PreCache NetInjector API Reference

Contents

PreCache NetInjector API Reference

Overview

Java Classes

C++ Classes

C Types and Functions

Overview

About this guide

— the goals and objectives of this reference

The *PreCache NetInjector API Reference* provides basic information about the APIs for the NetiInjector Event Agent. The APIs expose the Event Agent's basic functions so that application developers can build programs that yield meaningful notifications.

The API has been packaged in C Types and Functions, C++ Classes, and Java Classes. While each has its advantages and disadvantages for the application developer, each API exposes the same baisc functionality.

Organization and Content

— a high-level view of the content

API classes and types are arranged alphabetically, with frequent cross references in the PDF and HTML formats.

Essentially, the *API Reference* describes these features:

- Callbacks
- Exceptions
- Filters
- Notifications
- Proxies
- Subscription handlers

Conventions

— typefaces for variables, fields, methods, and the like

The following typeface conventions have been observed in these pages:

- `code`
- *fields*
- links
- *methods*
- **system items**
- `values`
- *variables*

Note: For better readability in both PDF and HTML, these conventions may sometimes print as either boldface or italic.

Java Classes

— classes, functions, and exception handlers for the Java deployment of the Event Agent

Class Hierarchy: *the Event Agent Java class hierarchy*

AndFilter: *a conjunction of attribute filter expressions*

ChannelInfo: *an abstraction for the channel configuration*

CompoundFilter: *an abstract class for aggregating filters*

CtrlNotification: *notifications for internal events*

Exception classes: *descriptions of the exception classes*

Filter: *an abstract class for attribute filters*

FilterFactory: *generate filters from expressions*

Notification: *the data for an event to be published*

OrFilter: *a disjunction of attribute filter expressions*

Proxy: *connectivity to a channel*

SimpleFilter: *filters for testing expressions*

Subscriber: *the interface for callback operations*

SubscriptionHandle: *the subscription identification class*

Class Hierarchy

— the Event Agent Java class hierarchy

java.lang.Object
- ↳com.precache.event.ChannelInfo
- ↳com.precache.event.Notification
  - ↳CtrlNotification
- ↳com.precache.event.FilterFactory
- ↳com.precache.event.Filter
  - ↳CompoundFilter
    - ↳AndFilter
    - ↳OrFilter
  - ↳SimpleFilter
- ↳com.precache.event.Proxy
- ↳com.precache.event.SubscriptionHandle
- ↳java.lang.Throwable
  - ↳java.lang.Exception
    - ↳com.precache.channel.InvalidChannelException
    - ↳com.precache.event.EventException
      - ↳InvalidFilterException
      - ↳InvalidNotificationException
      - ↳InvalidSubjectException
      - ↳InvalidSubscriptionHandleException
      - ↳ProxyStateException

AndFilter

— a conjunction of attribute filter expressions

AndFilter objects facilitate notification filtering by conjunctive expressions.

Hierarchy

java.lang.Object
- ↳com.precache.event.Filter
  - ↳com.precache.event.CompoundFilter
    - ↳com.precache.event.AndFilter

Constructors

AndFilter by ChannelInfo

```
public AndFilter(ChannelInfo c)
```

Constructs an AndFilter object and associates it with a ChannelInfo object, providing access to a channel's attributes and validation against the associated channel's configuration.

c **:** the ChannelInfo object to be associated with the AndFilter

AndFilter by Proxy

```
public AndFilter(Proxy p)
```

Constructs an AndFilter object and associates it with a Proxy object, providing access to a channel's attributes and validation against the associated channel's configuration.

p: the proxy to be associated with the AndFilter

Methods

AndFilter inherits *add*, *getFilters*, *remove*, *reset*, and *toString* from class com.precache.event.CompoundFilter.

AndFilter inherits *clone*, *equals*, *getClass*, *hashCode*, *notify*, *notifyAll*, and *wait* from class java.lang.Object.

finalize

```
public void finalize()
```

Frees up resources associated with this object.

overrides: *finalize* in class Object

match

```
public boolean match(Notification n)
```

Verifies whether a notification type matches the AndFilter object.

n: the notification to be matched

returns: `true`, if the notification matches; otherwise `false`

overrides: *match* in class Filter

clone

```
public Filter clone()
```

Constructs a clone of the AndFilter object.

overrides: *clone* in class Filter

returns: the cloned AndFilter

ChannelInfo

— an abstraction for the channel configuration

ChannelInfo needs to be associated with various objects such as Filter or Notification.

Hierarchy

java.lang.Object
↳com.precache.event.ChannelInfo

Methods

ChannelInfo inherits *clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString,* and *wait* from class java.lang.Object.

CompoundFilter

— an abstract class for aggregating filters

CompoundFilter is used to aggregate filters into a single object. CompoundFilter serves as a base class for AndFilter and OrFilter objects.

