cosNotification Application

version 1.1
Typeset in \LaTeX{} from SGML source using the DOCUMENT 3.4.1 Document System.
Contents

1 cosNotification User's Guide ........................................... 1
  1.1 The cosNotification Application .................................. 1
    1.1.1 Content Overview .................................. 1
    1.1.2 Brief Description of the User's Guide ..................... 1
  1.2 Introduction to cosNotification ................................ 1
    1.2.1 Overview ....................................... 1
  1.3 Installing cosNotification .................................... 2
    1.3.1 Installation Process .................................. 2
  1.4 The Notification Service Components ............................ 3
    1.4.1 The Notification Service Components ..................... 3
  1.5 Filters and the Constraint Language BNF ....................... 5
    1.5.1 Filters and the Constraint Language BNF ................. 5
  1.6 Quality of Service and Admin Properties ....................... 13
    1.6.1 Quality of Service and Admin Properties ................ 13
  1.7 cosNotification Examples ..................................... 16
    1.7.1 A Tutorial on How to Create a Simple Service ........... 16
List of Figures

List of Tables
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.

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 books recommended by the OMG 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 90] 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 45] 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'.
1.4: The Notification Service Components

- `{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.

It is possible to define a set of global configuration parameters:

<table>
<thead>
<tr>
<th>Key</th>
<th>Range</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>type_check</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>notify</td>
<td>atom()</td>
<td>false</td>
</tr>
<tr>
<td>max_events</td>
<td>integer() &gt; 0</td>
<td>50</td>
</tr>
<tr>
<td>interval_events</td>
<td>integer() &gt; 0</td>
<td>10000 milliseconds</td>
</tr>
<tr>
<td>timeout_events</td>
<td>integer() &gt; interval_events</td>
<td>3000000 milliseconds</td>
</tr>
</tbody>
</table>

Table 1.1: Global Configuration Parameters

Comments on the table ‘Global Configuration Parameters’:

**type_check**  Determine if supplied IOR:s shall be type checked, i.e. invoking corba\_object:is\_a/2, or not.

**notify**  The given value shall point to an existing module exporting a function (arity 1) called terminated. This operation is invoked when a proxy terminates and the argument is a list containing `{proxy, IOR}`, `{client, IOR}` and `{reason, term()}. The return value is ignored.

**max_events**  If a supplier proxy has not been able to push events to a consumer and the queue exceeds this limit, then the proxy will terminate. For this option to have any effect, the EventReliability and ConnectionReliability QoS parameters must be set to Persistent. For more information, see also the QoS [page 13] chapter.

**interval_events**  The same requirements as for max_events. When a supplier proxy detects problems when trying to push events, this parameter determines how often it should try to call the consumer.

**timeout_events**  The same requirements as for max_events. If the proxy has not been able to contact the consumer and this time-limit is reached, then the proxy will terminate.

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.
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 79] and CosNotifyFilter MappingFilter [page 86], 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 Orber 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:

```prolog
#'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:

```prolog
<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>
```

6

cosNotification Application
1.5: Filters and the Constraint Language BNF

| <term> / <factor_not>  
| <factor_not>  

<factor_not> := not <factor>  
| <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> := /* empty */  
| <Digits>  
| - <Digits>  
| + <Digits>  
| <String>  

/* Character set issues */
Chapter 1: cosNotification User's Guide

<Ident> ::= <Leader> <FollowSeq>
   | \ <Leader> <FollowSeq>

<FollowSeq> ::= /* empty */
   | <FollowSeq> <Follow>

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

<Mantissa> ::= <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
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>~</td>
<td>&quot;'String1' ~ 'String2'&quot;</td>
<td>The operator '~' is called the substring operator and mean &quot;String1 is contained within String2&quot;.</td>
</tr>
<tr>
<td>boolean</td>
<td>&quot;TRUE == ((&quot;lang&quot; ~ '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 $.FunctionalLanguagesStringSeq&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>
</tbody>
</table>

*continued...*
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td>.</td>
<td>&quot;.MyStructMember == 40&quot;</td>
<td>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.</td>
</tr>
<tr>
<td>.type_id</td>
<td>&quot;.type_id == 'MyStruct'&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>.repos_id</td>
<td>&quot;.repos_id == 'IDL:MyModule/MyStruct::1.0'&quot;</td>
<td>Returns the RepositoryId of the component. This operation is only valid if said information can be obtained.</td>
</tr>
<tr>
<td>.d</td>
<td>&quot;.eventUnion.d&quot;</td>
<td>May only be used when accessing discriminated unions and refers to the discriminator.</td>
</tr>
<tr>
<td>exist</td>
<td>&quot;exist .eventUnion.d and .eventUnion.d == 10&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>default</td>
<td>&quot;default .eventUnion.d&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>union</td>
<td>&quot;.(0) == 5&quot; eq &quot;.('zero') == 5&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 ...
1.5: Filters and the Constraint Language BNF

--- continued

### name-value pairs

<table>
<thead>
<tr>
<th>Name-value pairs</th>
<th>Constrained Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$.NameValueSeq('myID') == 5</code></td>
<td>eq. <code>$.NameValueSeq[0].name == 'myID' and $.NameValueSeq[0].value == 5</code></td>
<td>The Notification service makes extensive use of name-value pairs sequences within structured events, which allow us to via the identifier name access its value, as shown in the example.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 1.2: Table 1: Type &amp; Operator Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>In the next section we will take a closer look at how it is possible to write constraints using different types of notation etc.</td>
</tr>
</tbody>
</table>

**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.

![Event Structure Diagram](image.png)

**Figure 1.2: The structure of a structured event.**

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;$\text{.header.fixed}_\text{event}_{\text{type}}_\text{type_name} == \text{Type}&quot;</td>
<td>$\text{type_name} == \text{Type}&quot;</td>
</tr>
<tr>
<td>domain_name</td>
<td>&quot;$\text{.header.fixed}_\text{event}_{\text{domain_name}} == \text{Domain}&quot;</td>
<td>$\text{domain_name} == \text{Domain}&quot;</td>
</tr>
<tr>
<td>event_name</td>
<td>&quot;$\text{.header.fixed}_\text{event_name} == \text{Event}&quot;</td>
<td>$\text{event_name} == \text{Event}&quot;</td>
</tr>
</tbody>
</table>

Table 1.3: Table 2: Fixed Header Constraint Examples

If we are only interested in receiving events regarding 'Domain', 'Event' and 'Type', the constraint can look like "$\text{domain\_name} == \text{Domain} \text{ and } \text{event\_name} == \text{Event} \text{ and } \text{type\_name} == \text{Type}". The variable event header consists of a sequence of name-value pairs. One way to filter on these are to use a constraint that looks like "($\text{.header.variable}\_\text{header[1].name} == \text{priority} \text{ and } \text{.header.variable}\_\text{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 "$\text{.header.variable}\_\text{header(priority)} > 0". Accessing the event body is done in the same way as for the event header fields. The user must, however, be aware of, that if a run-time variable ($\text{variable}$) is used data in the event header may take precedence. The order of precedence is:

1. Reserved, e.g., $\text{curtime}$
2. A simple-typed member of $\text{.header.fixed}\_\text{header}$.
3. Properties in $\text{.header.variable}\_\text{header}$.
4. Properties in $\text{.filterable}\_\text{data}$.
5. If no match is found it is translated to $\text{.variable}$.
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 related Quality of Service system settings.

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

```plaintext
#'CosNotifyFilter_MappingConstraintPair'{
  constraint_expression = #'CosNotifyFilter_ConstraintExp'{
    event_types = [#'CosNotification_EventType'{
      domain_name = string(),
      type_name = string()},
    constraint_expr = string(),
  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 37] and CosNotification_AdminPropertiesAdmin [page 36].

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/Persistent</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>1</td>
</tr>
<tr>
<td>PacingInterval</td>
<td>TimeBase::TimeT (see cosTime)</td>
<td>0</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 1.4: Table 1: Supported QoS Settings
Comments on the table 'Supported QoS Settings':

**EventReliability** To allow full Persistent EventReliability, every event must be stored in a stable storage which would create a relatively huge overhead. Hence, only lightweight version of the Persistent QoS is supported. The configuration parameters `max_events`, `interval_events` and `timeout_events` determine the behavior of this setting.

**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. If `OBJECT_NOT_EXIST`, `NO_PERMISSION` or `CosEventComm_Disconnected` is thrown, the associated object will terminate even if this parameter is set to Persistent.

**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. If set to 0, which is the default behavior, no timeout is used and the events are forwarded when the MaximumBatchSize is reached.

**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:

```java
QoSPersistent = [#'CosNotification_Property'
{name='CosNotification':'ConnectionReliability'(),
 value=any:create(orber_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 it 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</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 1.5: 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.
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);
    }
}
```

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.
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.

