_connection/1, 50  
suspend_connection/1, 50  

CosNotifyChannelAdmin

_ProxySequencePullConsumer_  
connect_sequence_pull_supplier/2, 50  
disconnect_sequence_pull_consumer/1, 50  
resume_connection/1, 50  
suspend_connection/1, 50  

CosNotifyChannelAdmin

_ProxySequencePullSupplier_  
connect_sequence_pull_supplier/2, 51  
disconnect_sequence_pull_consumer/1, 51  
pull/1, 47  
try_pull/1, 47  

CosNotifyChannelAdmin

_ProxySequencePushConsumer_  
connect_sequence_push_supplier/2, 51  
disconnect_sequence_push_consumer/1, 51  
pull/1, 47  
try_pull/1, 47  

CosNotifyChannelAdmin

_ProxySequencePushSupplier_  
connect_sequence_push_supplier/2, 52  
disconnect_sequence_push_consumer/1, 52  
pull/1, 47  
try_pull/1, 47  

CosNotifyChannelAdmin

_ProxyStructuredPullConsumer_  
connect_structured_pull_supplier/2, 52  
disconnect_sequence_push_consumer/1, 52  
pull/1, 47  
try_pull/1, 47  

CosNotifyChannelAdmin

_ProxyStructuredPushConsumer_  
connect_sequence_push_supplier/2, 53  
disconnect_sequence_push_consumer/1, 53  
pull/1, 47  
try_pull/1, 47  

CosNotifyChannelAdmin

_ProxyStructuredPushSupplier_  
connect_sequence_push_supplier/2, 54  
disconnect_sequence_push_consumer/1, 54  
pull/1, 47  
try_pull/1, 47  

CosNotifyChannelApplication
Index of Modules and Functions

66
disconnect_structured_pull_consumer/1, 67
resume_connection/1, 66
suspend_connection/1, 66
CosNotifyChannelAdmin-
StructuredProxyPullSupplier
connect_structured_pull_consumer/2, 68
disconnect_structured_pull_supplier/1, 69
pull_structured_event/1, 68
try_pull_structured_event/1, 69
CosNotifyChannelAdmin-
StructuredProxyPushConsumer
connect_structured_push_consumer/2, 70
disconnect_structured_push_supplier/1, 71
push_structured_event/2, 70
CosNotifyChannelAdmin-
StructuredProxyPushSupplier
connect_structured_push_consumer/2, 72
disconnect_structured_push_supplier/1, 73
resume_connection/1, 72
suspend_connection/1, 72
CosNotifyChannelAdmin_SupplierAdmin
_get_MyChannel/1, 59
_get_MyID/1, 59
_get_PullOperator/1, 59
_get_pull_consumers/1, 59
_get_push_consumers/1, 60
destroy/1, 61
get_proxy_consumer/2, 60
obtain_notification_pull_consumer/2, 60
obtain_notification_push_consumer/2, 60
obtain_pull_consumer/1, 60
obtain_push_consumer/1, 61
CosNotifyComm_NotifySubscribe
subscription_change/3, 75
CosNotifyComm_NotifyPublish
offer_change/3, 74
CosNotifyFilter_Filter
_get_constraint_grammar/1, 76
add_constraints/2, 76
attach_callback/2, 78
destroy/1, 78
detach_callback/2, 78
get_all_constraints/1, 77
get_callbacks/1, 79
get_constraints/2, 77
match/2, 78
match_structured/2, 78
modify_constraints/3, 76
remove_all_constraints/1, 77
CosNotifyFilter_FilterAdmin
add_filter/2, 80
get_all_filters/1, 80
get_filter/2, 80
remove_all_filters/1, 81
remove_filter/2, 80
CosNotifyFilter_FilterFactory
create_filter/2, 82
create_mapping_filter/2, 82
CosNotifyFilter_MappingFilter
_get_constraint_grammar/1, 83
_get_default_value/1, 83
_get_value_type/1, 83
add_mapping_constraints/2, 83
destroy/1, 86
get_all_mapping_constraints/1, 85
get_mapping_constraints/2, 85
match/2, 86
match_structured/2, 86
modify_constraints/3, 84
remove_all_mapping_constraints/1, 85
create_channel/3
CosNotifyChannelAdmin.EventChannelFactory
create_filter/2
CosNotifyFilter_FilterFactory, 82
create_mapping_filter/2
CosNotifyFilter_FilterFactory, 82
destroy/1
CosNotifyChannelAdmin.ConsumerAdmin
CosNotifyChannelAdmin.EventChannel
CosNotifyChannelAdmin.SupplierAdmin, 61
CosNotifyFilter.Filter
CosNotifyFilter.MappingFilter
create_mapping_filter/2
CosNotifyFilter.MappingFilter, 86
Index of Modules and Functions