Hierarchy

java.lang.Object
↳com.precache.event.Filter
↳com.precache.event.CompoundFilter

subclasses: AndFilter, OrFilter

Constructors

CompoundFilter()

public CompoundFilter()

Methods

CompoundFilter inherits *clone* and *match* from class com.precache.event.Filter .

CompoundFilter inherits *clone, equals, finalize, getClass, hashCode, notify, notifyAll,* and *wait* from class java.lang.Object.

add

```
public void add(Filter f)
```

Adds another subfilter to this object.

f: a Filter object for the added subfilter

throws: *add* throws an InvalidFilterException if the subfilter has already been added.

remove

```
public void remove(Filter f)
```

Removes a subfilter from this object.

f: a Filter object for a subfilter to be removed

reset

```
public void reset()
```

Removes all existing subfilters for this object.

getFilters

```
public vector getFilters()
```

returns: *getFilters* returns a vector object storing the constituent Filter objects in the order in which they were added.

toString

```
public string toString()
```

Converts the filtered object to a string representation.

overrides: *toString* in class Object

CtrlNotification

— notifications for internal events

CtrlNotification delivers notifications for internal events such as an asynchronous exception or subscription confirmations.

Control notifications have a fixed attribute configuration for *name*, *message*, and *level*.

name: *PC_STRING*

message: *PC_STRING*

level: *PC_LONG*

CtrlNotification inherits all the Notification class functionality.

Hierarchy

java.lang.Object
↳com.precache.event.Notification
↳com.precache.event.CtrlNotification

Level attribute values

PC_INFO_LEVEL

The control notification has three attributes: name, message, and level. The level attribute has three possible values: *0 = PC_INFO_LEVEL*; *1 = PC_CTRL_LEVEL*; *2 = PC_ERROR_LEVEL*.

```
public static final int PC_INFO_LEVEL
```

PC_INFO_LEVEL control notifications deliver status updates and debugging information. They do not play an active role in API usage; their delivery should not affect the behavior of an application.

PC_CTRL_LEVEL

```
public static final int PC_CTRL_LEVEL
```

PC_CTRL_LEVEL control notifications report asynchronous system events that might be relevant for application behavior. Applications should handle such events in order to rely on a deterministic behavior.

PC_ERROR_LEVEL

```
public static final int PC_ERROR_LEVEL
```

PC_ERROR_LEVEL notifies the application of the occurrence of a critical asynchronous event such us an unhandled exception thrown by an internal Proxy thread.

Note: Failure to handle *PC_ERROR_LEVEL* triggers program termination.

Methods

GetName

```
public string GetName()
```

GetName retrieves the value of the *name* attribute of the control notification.

GetMessage

```
public string GetMessage()
```

GetMessage retrieves the value of the *message* attribute of the control notification.

GetLevel

```
public int GetLevel()
```

GetLevel retrieves the value of the *level* attribute of the control notification. The possible values passed are:

0: PC_INFO_LEVEL

1: PC_CTRL_LEVEL

2: PC_ERROR_LEVEL

SetName

```
public void SetName(string szName)
```

SetName changes the value of the *name* attribute of the control notification.

SetMessage

```
public void SetMessage(string szMessage)
```

SetMessage changes the content of the *message* attribute of the control notification.

SetLevel

```
public void SetLevel(int unLevel)
```

SetLevel changes the severity level of the *level* attribute of the control notification.

Exception classes

— descriptions of the exception classes

This section describes all the NetInjector Event Agent exception classes.

EventException

Hierarchy

java.lang.Object
↳java.lang.Throwable
↳java.lang.Exception
↳com.precache.event.EventException

subclasses: InvalidFilterException, InvalidNotificationException, InvalidSubjectException, InvalidSubscriptionHandleException, ProxyStateException

Methods

EventException inherits *fillInStackTrace*, *getLocalizedMessage*, *getMessage*, *printStackTrace*, and *toString* from class java.lang.Throwable.

EventException inherits *clone*, *equals*, *finalize*, *getClass*, *hashCode*, *notify*, *notifyAll*, and *wait* from class java.lang.Object.

InvalidChannelException

InvalidChannelException occurs if a specified channel ID is invalid, or if the specified publisher ID is not allowed to access the channel.

Hierarchy

java.lang.Object
↳java.lang.Throwable
↳java.lang.Exception
↳com.precache.channel.InvalidChannelException

Methods

InvalidFilterException inherits *fillInStackTrace*, *getLocalizedMessage*, *getMessage*, *printStackTrace* *printStackTrace*, and *toString* from class java.lang.Throwable

InvalidFilterException inherits *clone*, *equals*, *finalize*, *getClass*, *hashCode*, *notify*, *notifyAll*, and *wait* from class java.lang.Object

InvalidFilterException

An *InvalidFilterException* occurs if the application attempts to construct a Filter object with an invalid attribute filter expression.