%% 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.
cosEventApp:install(),

%% 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. Note, if no QoS- anr/or Admin-properties
%% are supplied (i.e. empty list) the default settings are used.
{Ch, ChID} = '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'),
Chapter 1: cosNotification User’s Guide

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

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

%%% 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.
cosNotification Reference Manual

Short Summaries

- Erlang Module **CosNotification** [page 33] - This module export functions which return QoS and Admin Properties constants.
- Erlang Module **CosNotification_AdminPropertiesAdmin** [page 36] - This module implements the OMG CosNotification::AdminPropertiesAdmin interface.
- Erlang Module **CosNotification_QoSAdmin** [page 37] - This module implements the OMG CosNotification::QoSAdmin interface.
- Erlang Module **CosNotifyChannelAdmin_ConsumerAdmin** [page 39] - This module implements the OMG CosNotifyChannelAdmin::ConsumerAdmin interface.
- Erlang Module **CosNotifyChannelAdmin_EventChannel** [page 42] - This module implements the OMG CosNotifyChannelAdmin::EventChannel interface.
- Erlang Module **CosNotifyChannelAdmin_EventChannelFactory** [page 45] - This module implements the OMG CosNotifyChannelAdmin::EventChannelFactory interface.
- Erlang Module **CosNotifyChannelAdmin_ProxyConsumer** [page 46] - This module implements the OMG CosNotifyChannelAdmin::ProxyConsumer interface.
- Erlang Module **CosNotifyChannelAdmin_ProxyPullConsumer** [page 48] - This module implements the OMG CosNotifyChannelAdmin::ProxyPullConsumer interface.
- Erlang Module **CosNotifyChannelAdmin_ProxyPushConsumer** [page 50] - This module implements the OMG CosNotifyChannelAdmin::ProxyPushConsumer interface.
- Erlang Module **CosNotifyChannelAdmin_ProxyPushSupplier** [page 52] - This module implements the OMG CosNotifyChannelAdmin::ProxyPushSupplier interface.
- Erlang Module **CosNotifyChannelAdmin_ProxySupplier** [page 55] - This module implements the OMG CosNotifyChannelAdmin::ProxySupplier interface.
The following functions are exported:

- 'EventReliability'() -> string()
  [page 33] Return the EventReliability QoS identifier
• 'BestEffort'() -> short()
  [page 33] Return the BestEffort QoS value
• 'Persistent'() -> short()
  [page 33] Return the Persistent QoS value
• 'ConnectionReliability'() -> string()
  [page 33] Return the ConnectionReliability QoS identifier
• 'Priority'() -> string()
  [page 33] Return the Priority QoS identifier
• 'LowestPriority'() -> short()
  [page 33] Return the LowestPriority QoS value
• 'HighestPriority'() -> short()
  [page 33] Return the HighestPriority QoS value
• 'DefaultPriority'() -> short()
  [page 33] Return the DefaultPriority QoS value
• 'StartTime'() -> string()
  [page 33] Return the StartTime QoS identifier
• 'StopTime'() -> string()
  [page 33] Return the StopTime QoS identifier
• 'Timeout'() -> string()
  [page 34] Return the Timeout QoS identifier
• 'OrderPolicy'() -> string()
  [page 34] Return the OrderPolicy QoS identifier
• 'AnyOrder'() -> short()
  [page 34] Return the AnyOrder QoS value
• 'FifoOrder'() -> short()
  [page 34] Return the FifoOrder QoS value
• 'PriorityOrder'() -> short()
  [page 34] Return the PriorityOrder QoS value
• 'DeadlineOrder'() -> short()
  [page 34] Return the DeadlineOrder QoS value
• 'DiscardPolicy'() -> string()
  [page 34] Return the DiscardPolicy QoS identifier
• 'LifoOrder'() -> short()
  [page 34] Return the LifoOrder QoS value
• 'RejectNewEvents'() -> short()
  [page 34] Return the RejectNewEvents QoS value
• 'MaximumBatchSize'() -> string()
  [page 34] Return the MaximumBatchSize QoS identifier
• 'PacingInterval'() -> string()
  [page 34] Return the PacingInterval QoS identifier
• 'StartTimeSupported'() -> string()
  [page 34] Return the StartTimeSupported QoS identifier
• 'StopTimeSupported'() -> string()
  [page 34] Return the StopTimeSupported QoS identifier
• 'MaxEventsPerConsumer'() -> string()
  [page 34] Return the MaxEventsPerConsumer QoS identifier
CosNotification Reference Manual

The following functions are exported:

- `get_admin(Object) -> AdminProperties`
  [page 36] Return a list of `AdminProperties` associated with the target object

- `set_admin(Object, AdminProperties) -> Reply`
  [page 36] Update the `AdminProperties` for the target object

CosNotification_QoSAdmin

The following functions are exported:

- `get qos(Object) -> Reply`
  [page 37] Return a list of name-value pairs which encapsulates the current QoS settings for the target object

- `set qos(Object, QoS) -> Reply`
  [page 37] Change the QoS settings for the target object

- `validate qos(Object, QoS) -> Reply`
  [page 38] Validate if the supplied QoS properties is valid for the target object

CosNotifyChannelAdmin_ConsumerAdmin

The following functions are exported:

- `get MyID(ConsumerAdmin) -> AdminID`
  [page 39] Return the target object’s Id

- `get MyChannel(ConsumerAdmin) -> Channel`
  [page 39] Return the ancestor channel

- `get MyOperator(ConsumerAdmin) -> OpType`
  [page 39] Return the filtering schema used by the target object

- `get priority filter(ConsumerAdmin) -> MappingFilter`
  [page 39] Return the associated priority `MappingFilter`

- `set priority filter(ConsumerAdmin, MappingFilter) -> ok`
  [page 40] Set the priority `MappingFilter`

- `get lifetime filter(ConsumerAdmin) -> MappingFilter`
  [page 40] Return the associated lifetime `MappingFilter`

- `set lifetime filter(ConsumerAdmin, MappingFilter) -> ok`
  [page 40] Set the lifetime `MappingFilter`

- `get pull suppliers(ConsumerAdmin) -> ProxyIDSeq`
  [page 40] Return a list of all associated pull supplier Id:s
- **get_push_suppliers(ConsumerAdmin) -> ProxyIDSeq**
  [page 40] Return a list of all associated push supplier Id:s
- **get_proxy_supplier(ConsumerAdmin, ProxyID) -> Reply**
  [page 40] Return the proxy supplier with matching Id
- **obtain_notification_pull_supplier(ConsumerAdmin, ConsumerType) -> Reply**
  [page 41] Create a supplier proxy
- **obtain_pull_supplier(ConsumerAdmin) -> Proxy**
  [page 41] Create a supplier proxy
- **obtain_notification_push_supplier(ConsumerAdmin, ConsumerType) -> Reply**
  [page 41] Create a supplier proxy
- **obtain_push_supplier(ConsumerAdmin) -> Proxy**
  [page 41] Create a supplier proxy
- **destroy(ConsumerAdmin) -> ok**
  [page 41] Terminate the target object and all its children

**CosNotifyChannelAdmin_EventChannel**

The following functions are exported:

- **get_MyFactory(Channel) -> ChannelFactory**
  [page 42] Return the factory object which created the target object
- **get_default_consumer_admin(Channel) -> ConsumerAdmin**
  [page 42] Return the default consumer admin associated with the target object
- **get_default_supplier_admin(Channel) -> SupplierAdmin**
  [page 42] Return the default supplier admin associated with the target object
- **get_default_filter_factory(Channel) -> FilterFactory**
  [page 42] Return the default filter factory associated with the target object
- **new_for_consumers(Channel, OpType) -> Return**
  [page 43] Create a new ConsumerAdmin object
- **for_consumers(Channel) -> ConsumerAdmin**
  [page 43] Create a new ConsumerAdmin object
- **new_for_suppliers(Channel, OpType) -> Return**
  [page 43] Create a new SupplierAdmin object
- **for_suppliers(Channel) -> SupplierAdmin**
  [page 43] Create a new SupplierAdmin object
- **get_consumeradmin(Channel, AdminID) -> ConsumerAdmin**
  [page 43] Return the ConsumerAdmin matching AdminID
- **get_supplieradmin(Channel, AdminID) -> SupplierAdmin**
  [page 44] Return the SupplierAdmin matching AdminID
- **get_all_consumeradmins(Channel) -> Reply**
  [page 44] Return a list of all ConsumerAdmins, currently active, Id:s
- **get_all_supplieradmins(Channel) -> Reply**
  [page 44] Return a list of all SupplierAdmins, currently active, Id:s
- **destroy(Channel) -> ok**
  [page 44] Terminate the channel and all its childrens
CosNotifyChannelAdmin_EventChannelFactory

The following functions are exported:

- `create_channel(ChannelFactory, InitialQoS, InitialAdmin) --> Return` [page 45] Create a new channel
- `get_all_channels(ChannelFactory) --> ChannelIDSeq` [page 45] Return all Id:s for channels, currently alive, created by the target object
- `get_event_channel(ChannelFactory, ChannelID) --> Return` [page 45] Return the channel object associated with the given Id

CosNotifyChannelAdmin_ProxyConsumer

The following functions are exported:

- `_get_MyType(ProxyConsumer) --> ProxyType` [page 46] Return the proxy type
- `_get_MyAdmin(ProxyConsumer) --> AdminObject` [page 46] return the associated Admin object
- `obtain_subscription_types(ProxyConsumer, ObtainInfoMode) --> EventTypeSeq` [page 46] Administer subscription types
- `validate_event_qos(ProxyConsumer, QoSProperties) --> Reply` [page 47] 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 48] Connect a supplier to the proxy
- `suspend_connection(ProxyPullConsumer) --> Reply` [page 48] Suspend the connection between the client and the proxy
- `resume_connection(ProxyPullConsumer) --> Reply` [page 48] Resume a previously suspended connection with the proxy
- `disconnect_pull_supplier(ProxyPullConsumer) --> ok` [page 49] Close the connection and terminate the proxy

CosNotifyChannelAdmin_ProxyPullSupplier

The following functions are exported:

- `connect_any_pull_consumer(ProxyPullSupplier, PullConsumer) --> Reply` [page 50] Connect a consumer to the proxy
- `pull(ProxyPullSupplier) --> Reply` [page 50] Pull an Any event from the proxy
- `try_pull(ProxyPullSupplier) --> Reply` [page 50] Try and pull an Any event from the proxy
- `disconnect_pull_supplier(ProxyPullSupplier) --> ok` [page 51] Close the connection and terminate the proxy
CosNotifyChannelAdmin_ProxyPushConsumer

The following functions are exported:

- `connect_any_push_supplier(ProxyPushConsumer, PushSupplier) -> Reply`
  [page 52] Connect a supplier to the proxy
- `push(ProxyPushConsumer, Event) -> Reply`
  [page 52] Push an Any event to the proxy
- `disconnect_push_consumer(ProxyPushConsumer) -> ok`
  [page 52] Close the connection and terminate the proxy

CosNotifyChannelAdmin_ProxyPushSupplier

The following functions are exported:

- `connect_any_push_consumer(ProxyPushSupplier, PushConsumer) -> Reply`
  [page 53] Connect a consumer to the proxy
- `suspend_connection(ProxyPushSupplier) -> Reply`
  [page 53] Suspend the connection between the proxy and the client
- `resume_connection(ProxyPushSupplier) -> Reply`
  [page 53] Resume a previously suspended connection with the proxy
- `disconnect_push_supplier(ProxyPushSupplier) -> ok`
  [page 54] Close the connection and terminate the proxy

CosNotifyChannelAdmin_ProxySupplier

The following functions are exported:

- `_get_MyType(ProxySupplier) -> ProxyType`
  [page 55] Return the proxy type
- `_get_MyAdmin(ProxySupplier) -> AdminObject`
  [page 55] Return the target object’s associated Admin object
- `_get_priority_filter(ProxySupplier) -> MappingFilter`
  [page 55] Return the target object’s associated priority MappingFilter
- `_set_priority_filter(ProxySupplier, MappingFilter) -> ok`
  [page 55] Set the target object’s associated priority MappingFilter
- `_get_lifetime_filter(ProxySupplier) -> MappingFilter`
  [page 56] Return the target object’s associated lifetime MappingFilter
- `_set_lifetime_filter(ProxySupplier, MappingFilter) -> ok`
  [page 56] Set the target object’s associated lifetime MappingFilter
- `obtain_offered_types(ProxySupplier, ObtainInfoMode) -> EventTypeSeq`
  [page 56] Administer the type of events the proxy supplies
- `validate_event_qos(ProxySupplier, QoSProperties) -> Reply`
  [page 56] Check if the QoS properties can be set
CosNotifyChannelAdmin_SequenceProxyPullConsumer

The following functions are exported:

- `connect_sequence_pull_supplier(SequenceProxyPullConsumer, PullSupplier) -> Reply`
  - [page 58] Connect a supplier to the proxy
- `suspend_connection(SequenceProxyPullConsumer) -> Reply`
  - [page 58] Suspend the connection with the proxy
- `resume_connection(SequenceProxyPullConsumer) -> Reply`
  - [page 58] Resume a previously suspended connection with the proxy
- `disconnect_sequence_pull_consumer(SequenceProxyPullConsumer) -> ok`
  - [page 59] Close connection and terminate the proxy

CosNotifyChannelAdmin_SequenceProxyPullSupplier

The following functions are exported:

- `connect_sequence_pull_consumer(SequenceProxyPullSupplier, PullConsumer) -> Reply`
  - [page 60] Connect a consumer to the proxy
- `pull_structured_events(SequenceProxyPullSupplier, MaxEvents) -> Reply`
  - [page 60] Pull structured events from the proxy
- `try_pull_structured_events(SequenceProxyPullSupplier, MaxEvents) -> Reply`
  - [page 61] Try to pull structured events from the proxy
- `disconnect_sequence_pull_supplier(SequenceProxyPullSupplier) -> ok`
  - [page 61] Close the connection and terminate the proxy

CosNotifyChannelAdmin_SupplierAdmin

The following functions are exported:

- `_get_MyID(SupplierAdmin) -> AdminID`
  - [page 62] Return the objects Id
- `_get_MyChannel(SupplierAdmin) -> Channel`
  - [page 62] Return the objects associated channel
- `_get_MyOperator(SupplierAdmin) -> OpType`
  - [page 62] Return the filter scheme
- `_get_pull_consumers(SupplierAdmin) -> ProxyIDSeq`
  - [page 62] Return all associated pull consumers Id:s
- `_get_push_consumers(SupplierAdmin) -> ProxyIDSeq`
  - [page 63] Return all associated push consumers Id:s
- `get_proxy_consumer(SupplierAdmin, ProxyID) -> Reply`
  - [page 63] Return the Proxy which corresponds to the given Id
- `obtain_notification_pull_consumer(SupplierAdmin, SupplierType) -> Reply`
  - [page 63] Create a new proxy
CosNotifyChannelAdmin_SequenceProxyPushConsumer

The following functions are exported:

- `connect_sequence_push_supplier(SequenceProxyPushConsumer, PushSupplier) -> Reply`
  [page 65] Connect a supplier to the proxy
- `push_structured_events(SequenceProxyPushConsumer, EventBatch) -> Reply`
  [page 65] Push a structured event to the proxy
- `disconnect_sequence_push_consumer(SequenceProxyPushConsumer) -> ok`
  [page 66] Close connection and terminate the proxy

CosNotifyChannelAdmin_SequenceProxyPushSupplier

The following functions are exported:

- `connect_sequence_push_consumer(SequenceProxyPushSupplier, PushConsumer) -> Reply`
  [page 67] Connect a consumer to the proxy
- `suspend_connection(SequenceProxyPushSupplier) -> Reply`
  [page 67] Suspend the connection between the client and the target object
- `resume_connection(SequenceProxyPushSupplier) -> Reply`
  [page 67] Resume a previously suspended connection with the proxy
- `disconnect_sequence_push_supplier(SequenceProxyPushSupplier) -> ok`
  [page 68] Close the connection and terminate the proxy

CosNotifyChannelAdmin_StructuredProxyPullConsumer

The following functions are exported:

- `connect_structured_pull_supplier(StructuredProxyPullConsumer, PullSupplier) -> Reply`
  [page 69] Connect a supplier to the proxy
- `suspend_connection(StructuredProxyPullConsumer) -> Reply`
  [page 69] Suspend the connection between the target object and its client
- `resume_connection(StructuredProxyPullConsumer) -> Reply`
  [page 69] Resume a previously suspended connection with the proxy
• disconnect_structured_pull_consumer(StructuredProxyPullConsumer) -> ok
  [page 70] Close the connection and terminate the proxy

**CosNotifyChannelAdmin_StructuredProxyPullSupplier**

The following functions are exported:

• connect_structured_pull_consumer(StructuredProxyPullSupplier, PullConsumer) -> Reply
  [page 71] Connect a consumer to the proxy
• pull_structured_event(StructuredProxyPullSupplier) -> Reply
  [page 71] Pull a structured event from the proxy
• try_pull_structured_event(StructuredProxyPullSupplier) -> Reply
  [page 72] Try to pull a structured event from the proxy
• disconnect_structured_pull_supplier(StructuredProxyPullSupplier) -> ok
  [page 72] Close connection and terminate the proxy

**CosNotifyChannelAdmin_StructuredProxyPushConsumer**

The following functions are exported:

• connect_structured_push_supplier(StructuredProxyPushConsumer, PushSupplier) -> Reply
  [page 73] Connect a supplier to the proxy
• push_structured_event(StructuredProxyPushConsumer, StructuredEvent) -> Reply
  [page 73] Push a structured event to the proxy
• disconnect_structured_push_consumer(StructuredProxyPushConsumer) -> ok
  [page 74] Close the connection and terminate the proxy

**CosNotifyChannelAdmin_StructuredProxyPushSupplier**

The following functions are exported:

• connect_structured_push_consumer(StructuredProxyPushSupplier, PushConsumer) -> Reply
  [page 75] Connect a consumer to the proxy
• suspend_connection(StructuredProxyPushSupplier) -> Reply
  [page 75] Suspend the connection with the target object
• resume_connection(StructuredProxyPushSupplier) -> Reply
  [page 75] Resume a previously suspended connection
• disconnect_structured_push_supplier(StructuredProxyPushSupplier) -> ok
  [page 76] Close the connection and terminate the target object
CosNotifyComm.NotifyPublish

The following functions are exported:

- **offer_change**(Object, Added, Removed) \to Reply
  [page 77] Inform the target object which type of events the supplier will deliver

CosNotifyComm.NotifySubscribe

The following functions are exported:

- **subscription_change**(Object, Added, Removed) \to Reply
  [page 78] 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) \to Grammar
  [page 79] Return which type of Grammar the Filter uses
- **add_constraints**(Filter, ConstraintExpSeq) \to Reply
  [page 79] Add new constraints to the filter
- **modify_constraints**(Filter, ConstraintIDSeq, ConstraintInfoSeq) \to Reply
  [page 79] Modify existing constraints
- **get_constraints**(Filter, ConstraintIDSeq) \to Reply
  [page 80] Return all constraints which match the supplied Ids
- **get_all_constraints**(Filter) \to ConstraintInfoSeq
  [page 80] Return all constraints associated with the target object
- **remove_all_constraints**(Filter) \to ok
  [page 80] Remove all constraints associated with the target object
- **destroy**(Filter) \to ok
  [page 81] Terminate the target object
- **match**(Filter, Event) \to Reply
  [page 81] Match the Any event if it satisfies at least one constraint
- **match_structured**(Filter, Event) \to Reply
  [page 81] Match the structured event if it satisfies at least one constraint
- **attach_callback**(Filter, NotifySubscribe) \to CallbackID
  [page 81] Connect NotifySubscribe object, which should be informed when the target object’s constraints are updated
- **detach_callback**(Filter, CallbackID) \to Reply
  [page 81] Disconnect the NotifySubscribe object with the given Id
- **get_callbacks**(Filter) \to CallbackIDSeq
  [page 82] Return all NotifySubscribe Id’s associated with the target object
CosNotifyFilter_FilterAdmin

The following functions are exported:

- `add_filter(Object, Filter) -> FilterID`
  [page 83] Add a new filter to the target object
- `remove_filter(Object, FilterID) -> ok`
  [page 83] Remove a filter associated with the target object
- `get_filter(Object, FilterID) -> Reply`
  [page 83] Return the filter with the given Id
- `get_all_filters(Object) -> FilterIDSeq`
  [page 83] Return a list of all filter Id:s associated with the target object
- `remove_all_filters(Object) -> ok`
  [page 84] Remove all filters from the target object

CosNotifyFilter_FilterFactory

The following functions are exported:

- `create_filter(FilterFactory, Grammar) -> Reply`
  [page 85] Create a Filter object
- `create_mapping_filter(FilterFactory, Grammar) -> Reply`
  [page 85] Create a MappingFilter object

CosNotifyFilter_MappingFilter

The following functions are exported:

- `get_constraint_grammar(MappingFilter) -> Grammar`
  [page 86] Return which type of Grammar the MappingFilter uses
- `get_value_type(MappingFilter) -> CORBA::TypeCode`
  [page 86] Return the CORBA::TypeCode of the default value associated with the target object
- `get_default_value(MappingFilter) -> #any`
  [page 86] Return the #any{} default value associated with the target object
- `add_mapping_constraints(MappingFilter, MappingConstraintPairSeq) -> Reply`
  [page 86] Add new mapping constraints
- `modify_constraints(MappingFilter, ConstraintIDSeq, MappingConstraintInfoSeq) -> Reply`
  [page 87] Modify the constraints associated with the target object
- `get_mapping_constraints(MappingFilter, ConstraintIDSeq) -> Reply`
  [page 88] Return the target object’s associated constraints with given ID:s
- `get_all_mapping_constraints(MappingFilter) -> MappingConstraintInfoSeq`
  [page 88] Return the target object’s all associated constraints
- `remove_all_mapping_constraints(MappingFilter) -> ok`
  [page 88] Remove all constraints associated with the target object
**cosNotification Application**

The following functions are exported:

- `install()` -> Return
  [page 90] Install the cosNotification application

- `install(Seconds)` -> Return
  [page 90] Install the cosNotification application

- `install_event()` -> Return
  [page 90] Install the necessary cosEvent interfaces

- `install_event(Seconds)` -> Return
  [page 90] Install the necessary cosEvent interfaces

- `uninstall()` -> Return
  [page 90] Uninstall the cosNotification application

- `uninstall(Seconds)` -> Return
  [page 91] Uninstall the cosNotification application

- `uninstall_event()` -> Return
  [page 91] Uninstall the inheritted cosEvent interfaces

- `uninstall_event(Seconds)` -> Return
  [page 91] Uninstall the inheritted cosEvent interfaces

- `start()` -> Return
  [page 91] Start the cosNotification application

- `stop()` -> Return
  [page 91] Stop the cosNotification application

- `start_global_factory()` -> ChannelFactory
  [page 91] Start a global channel factory as default

- `start_global_factory(Options)` -> ChannelFactory
  [page 91] Start a global channel factory with options

- `start_factory()` -> ChannelFactory
  [page 92] Start a channel factory as default

- `start_factory(Options)` -> ChannelFactory
  [page 92] Start a channel factory with options

- `stop_factory(ChannelFactory)` -> Reply
  [page 92] Terminate the target object

- `start_filter_factory()` -> FilterFactory
  [page 93] Start a filter factory

- `stop_filter_factory(FilterFactory)` -> Reply
  [page 93] Terminate the target object

- `destroy(MappingFilter)` -> ok
  [page 89] Terminate the target object

- `match(MappingFilter, Event)` -> Reply
  [page 89] Evaluate the given Any event with the Filter's constraints

- `match_structured(MappingFilter, Event)` -> Reply
  [page 89] Evaluate the given structured event with the Filter's constraints
• `create_structured_event(Domain, Type, Event, VariableHeader, FilterableBody, BodyRemainder)` -> Reply
  [page 93] Create a structured event

• `type_check()` -> Reply
  [page 93] Return the value of the configuration parameter `type_check`
CosNotification

Erlang Module

To get access to all definitions include necessary hrl files by using:
-include_lib("cosNotification/include/*.hrl").

Exports

'EventReliability'() -> string()
  This function returns the EventReliability QoS identifier

'BestEffort'() -> short()
  This function returns the BestEffort QoS value.

'Persistent'() -> short()
  This function returns the Persistent QoS value.

'ConnectionReliability'() -> string()
  This function returns the ConnectionReliability QoS identifier.

'Priority'() -> string()
  This function returns the Priority QoS identifier.

'LowestPriority'() -> short()
  This function returns the LowestPriority QoS value.

'HighestPriority'() -> short()
  This function returns the HighestPriority QoS value.

'DefaultPriority'() -> short()
  This function returns the DefaultPriority QoS value.

'StartTime'() -> string()
  This function returns the StartTime QoS identifier.

'StopTime'() -> string()
This function returns the StopTime QoS identifier.

'Timeout'() -> string()
This function returns the Timeout QoS identifier.

'OrderPolicy'() -> string()
This function returns the OrderPolicy QoS identifier.

'AnyOrder'() -> short()
This function returns the AnyOrder QoS value.

'FifoOrder'() -> short()
This function returns the FifoOrder QoS value.

'PriorityOrder'() -> short()
This function returns the PriorityOrder QoS value.

'DeadlineOrder'() -> short()
This function returns the DeadlineOrder QoS value.

'DiscardPolicy'() -> string()
This function returns the DiscardPolicy QoS identifier.

'LifoOrder'() -> short()
This function returns the LifoOrder QoS value.

'RejectNewEvents'() -> short()
This function returns the RejectNewEvents QoS value.

'MaximumBatchSize'() -> string()
This function returns the MaximumBatchSize QoS identifier.

'PacingInterval'() -> string()
This function returns the PacingInterval QoS identifier.

'StartTimeSupported'() -> string()
This function returns the StartTimeSupported QoS identifier.

'StopTimeSupported'() -> string()
This function returns the StopTimeSupported QoS identifier.

'MaxEventsPerConsumer'() -> string()
This function returns the MaxEventsPerConsumer QoS identifier.

'MaxQueueLength'() -> string()
This function returns the MaxQueueLength Admin identifier.

'MaxConsumers'() -> string()
This function returns the MaxConsumers Admin identifier.

'MaxSuppliers'() -> string()
This function returns the MaxSuppliers Admin identifier.
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** = 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** = string()
  - **value** = #any
- **Reply** = ok | {**EXCEPTION**, **UnSupportedQoS**} (**qos_err**)
- **qos_err** = PropertyErrorSeq
- **PropertyErrorSeq** = [**PropertyError**]
- **PropertyError** = #CosNotification.PropertyError
  - **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 = #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 37]
- CosNotifyComm_NotifySubscribe [page 78]
- CosNotifyFilter_FilterAdmin [page 83]

Exports

\texttt{\_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.

\texttt{\_get\_MyChannel(ConsumerAdmin) -> Channel}
Types:
- ConsumerAdmin = \#objref
- Channel = \#objref

The creating channel's reference is maintained by this readonly attribute.

\texttt{\_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.

\texttt{\_get\_priority\_filter(ConsumerAdmin) -> MappingFilter}
Types:
- ConsumerAdmin = MappingFilter = \#objref

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

\_set\_priority\_filter(ConsumerAdmin, MappingFilter) \rightarrow ok

Types:
- ConsumerAdmin = MappingFilter = #objref

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

\_get\_lifetime\_filter(ConsumerAdmin) \rightarrow 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) \rightarrow ok

Types:
- ConsumerAdmin = MappingFilter = #objref

This operation associate a lifetime MappingFilter with the target object.

\_get\_pull\_suppliers(ConsumerAdmin) \rightarrow ProxyIDSeq

Types:
- ConsumerAdmin = #objref
- ProxyID Seq = [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) \rightarrow ProxyIDSeq

Types:
- ConsumerAdmin = #objref
- ProxyID Seq = [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) \rightarrow Reply

Types:
- ConsumerAdmin = #objref
- ProxyID = long()

- Reply = Proxy | {'EXCEPTION', 'CosNotifyChannelAdmin_Protocol_NotSupported'}
- Proxy = #objref
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 37]
- CosNotification_AdminPropertiesAdmin [page 36]

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:
- Channel = #objref
- FilterFactory = #objref
The default FilterFactory associated with the target channel is maintained by this readonly attribute.

```
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.

```
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 filterOp. The target object's Quality of Service properties will be inherited by the returned ConsumerAdmin.

```
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.