detach_callback/2
  CosNotifyFilter, 78
disconnect_pull_consumer/1
  CosNotifyChannelAdmin_ProxyPullConsumer, 46
disconnect_pull_supplier/1
  CosNotifyChannelAdmin_ProxyPullSupplier, 48
disconnect_push_consumer/1
  CosNotifyChannelAdmin_ProxyPushConsumer, 49
disconnect_push_supplier/1
  CosNotifyChannelAdmin_ProxyPushSupplier, 51
disconnect_sequence_pull_consumer/1
  CosNotifyChannelAdmin_SequenceProxyPullConsumer, 56
disconnect_sequence_pull_supplier/1
  CosNotifyChannelAdmin_SequenceProxyPullSupplier, 58
disconnect_sequence_push_consumer/1
  CosNotifyChannelAdmin_SequenceProxyPushConsumer, 63
disconnect_sequence_push_supplier/1
  CosNotifyChannelAdmin_SequenceProxyPushSupplier, 65
disconnect_structured_pull_consumer/1
  CosNotifyChannelAdmin_StructuredProxyPullConsumer, 67
disconnect_structured_pull_supplier/1
  CosNotifyChannelAdmin_StructuredProxyPullSupplier, 69
disconnect_structured_push_consumer/1
  CosNotifyChannelAdmin_StructuredProxyPushConsumer, 71
disconnect_structured_push_supplier/1
  CosNotifyChannelAdmin_StructuredProxyPushSupplier, 73
for_consumers/1
  CosNotifyChannelAdmin_EventChannel, 40
for_suppliers/1
  CosNotifyChannelAdmin_EventChannel, 40
get_admin/1
  CosNotification_AdminPropertiesAdmin, 33
get_all_channels/1
  CosNotifyChannelAdmin_EventChannelFactory, 42
get_all_constraints/1
  CosNotifyFilter_Filter, 77
get_all_consumeradmins/1
  CosNotifyChannelAdmin_EventChannel, 41
get_all_filters/1
  CosNotifyFilter_FilterAdmin, 80
get_all_mapping_constraints/1
  CosNotifyFilter_MappingFilter, 85
get_all_supplieradmins/1
  CosNotifyChannelAdmin_EventChannel, 41
get_callbacks/1
  CosNotifyFilter_Filter, 79
get_constraints/2
  CosNotifyFilter_Filter, 77
get_consumeradmin/2
  CosNotifyChannelAdmin_EventChannel, 41
get_event_channel/2
  CosNotifyChannelAdmin_EventChannelFactory, 42
get_filter/2
  CosNotifyFilter_FilterAdmin, 80
get_mapping_constraints/2
  CosNotifyFilter_MappingFilter, 85
get_proxy_consumer/2

cosNotification Application
<table>
<thead>
<tr>
<th>Function/Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_proxy_supplier/2</td>
<td>60</td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_SupplierAdmin</td>
<td></td>
</tr>
<tr>
<td>obtain_notification_push_supplier/2</td>
<td>60</td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_SupplierAdmin</td>
<td></td>
</tr>
<tr>
<td>obtain_notification_push_supplier/2</td>
<td></td>
</tr>
<tr>
<td>CosNotifyFilter_Filter</td>
<td>78</td>
</tr>
<tr>
<td>match/2</td>
<td></td>
</tr>
<tr>
<td>CosNotifyFilter_MappingFilter</td>
<td>86</td>
</tr>
<tr>
<td>obtain_push_consumer/1</td>
<td>37</td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_ConsumerAdmin</td>
<td></td>
</tr>
<tr>
<td>obtain_push_consumer/1</td>
<td></td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_SupplierAdmin</td>
<td></td>
</tr>
<tr>
<td>obtain_push_supplier/1</td>
<td></td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_SupplierAdmin</td>
<td></td>
</tr>
<tr>
<td>obtain_push_supplier/1</td>
<td></td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_SupplierAdmin</td>
<td></td>
</tr>
<tr>
<td>obtain_pull_consumer/1</td>
<td></td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_SupplierAdmin</td>
<td></td>
</tr>
<tr>
<td>obtain_pull_supplier/1</td>
<td></td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_SupplierAdmin</td>
<td></td>
</tr>
<tr>
<td>obtain_pull_supplier/1</td>
<td></td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_SupplierAdmin</td>
<td></td>
</tr>
<tr>
<td>obtain_push_supplier/1</td>
<td></td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_SupplierAdmin</td>
<td></td>
</tr>
<tr>
<td>obtain_push_supplier/1</td>
<td></td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_SupplierAdmin</td>
<td></td>
</tr>
<tr>
<td>obtain_push_supplier/1</td>
<td></td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_SupplierAdmin</td>
<td></td>
</tr>
<tr>
<td>obtain_push_supplier/1</td>
<td></td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_SupplierAdmin</td>
<td></td>
</tr>
<tr>
<td>obtain_push_supplier/1</td>
<td></td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_SupplierAdmin</td>
<td></td>
</tr>
<tr>
<td>obtain_push_supplier/1</td>
<td></td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_SupplierAdmin</td>
<td></td>
</tr>
<tr>
<td>obtain_push_supplier/1</td>
<td></td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_SupplierAdmin</td>
<td></td>
</tr>
<tr>
<td>obtain_push_supplier/1</td>
<td></td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_SupplierAdmin</td>
<td></td>
</tr>
<tr>
<td>obtain_push_supplier/1</td>
<td></td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_SupplierAdmin</td>
<td></td>
</tr>
<tr>
<td>obtain_subscribed_types/2</td>
<td>43</td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_PullSupplier</td>
<td></td>
</tr>
<tr>
<td>offer_change/3</td>
<td></td>
</tr>
<tr>
<td>CosNotifyComm_NotifyPublish</td>
<td>74</td>
</tr>
<tr>
<td>pull/1</td>
<td></td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_PullSupplier</td>
<td></td>
</tr>
<tr>
<td>pull_structured_event/1</td>
<td></td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_PullSupplier</td>
<td></td>
</tr>
<tr>
<td>pull_structured_events/2</td>
<td>68</td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_PullSupplier</td>
<td></td>
</tr>
<tr>
<td>pull_structured_events/2</td>
<td></td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_PullSupplier</td>
<td></td>
</tr>
<tr>
<td>push/2</td>
<td></td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_PullSupplier</td>
<td>49</td>
</tr>
<tr>
<td>push_structured_event/2</td>
<td></td>
</tr>
<tr>
<td>CosNotifyChannelAdmin_PullSupplier</td>
<td></td>
</tr>
</tbody>
</table>
Index of Modules and Functions