Hierarchy

java.lang.Object
↳java.lang.Throwable
↳java.lang.Exception
↳com.precache.event.EventException
↳com.precache.event.InvalidFilterException

Methods

InvalidFilterException inherits *fillInStackTrace*, *getLocalizedMessage*, *getMessage*, *printStackTrace* *printStackTrace*, and *toString* from class java.lang.Throwable

InvalidFilterException inherits *clone*, *equals*, *finalize*, *getClass*, *hashCode*, *notify*, *notifyAll*, and *wait* from class java.lang.Object

InvalidNotificationException

An *InvalidNotificationException* occurs if the application attempts to construct a Notification object with attribute values inconsistent with the notification type of the associated channel.

Hierarchy

java.lang.Object
↳ java.lang.Throwable
↳ java.lang.Exception
↳com.precache.event.EventException
↳com.precache.event.InvalidNotificationException

Methods

InvalidNotificationException inherits *fillInStackTrace*, *getLocalizedMessage*, *getMessage*, *printStackTrace*, and *toString* from class java.lang.Throwable

InvalidNotificationException inherits *clone*, *equals*, *finalize*, *getClass*, *hashCode*, *notify*, *notifyAll*, and *wait* from class java.lang.Object

InvalidSubjectException

An *InvalidSubjectException* occurs if the application references a nonexistent subject in a Notification object or an invalid subject filter in a subscription, as determined by the associated channel.

Hierarchy

java.lang.Object
↳ java.lang.Throwable
↳java.lang.Exception
↳com.precache.event.EventException
↳com.precache.event.InvalidSubjectException

Methods

InvalidSubjectException inherits *fillInStackTrace*, *getLocalizedMessage*, *getMessage*, *printStackTrace*, and *toString* from class java.lang.Throwable

InvalidSubjectException inherits *clone*, *equals*, *finalize*, *getClass*, *hashCode*, *notify*, *notifyAll*, and *wait* from class java.lang.Object

InvalidSubscriptionHandleException

An *InvalidSubscriptionHandleException* occurs if the application references an obsolete or invalid SubscriptionHandle object.

Hierarchy

java.lang.Object
↳java.lang.Throwable
↳java.lang.Exception
↳com.precache.event.EventException
↳com.precache.event.InvalidSubscriptionHandleException

Methods

InvalidSubscriptionHandleException inherits *fillInStackTrace*, *getLocalizedMessage*, *getMessage*, *printStackTrace*, and *toString* from class java.lang.Throwable

InvalidSubscriptionHandleException inherits *clone*, *equals*, *finalize*, *getClass*, *hashCode*, *notify*, *notifyAll*, and *wait* from class java.lang.Object

ProxyStateException

ProxyStateException is thrown by the Proxy state management routines.

Hierarchy

java.lang.Object
↳java.lang.Throwable
↳ java.lang.Exception
↳ com.precache.event.EventException
↳com.precache.event.ProxyStateException

Methods

ProxyStateException inherits *fillInStackTrace*, *getLocalizedMessage*, *getMessage*, *printStackTrace* *printStackTrace*, and *toString* from class java.lang.Throwable

ProxyStateException inherits *clone*, *equals*, *finalize*, *getClass*, *hashCode*, *notify*, *notifyAll*, and *wait* from class java.lang.Object

Filter

— an abstract class for attribute filters

Class Filter is the base class for all attribute filters.

Hierarchy

java.lang.Object
↳com.precache.event.Filter

subclasses: CompoundFilter, SimpleFilter

Constructor

public Filter()

Methods

Filter inherits *clone*, *equals*, *finalize*, *getClass*, *hashCode*, *notify*, *notifyAll*, *toString*, and *wait* from class java.lang.Object.

match

```
public abstract boolean match(Notification n)
```

Verifies whether a notification object matches the Filter object.

n: the notification that is to be matched

returns: `true` if the notification matches the Filter object; otherwise `false`

clone

```
public abstract Filter clone()
```

Constructs a clone of this object.

FilterFactory

— generate filters from expressions

The FilterFactory is a singleton class that generates Filter objects from string attribute filter expressions.

Hierarchy

java.lang.Object
↳com.precache.event.FilterFactory

Methods

FilterFactory inherits *clone*, *equals*, *getClass*, *hashCode*, *notify*, *notifyAll*, *toString*, and *wait* from class java.lang.Object.

finalize

```
public void finalize()
```

Frees up resources associated with this object

overrides: *finalize* in class Object

createFilter

```
public Filter createFilter(String strFilterExp)
```

Creates a Filter object from a string attribute filter expression and associates the object with a Proxy.

strFilterExp: the attribute filter expression to be parsed

returns: the new Filter object

throws: *createFilter* throws an InvalidFilterException if there are syntactic or semantic errors in strFilterExp.