```
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 filterOp.

```
get_consumeradmin(Channel, AdminID) -> ConsumerAdmin

Types:
```

---

cosNotification Application 43
EventChannel cosNotification Reference Manual

- Channel = #objref
- AdminID = long()
- ConsumerAdmin = #objref | {'EXCEPTION',
  #CosNotifyChannelAdmin_AdmInNotFound'}

If the given 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(Channel, AdminID) -> SupplierAdmin

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

Equal to the operation get_consumeradmin/2 but a reference to a SupplierAdmin is returned.

get_all_consumeradmins(Channel) -> Reply

Types:
- Channel = #objref
- Reply = [AdminID]
- AdminID = 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(Channel) -> Reply

Types:
- Channel = #objref
- Reply = [AdminID]
- AdminID = long()

Equal to the operation get_all_consumeradmins/1 but returns a list of all SupplierAdmin object ID's.

destroy(Channel) -> ok

Types:
- Channel = #objref

The 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.
If no QoS- and/or Admin-properties are supplied (i.e. empty list), the default settings
will be used. For more information, see the User’s Guide.

get_all_channels(ChannelFactory) -> ChannelIDSeq
Types:
  • ChannelFactory = #objref
  • ChannelID Seq = [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 37]
- CosNotifyFilter_FilterAdmin [page 83]

Exports

_get_MyType(ProxyConsumer) -> ProxyType

Types:
- ProxyConsumer = #objref
- ProxyType = 'PUSH_ANY' | 'PULL_ANY' | 'PUSH_STRUCTURED' | 'PULL_STRUCTURED' | 'PULL_SEQUENCE' | 'PULL_STRUCTURED'

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 77]
- QoSAdmin [page 37]
- CosNotifyFilter_FilterAdmin [page 83]
- ProxyConsumer [page 46]

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 = #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 has 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 informs its administrative parent.
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 78]
- CosNotification_QoSAdmin [page 37]
- CosNotifyFilter_FilterAdmin [page 83]
- CosNotifyChannelAdmin_ProxySupplier [page 55]

Exports

connect_any_pull_consumer(ProxyPullSupplier, PullConsumer) -> Reply
Types:
- ProxyPullSupplier = #objref
- PullConsumer = #objref
- Reply = ok | {'EXCEPTIO N', #CosEventChannelAdmin_AlreadyConnected'}

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 | {'EXCEPTIO N', #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} | {'EXCEPTIO N', #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.

disconnect_pull_supplier(ProxyPullSupplier) -> ok

Types:
• 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 77]
- CosNotification_QoSAdmin [page 37]
- CosNotifyFilter_FilterAdmin [page 83]
- CosNotifyChannelAdmin_ProxyConsumer [page 46]

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:
- CosNotifyComm_NotifySubscribe [page 78]
- CosNotification_QoSAdmin [page 37]
- CosNotifyFilter_FilterAdmin [page 83]
- CosNotifyChannelAdmin_ProxySupplier [page 55]

Exports

connect_any_push_consumer(ProxyPushSupplier, PushConsumer) -> Reply
Types:
- ProxyPushSupplier = #objref
- PushConsumer = #objref
- Reply = ok | {'EXCEPTION', '#CosNotifyChannelAdmin_AlreadyConnected'} | {'EXCEPTION', '#CosNotifyChannelAdmin_TypeError'}

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', #CosNotifyChannelAdmin_ConnectionAlreadyInactive'} | {'EXCEPTION', #CosNotifyChannelAdmin_NotConnected'}

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
• Reply = ok | {'EXCEPTION',
  #CosNotifyChannelAdmin_ConnectionAlreadyInactive'} | {'EXCEPTION',
  #CosNotifyChannelAdmin_NotConnected'}

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 37]
- CosNotifyFilter_FilterAdmin [page 83]

Exports

_get_MyType(ProxySupplier) -> ProxyType
Types:
- ProxySupplier = #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(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.

```plaintext
_get_lifetime_filter(ProxySupplier) -> MappingFilter
```

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

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

```plaintext
_set_lifetime_filter(ProxySupplier, MappingFilter) -> ok
```

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

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

```plaintext
obtain_offered_types(ProxySupplier, ObtainInfoMode) -> 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.

```plaintext
validate_event_qos(ProxySupplier, QoSProperties) -> Reply
```

Types:
- ProxySupplier = #objref
- QoSProperties = [ QoSProperty ]
- QoSProperty = #'CosNotificationProperty'(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.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 77]
- CosNotification_QoSAdmin [page 37]
- CosNotifyFilter_FilterAdmin [page 83]
- CosNotifyChannelAdmin.ProxyConsumer [page 46]

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 | {'EXCEPTION',
  #CosNotifyChannelAdmin_ConnectionAlreadyInactive'} | {'EXCEPTION',
  #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.
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 78]
- CosNotification_QoSAdmin [page 37]
- CosNotifyFilter_FilterAdmin [page 83]
- CosNotifyChannelAdmin_ProxySupplier [page 55]

Exports

connect_sequence_pull_consumer(SequenceProxyPullSupplier, PullConsumer) -> Reply
Types:
  • SequenceProxyPullSupplier = #objref
  • PullConsumer = #objref
  • Reply = ok | {'EXCEPTION', #CosEventChannelAdmin_AreadyConnected'}
This operation connects a PullConsumer to the target object. If a connection already exists an exception is raised.

pull_structured_events(SequenceProxyPullSupplier, MaxEvents) -> Reply
Types:
  • SequenceProxyPullSupplier = #objref
  • 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
  • EventHeader = #CosNotification_EventHeader{'fixed_header', variable_header}
  • fixed_header = FixedEventHeader
variable_header = OptionalHeaderFields
FixedEventHeader = #CosNotification_FixedEventHeader'\{event_type, event_name\}
eventType = EventType
eventName = string()
EventType = #CosNotification_EventType'\{domain_name, type_name\}
domain_name = type_name = string()
OptionalHeaderFields = [ #CosNotification_Property'\{name, value\}]

A client use this operation to pull next event sequence of maximum length MaxEvents. This operation is blocking and will not reply until the requested amount of events can be delivered or the QoS property PacingInterval is reached. For more information see the User’s Guide.

try_pull_structured_events(SequenceProxyPullSupplier, MaxEvents) -> Reply

Types:
SequenceProxyPullSupplier = #objref
MaxEvents = long()
Reply = {EventBatch, HasEvent} | {EXCEPTION', #CosEventChannelAdmin_Disconnected'}
HasEvent = boolean()
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 = OptionalHeaderFields
FixedEventHeader = #CosNotification_FixedEventHeader'\{event_type, event_name\}
eventType = EventType
eventName = string()
EventType = #CosNotification_EventType'\{domain_name, type_name\}
domain_name = type_name = string()
OptionalHeaderFields = [ #CosNotification_Property'\{name, value\}]

This operation pulls an event sequence of the maximum length MaxEvents, but do not block if the target object have no events to forward. The outparameter, HasEvent is true if the sequence contain any events.

disconnect_sequence_pull_supplier(SequenceProxyPullSupplier) -> ok

Types:
SequenceProxyPullSupplier = #objref

This operation cause 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_QoSAdmin [page 37]
- CosNotifyComm_NotifyPublish [page 78]
- CosNotifyFilter_FilterAdmin [page 83]

Exports

-get_MyID(SupplierAdmin) -> AdminID
Types:
- SupplierAdmin = #objcref
- 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_MyChannel(SupplierAdmin) -> Channel
Types:
- SupplierAdmin = #objcref
- Channel = #objcref

The creating channel’s reference is maintained by this readonly attribute.

-get_MyOperator(SupplierAdmin) -> OpType
Types:
- SupplierAdmin = #objcref
- OpType = ‘AND_OP’ | ‘OR_OP’

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 = #objcref
- 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 PullProxy 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_push_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.

`obtain_push_consumer(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.NotifyPublish [page 77]
- CosNotification.QoSAdmin [page 37]
- CosNotifyFilter.FilterAdmin [page 83]
- CosNotifyChannelAdmin.ProxyConsumer [page 46]

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 = OptionalHeaderFields
- FixedEventHeader = #CosNotification.FixedEventHeader {event_type, event_name}
- event_type = EventType
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.
```
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 78]
- CosNotification.QoSAdmin [page 37]
- CosNotifyFilter.FilterAdmin [page 83]
- CosNotifyChannelAdmin.ProxySupplier [page 55]

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
• Reply = ok | {'EXCEPTION',
  '#CosNotifyChannelAdmin_NotConnected'}
| {'EXCEPTION',
  '#CosNotifyChannelAdmin_ConnectionAlreadyInactive'}