CosNotifyChannelAdmin - StructuredProxyPushConsumer, 70
push_structured_events/2

CosNotifyChannelAdmin - SequenceProxyPushConsumer, 62
remoce_all_constraints/1
CosNotifyFilter_Filter, 77
remove_filters/1
CosNotifyFilter_FilterAdmin, 81
remove_all_mapping_constraints/1
CosNotifyFilter_MappingFilter, 85
remove_2ilter/2
CosNotifyFilter_FilterAdmin, 80
resume_connection/1

CosNotifyChannelAdmin - ProxyPullConsumer, 45
CosNotifyChannelAdmin - ProxyPushSupplier, 50
CosNotifyChannelAdmin - SequenceProxyPullConsumer, 55
CosNotifyChannelAdmin - SequenceProxyPushSupplier, 64
CosNotifyChannelAdmin - StructuredProxyPullConsumer, 66
CosNotifyChannelAdmin - StructuredProxyPushSupplier, 72

set_admin/2
CosNotification_AdminPropertiesAdmin, 33
set_qos/2
CosNotification_QoSAdmin, 34
start/0
cosNotificationApp, 89
start_factory/0
cosNotificationApp, 89
start_factory/1
cosNotificationApp, 89
start_filter_actory/0

start_global_factory/0
cosNotificationApp, 89
start_global_factory/1
cosNotificationApp, 89
stop/0
cosNotificationApp, 89
stop_actory/1
cosNotificationApp, 89
stop_filter_actory/1
cosNotificationApp, 90
subscription_change/3
CosNotifyComm_NotifySubscribe, 75

suspend_connection/1

CosNotifyChannelAdmin - ProxyPullConsumer, 45
CosNotifyChannelAdmin - ProxyPushSupplier, 50
CosNotifyChannelAdmin - SequenceProxyPullConsumer, 55
CosNotifyChannelAdmin - SequenceProxyPushSupplier, 64
CosNotifyChannelAdmin - StructuredProxyPullConsumer, 66
CosNotifyChannelAdmin - StructuredProxyPushSupplier, 72

try_pull/1
CosNotifyChannelAdmin - ProxyPullSupplier, 47
try_pull_structured_event/1
CosNotifyChannelAdmin - StructuredProxyPullSupplier, 69
try_pull_structured_events/2
CosNotifyChannelAdmin - SequenceProxyPullSupplier, 58

uninstall/0
cosNotificationApp, 87
Index of Modules and Functions

uninstall/1
  cosNotificationApp , 88

uninstall_event/0
  cosNotificationApp , 88

uninstall_event/1
  cosNotificationApp , 88

validate_event_qos/2
  CosNotifyChannelAdmin_ProxyConsumer , 44
  CosNotifyChannelAdmin_ProxySupplier , 53

validate_qos/2
  CosNotification_QoSAdmin , 35