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.

```haskell
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:

- CosNotifyComm_NotifyPublish [page 77]
- CosNotification_QoSAdmin [page 37]
- CosNotifyFilter_FilterAdmin [page 83]
- CosNotifyChannelAdmin_ProxyConsumer [page 46]

Exports

connect_structured_pull_supplier(StructuredProxyPullConsumer, PullSupplier) -> Reply
Types:
- StructuredProxyPullConsumer = #objref
- PullSupplier = #objref
- Reply = ok | {'EXCEPTION', 'CosNotifyChannelAdmin_AlreadyConnected'} | {'EXCEPTION', 'CosNotifyChannelAdmin_TypeError'}

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

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

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.

\[\text{disconnect}_\text{structured_pull_consumer}(\text{StructuredProxyPullConsumer}) \rightarrow \text{ok}\]

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

This operation cause the target object to close the connection and terminate.
CosN otifyCh annelAd m in _StructuredProxyPullSup pli er

Erlang Module

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

This module also exports the functions described in:
- CosNotifyComm_NotifySubscribe [page 78]
- CosN otification_QoSAdmin [page 37]
- CosNotifyFilter_FilterAdmin [page 83]
- CosN otifyChannelA dmin_ProxySupplier [page 55]

Exports

connect_structured_pull_consumer(StructuredProxyPullSupplier, PullConsumer) -> Reply
Types:
- StructuredProxyPullSupplier = #objref
- PullConsumer = #objref
- Reply = ok | '{EXCEPTIO N', '#CosEventChannelAdmin_A lreadyConnected'[]}
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 | '{EXCEPTIO N',
  '#CosEventChannelAdmin_Disconnected'[]}
- StructuredEvent = '#CosNotification_StructuredEvent'{'header, filterable_data, remainder_of_body'}
- header = EventHeader
- filterable_data = [{#CosN otification_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}
- `event_type = EventType`
- `event_name = string()`
- `EventType = #CosNotification_EventType’{domain_name, type_name}`
- `domain_name = type_name = string()`
- `OptionalHeaderFields = [ #CosNotification_Property’{name, value}]`

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

\[
\text{try\_pull\_structured\_event(StructuredProxyPullSupplier) -> Reply}
\]

Types:
- `StructuredProxyPullSupplier = #objref`
- `Reply = {StructuredEvent, HasEvent} | {EXCEPTION', #CosEventChannelAdmin_Disconnected'}`
  - `HasEvent = boolean()`
- `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}`
  - `event\_type = EventType`
  - `event\_name = string()`
- `EventType = #CosNotification_EventType’{domain\_name, type\_name}`
  - `domain\_name = type\_name = string()`
  - `OptionalHeaderFields = [ #CosNotification\_Property’{name, 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.

\[
\text{disconnect\_structured\_pull\_supplier(StructuredProxyPullSupplier) -> ok}
\]

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 77]
- CosNotification_qosAdmin [page 37]
- CosNotifyFilter_FilterAdmin [page 83]
- CosNotifyChannelAdmin_ProxyConsumer [page 46]

Exports

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.

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}
- event_type = EventType
- event_name = string()}
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 78]
- CosNotification_qosAdmin [page 37]
- CosNotifyFilter_FilterAdmin [page 83]
- CosNotifyChannelAdmin_ProxySupplier [page 55]

Exports

**connect_structured_push_consumer(StructuredProxyPushSupplier, PushConsumer) -> Reply**

Types:
- StructuredProxyPushSupplier = #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_event is not supported by the client object an exception is raised.

**suspend_connection(StructuredProxyPushSupplier) -> Reply**

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

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 = #objref
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.

\[
\text{disconnect} \_\text{structured}\_\text{push}\_\text{supplier}(\text{StructuredProxyPushSupplier}) \to \text{ok}
\]

Types:
- \text{StructuredProxyPushSupplier} = \#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

\[\text{offer\_change}(\text{Object}, \text{Added}, \text{Removed}) \rightarrow \text{Reply}\]

Types:
- \text{Object} = \#\text{objref}
- \text{Added} = \text{Removed} = \text{EventTypeSeq}
- \text{EventTypeSeq} = [\text{type}]
- \text{Reply} = \text{ok} \mid \{\text{EXCEPTION}, \text{CosNotifyComm}\_Invalid\text{EventType}(\text{type})\}
- \text{type} = \#\text{CosNotification\_EventType}\(\text{domain\_name}, \text{type\_name}\)
- \text{domain\_name} = \text{type\_name} = \text{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) -> 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}}
  • constr = ConstraintExp
  • ConstraintInfoSeq = [ConstraintInfo]
  • ConstraintInfo = #'CosNotifyFilter.ConstraintInfo'(constraint_expression, constraint_id)
    • constraint_expression = ConstraintExp
    • constraint_id = long()
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]
  • ConstraintID = long()
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', #CosNotifyFilter_ConstraintNotFound{id}}
  - ConstraintInfoSeq = [ConstraintInfo]
  - ConstraintInfo = #CosNotifyFilter_ConstraintInfo{constraint_expression, constraint_id}
  - constraint_expression = ConstraintExp
  - constraint_id = 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 = #CosNotifyFilter_ConstraintInfo{constraint_expression, constraint_id}
  - constraint_expression = ConstraintExp
  - constraint_id = id = 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 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

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.

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.

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.

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_allFilters(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{} default 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.

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.

destroy(MappingFilter) -> ok

Types:
- MappingFilter = #objref
This operation terminates the target object. Remember to remove this Filter from the objects it have been associated with.

match(MappingFilter, Event) -> Reply

Types:
- MappingFilter = #objref
- Event = #any
- Reply = {boolean(), #any} | {'EXCEPTION', #CosNotifyFilterUnsupportedFilterableData{}]
This operation evaluates 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.

match_structured(MappingFilter, Event) -> Reply

Types:
- MappingFilter = #objref
- Event = #CosNotification_StructuredEvent{}
- Reply = {boolean(), #any} | {'EXCEPTION', #CosNotifyFilterUnsupportedFilterableData{}]
Similar to 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, which may only be used if it is impossible to upgrade to cosEvent-2.0 or later, installs the necessary cosEvent interfaces. If cosEvent-2.0 is available, use cosEventApp:install() instead.

install_event(Seconds) -> Return
  Types:
  • Return = ok | {'EXCEPTION', E}
  This operation, which may only be used if it is impossible to upgrade to cosEvent-2.0 or later, installs the necessary cosEvent interfaces using Seconds delay between each block of IFR-registrations. If cosEvent-2.0 is available, use cosEventApp:install() instead.

uninstall() -> Return
  Types:
uninstall(Seconds) -> Return
Types:
  - Return = ok | {'EXCEPTI0N', 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 | {'EXCEPTI0N', E}
This operation uninstalls the inheritted cosEvent interfaces. If cosEvent is in use this function may not be used. This function may only be used if cosNotificationApp:install_event/1/2 was used. If not, use cosEventApp:uninstall() instead.

uninstall_event(Seconds) -> Return
Types:
  - Return = ok | {'EXCEPTI0N', E}
This operation uninstalls the inheritted cosEvent interfaces, using Seconds delay between each block of IFR-unregistrations. If cosEvent is in use this function delay be used. This function may only be used if cosNotificationApp:install_event/1/2 was used. If not, use cosEventApp:uninstall() instead.

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 45] should be used for a multi-node Orber. The Factory is used to create a new channel [page 42].

start_global_factory(Options) -> ChannelFactory
Types:
- Options = [Option]
- Option = {pullInterval, Seconds} | {filterOp, OperationType} | {gcTime, Seconds} | {gcLimit, Amount} | {timeService, #objref}
- ChannelFactory = #objref

This operation creates an Event Channel Factory [page 45] and should be used for a multi-node Orber. The Factory is used to create a new channel [page 42].

- {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.

\[\text{start} \text{factory}() \rightarrow \text{ChannelFactory}\]

Types:
- ChannelFactory = #objref

This operation creates an Event Channel Factory [page 45]. The Factory is used to create a new channel [page 42].

\[\text{start} \text{factory}(\text{Options}) \rightarrow \text{ChannelFactory}\]

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

This operation creates an Event Channel Factory [page 45]. The Factory is used to create a new channel [page 42].

\[\text{stop} \text{factory}(\text{ChannelFactory}) \rightarrow \text{Reply}\]

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

This operation stop the target channel factory.
start_filter_factory() -> FilterFactory

Types:
• FilterFactory = #objref

This operation creates a Filter Factory [page 85]. The Factory is used to create a new
Filter's [page 79] and MappingFilter's [page 86].

stop_filter_factory(FilterFactory) -> Reply

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

This operation stops the target filter factory.

create_structured_event(Domain, Type, Event, VariableHeader, FilterableBody,
BodyRemainder) -> Reply

Types:
• Domain = string()
• Type = string()
• Event = string()
• VariableHeader = [CosNotification::Property]
• FilterableBody = [CosNotification::Property]
• BodyRemainder = #any data-type
• Reply = CosNotification::StructuredEvent | {'EXCEPTION', E}

An easy way to create a structured event is to use this function. Simple typechecks are
performed and if one of the arguments is not correct a ‘BAD_PARAM’ exception is
thrown.

type_check() -> Reply

Types:
• Reply = true | false

This operation returns the value of the configuration parameter type_check.
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 Global Configuration Parameters ........................................... 3
1.2 Table 1: Type & Operator Examples ...................................... 11
1.3 Table 2: Fixed Header Constraint Examples ............................ 12
1.4 Table 1: Supported QoS Settings .......................................... 13
1.5 Table 2: Supported Admin Properties .................................... 15
List of Tables
Index of Modules and Functions

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

'AnyOrder'/0
  CosNotification , 34

'BestEffort'/0
  CosNotification , 33

'ConnectionReliability'/0
  CosNotification , 33

'DeadlineOrder'/0
  CosNotification , 34

'DefaultPriority'/0
  CosNotification , 33

'DiscardPolicy'/0
  CosNotification , 34

'EventReliability'/0
  CosNotification , 33

'FifoOrder'/0
  CosNotification , 34

'HighestPriority'/0
  CosNotification , 33

'LifoOrder'/0
  CosNotification , 34

'LowestPriority'/0
  CosNotification , 33

'MaxConsumers'/0
  CosNotification , 35

'MaxEventsPerConsumer'/0
  CosNotification , 34

'MaxQueueLength'/0
  CosNotification , 35

'MaxSuppliers'/0
  CosNotification , 35

'MaximumBatchSize'/0
  CosNotification , 34

'OrderPolicy'/0
  CosNotification , 34

'PacingInterval'/0
  CosNotification , 34

'Persistent'/0
  CosNotification , 33

'Priority'/0
  CosNotification , 33

'PriorityOrder'/0
  CosNotification , 34

'RejectNewEvents'/0
  CosNotification , 34

'StartTime'/0
  CosNotification , 33

'StartTimeSupported'/0
  CosNotification , 34

'StopTime'/0
  CosNotification , 33

'StopTimeSupported'/0
  CosNotification , 34

'Timeout'/0
  CosNotification , 34

_get_MyAdmin/1
  CosNotifyChannelAdmin_ProxyConsumer , 46
  CosNotifyChannelAdmin_ProxySupplier , 55

_get_MyChannel/1
  CosNotifyChannelAdmin_ConsumerAdmin , 39
  CosNotifyChannelAdmin_SupplierAdmin , 62

_get_MyFactory/1
  CosNotifyChannelAdmin_EventChannel , 42

cosNotification Application
<table>
<thead>
<tr>
<th>Function</th>
<th>Module/Class</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>_get_MyID/1</td>
<td>CosNotifyChannelAdmin,  min_ConsumerAdmin, 39</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>CosNotifyChannelAdmin,  SupplierAdmin, 62</td>
<td></td>
</tr>
<tr>
<td>_get_MyOperator/1</td>
<td>CosNotifyChannelAdmin,  min_ConsumerAdmin, 39</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>CosNotifyChannelAdmin,  ProxySupplier, 55</td>
<td></td>
</tr>
<tr>
<td>_get_MyType/1</td>
<td>CosNotifyChannelAdmin,  ProxyConsumer, 46</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>CosNotifyChannelAdmin,  ProxySupplier, 55</td>
<td></td>
</tr>
<tr>
<td>_get_constraint_grammar/1</td>
<td>CosNotifyFilter,  Filter, 79</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>CosNotifyFilter_MappingFilter, 86</td>
<td></td>
</tr>
<tr>
<td>_get_default_consumer_admin/1</td>
<td>CosNotifyChannelAdmin_EventChannel, 42</td>
<td></td>
</tr>
<tr>
<td>_get_default_filter_factory/1</td>
<td>CosNotifyChannelAdmin_EventChannel, 42</td>
<td></td>
</tr>
<tr>
<td>_get_default_supplier_admin/1</td>
<td>CosNotifyChannelAdmin_EventChannel, 42</td>
<td></td>
</tr>
<tr>
<td>_get_default_value/1</td>
<td>CosNotifyFilter_MappingFilter, 86</td>
<td></td>
</tr>
<tr>
<td>_get_lifetime_filter/1</td>
<td>CosNotifyChannelAdmin,  min_ConsumerAdmin, 40</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>CosNotifyChannelAdmin,  ProxySupplier, 56</td>
<td></td>
</tr>
<tr>
<td>_get_priority_filter/1</td>
<td>CosNotifyChannelAdmin,  min_ConsumerAdmin, 39</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CosNotifyChannelAdmin,  ProxySupplier, 55</td>
<td></td>
</tr>
<tr>
<td>_get_pull_consumers/1</td>
<td>CosNotifyChannelAdmin,  SupplierAdmin, 62</td>
<td>62</td>
</tr>
<tr>
<td>_get_pull_suppliers/1</td>
<td>CosNotifyChannelAdmin,  SupplierAdmin, 63</td>
<td>63</td>
</tr>
<tr>
<td>_get_push_consumers/1</td>
<td>CosNotifyChannelAdmin,  min_SupplierAdmin, 40</td>
<td>40</td>
</tr>
<tr>
<td>_get_push_suppliers/1</td>
<td>CosNotifyChannelAdmin,  min_ConsumerAdmin, 40</td>
<td></td>
</tr>
<tr>
<td>_get_value_type/1</td>
<td>CosNotifyFilter_MappingFilter, 86</td>
<td></td>
</tr>
<tr>
<td>_set_lifetime_filter/2</td>
<td>CosNotifyChannelAdmin,  min_ConsumerAdmin, 40</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>CosNotifyChannelAdmin,  ProxySupplier, 56</td>
<td></td>
</tr>
<tr>
<td>_set_priority_filter/2</td>
<td>CosNotifyChannelAdmin,  min_ConsumerAdmin, 40</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CosNotifyChannelAdmin,  ProxySupplier, 55</td>
<td></td>
</tr>
<tr>
<td>add_constraints/2</td>
<td>CosNotifyFilter,  Filter, 79</td>
<td>79</td>
</tr>
<tr>
<td>add_filter/2</td>
<td>CosNotifyFilter_Admin, 83</td>
<td></td>
</tr>
<tr>
<td>add_mapping_constraints/2</td>
<td>CosNotifyFilter_MappingFilter, 86</td>
<td></td>
</tr>
<tr>
<td>attach_callback/2</td>
<td>CosNotifyFilter,  Filter, 81</td>
<td>81</td>
</tr>
<tr>
<td>connect_any_pull_consumer/2</td>
<td>CosNotifyChannelAdmin,  ProxyPullSupplier, 50</td>
<td>50</td>
</tr>
<tr>
<td>connect_any_pull_supplier/2</td>
<td>CosNotifyChannelAdmin,  ProxyPullConsumer, 48</td>
<td>48</td>
</tr>
<tr>
<td>connect_any_push_consumer/2</td>
<td>CosNotifyChannelAdmin,  ProxyPushSupplier, 53</td>
<td>53</td>
</tr>
<tr>
<td>connect_any_push_supplier/2</td>
<td>CosNotifyChannelAdmin,  ProxyPushSupplier, 53</td>
<td></td>
</tr>
</tbody>
</table>

cosNotification Application
Index of Modules and Functions

CosNotifyChannelAdmin

connect_sequence_pull_consumer/2
CosNotifyChannelAdminSequenceProxyPullConsumer, 52

connect_sequence_pull_supplier/2
CosNotifyChannelAdminSequenceProxyPullSupplier, 58

connect_sequence_push_consumer/2
CosNotifyChannelAdminSequenceProxyPushConsumer, 60

connect_sequence_push_supplier/2
CosNotifyChannelAdminSequenceProxyPushSupplier, 67

connect_structured_pull_consumer/2
CosNotifyChannelAdminStructuredProxyPullSupplier, 65

connect_structured_pull_supplier/2
CosNotifyChannelAdminStructuredProxyPullConsumer, 71

connect_structured_push_consumer/2
CosNotifyChannelAdminStructuredProxyPushSupplier, 69

connect_structured_push_supplier/2
CosNotifyChannelAdminStructuredProxyPushConsumer, 75

CosNotification

'AnyOrder'/0, 34
'BestEffort'/0, 33
'ConnectionReliability'/0, 33
'DeadlineOrder'/0, 34
'DefaultPriority'/0, 34
'DiscardPolicy'/0, 34
'EventReliability'/0, 33
'FifoOrder'/0, 34
'HighestPriority'/0, 33
'LifeOrder'/0, 34
'LowestPriority'/0, 33
'MaxConsumers'/0, 35
'MaxEventsPerConsumer'/0, 34
'MaxQueueLength'/0, 35
'MaxSuppliers'/0, 35
'MaximumBatchSize'/0, 34
'OrderPolicy'/0, 34
'PacingInterval'/0, 34
'Persistent'/0, 33
'Priority'/0, 33
'PriorityOrder'/0, 34
'RejectNewEvents'/0, 34
'StartTime'/0, 33
'StartTimeSupported'/0, 34
'StopTime'/0, 33
'StopTimeSupported'/0, 34
'Timeout'/0, 34

CosNotification_AdminPropertiesAdmin

get_admin/1, 36
set_admin/2, 36

CosNotification_QoSAdmin

get_qos/1, 37
set_qos/2, 37
validate_qos/2, 38

CosNotificationApplication

create_structured_event/6, 93
install/0, 90
install/1, 90
install_event/0, 90
install_event/1, 90
start/0, 91
start_factory/0, 92
start_factory/1, 92
start_filter_factory/0, 93
start_filter_factory/1, 91
start_global_factory/0, 91
start_global_factory/1, 91
stop/0, 91
stop_factory/1, 92
stop_filter_factory/1, 93
type_check/0, 93
uninstall/0, 90
uninstall/1, 91
uninstall_event/0, 91
uninstall_event/1, 91

CosNotifyChannelAdmin_ConsumerAdmin

_get_MyChannel/1, 39
_get_MyID/1, 39
_get_MyOperator/1, 39
_get_lifetime_filter/1, 40
_get_priority_filter/1, 39
_get_pull_suppliers/1, 40
_get_push_suppliers/1, 40
_get_lifetime_filter/2, 40
_set_lifetime_filter/2, 40
_set_priority_filter/2, 40

101
Index of Modules and Functions

destroy/1, 41
get_proxy_supplier/2, 40
obtain_notification_pull_supplier/2, 41
obtain_notification_push_supplier/2, 41
obtain_pull_supplier/1, 41
obtain_push_supplier/1, 41

CosNotifyChannelAdmin_EventChannel
_get_MyFactory/1, 42
_get_default_consumer_admin/1, 42
_get_default_filter_factory/1, 42
_get_default_supplier_admin/1, 42
destroy/1, 44
for_consumers/1, 43
for_suppliers/1, 43
get_all_consumeradmins/1, 44
get_all_supplieradmins/1, 44
get_consumeradmin/2, 43
get_supplieradmin/2, 44
new_for_consumers/2, 43
new_for_suppliers/2, 43

CosNotifyChannelAdmin_EventChannelFactory
create_channel/3, 45
get_all_channels/1, 45
get_event_channel/2, 45

CosNotifyChannelAdmin_ProxyConsumer
_get_MyAdmin/1, 46
_get_MyType/1, 46
obtain_subscription_types/2, 46
validate_event_qos/2, 47

CosNotifyChannelAdmin_ProxyPullConsumer
connect_any_pull_supplier/2, 48
disconnect_pull_consumer/1, 49
resume_connection/1, 48
suspend_connection/1, 48

CosNotifyChannelAdmin_ProxyPullSupplier
connect_any_pull_supplier/2, 50
disconnect_pull_supplier/1, 51
pull/1, 50
try_pull/1, 50

CosNotifyChannelAdmin_ProxyPushConsumer
connect_any_push_supplier/2, 52
disconnect_push_consumer/1, 52
push/2, 52

CosNotifyChannelAdmin_ProxyPushSupplier
connect_any_push_supplier/2, 53
disconnect_supplier/1, 54
resume_connection/1, 54

CosNotifyChannelAdmin_ProxyPushSupplier
connect_structured_pull_consumer/1, 70
resume_connection/1, 70
suspend_connection/1, 70

CosNotifyChannelAdmin_ProxyPushSupplier
connect_structured_pull_supplier/2, 102

cosNotification Application
Index of Modules and Functions

73 disconnect_structured_push_consumer/1, 74
push_structured_event/2, 73
CosNotifyChannelAdmin_StructuredProxyPushSupplier
connect_structured_push_consumer/2, 75
disconnect_structured_push_supplier/1, 76
resume_connection/1, 75
suspend_connection/1, 75
CosNotifyChannelAdmin_SupplierAdmin
_get_MyChannel/1, 62
_get_MyID/1, 62
_get_MyOperator/1, 62
_get_pull_consumers/1, 62
_get_push_consumers/1, 63
destroy/1, 64
get_proxy_consumer/2, 63
obtain_notification_pull_consumer/2, 63
obtain_notification_push_consumer/2, 63
obtain_pull_consumer/1, 63
obtain_push_consumer/1, 64
CosNotifyComm_NotifySubscribe
subscription_change/3, 78
CosNotifyComm_NotifyPublish
offer_change/3, 77
CosNotifyFilter_Filter
_get_constraint_grammar/1, 79
add_constraints/2, 79
attach_callback/2, 81
destroy/1, 81
detach_callback/2, 81
get_all_constraints/1, 80
get_callbacks/1, 82
get_constraints/2, 80
match/2, 81
match_structured/2, 81
modify_constraints/3, 79
remove_all_constraints/1, 80
CosNotifyFilter_FilterAdmin
add_filter/2, 83
get_all_filters/1, 83
get_filter/2, 83
remove_all_filters/1, 84
remove_filter/2, 83
CosNotifyFilter_FilterFactory
create_filter/2, 85
create_mapping_filter/2, 85
CosNotifyFilter_MappingFilter
_get_constraint_grammar/1, 86
_get_default_value/1, 86
_get_value_type/1, 86
add_mapping_constraints/2, 86
destroy/1, 89
get_all_mapping_constraints/1, 88
get_mapping_constraints/2, 88
match/2, 89
match_structured/2, 89
modify_constraints/3, 87
remove_all_mapping_constraints/1, 88
create_channel/3
CosNotifyChannelAdmin_EventChannelFactory
45
cosNotificationApplication
93
destroy/1
CosNotifyChannelAdmin_ConsumerAdmin
41
CosNotifyChannelAdmin_EventChannel
44
CosNotifyChannelAdmin_SupplierAdmin
64
CosNotifyFilter_Filter
81
CosNotifyFilter_MappingFilter
89
detach_callback/2
CosNotifyFilter_Filter
81
disconnect_pull_consumer/1
CosNotifyChannelAdmin_ProxyPullConsumer
49
disconnect_pull_supplier/1
CosNotifyChannelAdmin_ProxyPullSupplier
51
disconnect_push_consumer/1
CosNotifyChannelAdmin_ProxyPushConsumer
Index of Modules and Functions

52
disconnect_push_supplier/1
CosNotifyChannelAdmin_ProxyPushSupplier, 54
disconnect_sequence_pull_consumer/1
CosNotifyChannelAdmin_SequenceProxyPullConsumer, 59
disconnect_sequence_pull_supplier/1
CosNotifyChannelAdmin_SequenceProxyPullSupplier, 61
disconnect_sequence_push_consumer/1
CosNotifyChannelAdmin_SequenceProxyPushConsumer, 66
disconnect_sequence_push_supplier/1
CosNotifyChannelAdmin_SequenceProxyPushSupplier, 68
disconnect_structured_pull_consumer/1
CosNotifyChannelAdmin_StructuredProxyPullConsumer, 70
disconnect_structured_pull_supplier/1
CosNotifyChannelAdmin_StructuredProxyPullSupplier, 72
disconnect_structured_push_consumer/1
CosNotifyChannelAdmin_StructuredProxyPushConsumer, 74
disconnect_structured_push_supplier/1
CosNotifyChannelAdmin_StructuredProxyPushSupplier, 76

for_consumers/1
CosNotifyChannelAdmin_EventChannel, 43
for_suppliers/1
CosNotifyChannelAdmin_EventChannel, 43

get_admin/1
CosNotification_AdminPropertiesAdmin, 36

get_all_channels/1
CosNotifyChannelAdmin_EventChannelFactory, 45

get_all_constraints/1
CosNotifyFilter_Filter, 80

get_all_consumeradmins/1
CosNotifyChannelAdmin_EventChannel, 44

get_all_filters/1
CosNotifyFilter_FilterAdmin, 83

get_all_mapping_constraints/1
CosNotifyFilter_MappingFilter, 88

get_all_supplieradmins/1
CosNotifyChannelAdmin_EventChannel, 44

get_callbacks/1
CosNotifyFilter_Filter, 82

get_constraints/2
CosNotifyFilter_Filter, 80

get_consumeradmin/2
CosNotifyChannelAdmin_EventChannel, 43

get_event_channel/2
CosNotifyChannelAdmin_EventChannelFactory, 45

get_filter/2
CosNotifyFilter_FilterAdmin, 83

get_mapping_constraints/2
CosNotifyFilter_MappingFilter, 88

get_proxy_consumer/2
CosNotifyChannelAdmin_SupplierAdmin, 63

get_proxy_supplier/2
CosNotifyChannelAdmin_ConsumerAdmin, 40

get_qos/1
CosNotification_QoSAdmin, 37

get_supplieradmin/2
CosNotifyChannelAdmin_EventChannel, 44

install/0
CosNotificationApp, 90
install/1
  cosNotificationApp , 90
install_event/0
  cosNotificationApp , 90
install_event/1
  cosNotificationApp , 90

match/2
  CosNotifyFilter_Filter , 81
  CosNotifyFilter_MappingFilter , 89
match_structured/2
  CosNotifyFilter_Filter , 81
  CosNotifyFilter_MappingFilter , 89
modify_constraints/3
  CosNotifyFilter_Filter , 79
  CosNotifyFilter_MappingFilter , 87

new_for_consumers/2
  CosNotifyChannelAdmin_EventChannel , 43
new_for_suppliers/2
  CosNotifyChannelAdmin_EventChannel , 43

obtain_notification_pull_consumer/2
  CosNotifyChannelAdmin_SupplierAdmin , 63
obtain_notification_pull_supplier/2
  CosNotifyChannelAdmin_ConsumerAdmin , 41
obtain_notification_push_consumer/2
  CosNotifyChannelAdmin_SupplierAdmin , 63
obtain_notification_push_supplier/2
  CosNotifyChannelAdmin_ConsumerAdmin , 41
obtain_offered_types/2
  CosNotifyChannelAdmin_ProxySupplier , 56
obtain_pull_consumer/1
  CosNotifyChannelAdmin_SupplierAdmin , 63
obtain_pull_supplier/1

CosNotifyChannelAdmin_ConsumerAdmin , 41
obtain_push_consumer/1
  CosNotifyChannelAdmin_SupplierAdmin , 64
obtain_push_supplier/1
  CosNotifyChannelAdmin_ConsumerAdmin , 41
obtain_subscription_types/2
  CosNotifyChannelAdmin_ProxyConsumer , 46
offer_change/3
  CosNotifyComm_NotifyPublish , 77

pull/1
  CosNotifyChannelAdmin_ProxyPullSupplier , 50
pull_structured_event/1
  CosNotifyChannelAdmin_StructuredProxyPullSupplier , 71
pull_structured_events/2
  CosNotifyChannelAdmin_SequenceProxyPullSupplier , 60
push/2
  CosNotifyChannelAdmin_ProxyPushConsumer , 52
push_structured_event/2
  CosNotifyChannelAdmin_StructuredProxyPushConsumer , 73
push_structured_events/2
  CosNotifyChannelAdmin_SequenceProxyPushConsumer , 65
remove_all_constraints/1
  CosNotifyFilter_Filter , 80
remove_all_filters/1
  CosNotifyFilterFilterAdmin , 84
remove_all_mapping_constraints/1
  CosNotifyFilter_MappingFilter , 88

cosNotification Application
Index of Modules and Functions

remove_filter/2
   CosNotifyFilter_Admin, 83
resume_connection/1
   CosNotifyChannelAdmin_ProxyPullConsumer, 48
   CosNotifyChannelAdmin_ProxyPushSupplier, 53
   CosNotifyChannelAdmin_SequenceProxyPullConsumer, 58
   CosNotifyChannelAdmin_SequenceProxyPushSupplier, 67
   CosNotifyChannelAdmin_StructuredProxyPullConsumer, 69
   CosNotifyChannelAdmin_StructuredProxyPushSupplier, 75

set_admin/2
   CosNotification_AdminPropertiesAdmin, 36
set_qos/2
   CosNotification_QoSAdmin, 37
start/0
   cosNotificationApp, 91
start_factory/0
   cosNotificationApp, 92
start_factory/1
   cosNotificationApp, 92
start_filter_factory/0
   cosNotificationApp, 93
start_global_factory/0
   cosNotificationApp, 91
start_global_factory/1
   cosNotificationApp, 91
stop/0
   cosNotificationApp, 91
stop_factory/1
   cosNotificationApp, 92
stop_filter_factory/1
   cosNotificationApp, 93
subscription_change/3
   CosNotifyComm_NotifySubscribe, 78
suspend_connection/1
   CosNotifyChannelAdmin_ProxyPullConsumer, 48
   CosNotifyChannelAdmin_ProxyPushSupplier, 53
   CosNotifyChannelAdmin_SequenceProxyPullConsumer, 58
   CosNotifyChannelAdmin_SequenceProxyPushSupplier, 67
   CosNotifyChannelAdmin_StructuredProxyPullConsumer, 69
   CosNotifyChannelAdmin_StructuredProxyPushSupplier, 75
try_pull/1
   CosNotifyChannelAdmin_ProxyPullSupplier, 50
try_pull_structured_event/1
   CosNotifyChannelAdmin_StructuredProxyPullSupplier, 72
try_pull_structured_events/2
   CosNotifyChannelAdmin_SequenceProxyPullSupplier, 61
type_check/0
   cosNotificationApp, 93
uninstall/0
   cosNotificationApp, 90
uninstall/1
   cosNotificationApp, 91
uninstall_event/0
   cosNotificationApp, 91
uninstall_event/1
   cosNotificationApp, 91
validate_event_qos/2
   CosNotifyChannelAdmin_ProxyConsumer, 47
   CosNotifyChannelAdmin_ProxySupplier, 56
validate_qos/2

CosNotification, QoS Admin, 38