Notification

— the data for an event to be published

A *Notification* object stores information about a single event that is to be published over a channel.

The object stores the event's subject and the attribute values for the associated channel.

Hierarchy

java.lang.Object
↳com.precache.event.Notification

subclasses: CtrlNotification

The lack of unsigned types in Java requires a policy for the mapping between Java types and the actual types of predefined attributes as determined by a channel notification type.

For adding discretionary attributes, the attribute type selected is the closest equivalent of the Java type used (e.g., attribute type PC_FLOAT has been selected for Java type float).

For getting discretionary attributes, a single method is provided, which returns the value in an Object whose true class depends on the actual type of the retrieved attribute.

attribute type	Java type
PC_CHAR	char
PC_UCHAR	char

attribute type	Java type
PC_SHORT	short
PC_USHORT	int
PC_INT	int
PC_UINT	long
PC_LONG	long
PC_ULONG	long (does not support some values of type PC_ULONG)
PC_FLOAT	float
PC_DOUBLE	double
PC_BOOL	boolean
PC_STRING	string
PC_UCHARARRAY	char[]
PC_SHORTARRAY	short[]
PC_USHORTARRAY	int[]
PC_INTARRAY	int[]
PC_UINTARRAY	long[]
PC_LONGARRAY	long[]
PC_ULONGARRAY	long[] (does not support some values of type PC_ULONG)
PC_FLOATARRAY	float[]
PC_DOUBLEARRAY	double[]
PC_BOOLARRAY	boolean[]
PC_BYTEARRAY	byte[]

Constructors

Notification(Proxy *p*, string *subject*)

```
public Notification(Proxy p, string subject)
```

Constructs a Notification object with a partiucular subject string and associates this notification with a proxy.

p: the proxy with which this object will be associated, providing access to a channel and validation against the channel's subject tree and notification type

subject: a subject for the notification

throws: Notification throws an InvalidSubjectException if the specified subject is invalid for the underlying channel.

Notification (Proxy p, string subject, string attrvals)

```
public Notification(Proxy p, string subject, string
    attrvals)
```

Constructs a Notification object with a partiucular subject string and attribute assignments and associates this notification with a Proxy.

p: the proxy with which this object will be associated, providing access to a channel and validation against the channel's subject tree and notification type

subject: a subject for the notification

attrvals: a comma-separated sequences of attribute assignments of the form `name=(type)value`

throws: Notification throws an InvalidSubjectException if the specified subject is invalid for the underlying channel

Notification throws an InvalidNotificationException if the specified attribute assignments are syntactically incorrect or are inconsistent with the notification type of the underlying channel.

Methods

Notification inherits *clone*, *equals*, *getClass*, *hashCode*, *notify*, *notifyAll*, *toString*, and *wait* from class java.lang.Object

finalize

```
public void finalize()
```

Frees up resources associated with this object.

overrides: *finalize* in class Object

subject

```
public string subject()
```

returns: the subject of this notification

subjectMapping

```
public int[] subjectMapping()
```

returns: the integer mapping of the subject associated with the notification

print

```
public void print()
```

Prints a text rendering of this object's contents on the standard output stream.

setPredefinedAttr by name

```
public void setPredefinedAttr(string name,type val)
```

Sets the value of a specified attribute of the notification, identified by its name.

name: the name of the predefined attribute

type: the data type of the attribute

val: the value to be assigned to the attribute

throws: *setPredefinedAttr* throws an InvalidNotificationException if the specified attribute is not a predefined attribute.

As delineated below, this method is overloaded on all allowed types for parameter *val*.

```
setPredefinedAttr(string name,char val)
setPredefinedAttr(string name,short val)
setPredefinedAttr(string name,int val)
setPredefinedAttr(string name,long val)
setPredefinedAttr(string name,float val)
setPredefinedAttr(string name,double val)
setPredefinedAttr(string name,boolean val)
setPredefinedAttr(string name,string val)
```

```
setPredefinedAttr(string name,char[] val)
setPredefinedAttr(string name,short[] val)
setPredefinedAttr(string name,int[] val)
setPredefinedAttr(string name,long[] val)
setPredefinedAttr(string name,float[] val)
setPredefinedAttr(string name,double[] val)
setPredefinedAttr(string name,boolean[] val)
setPredefinedAttr(string name,byte[] val)
```

setPredefinedAttr by position

```
public void setPredefinedAttr(int pos,type val)
```

Sets the value of a specified attribute of the notification, identified by its position.

Note: The position of an attribute is determined by the sequence of attributes set when the channel was configured by the NetInjector administrator.

pos: the position (starting with 0) of the channel's predefined attribute

type: the data type of the attribute

val: the value to be assigned to the attribute

throws: *setPredefinedAttr* throws an InvalidNotificationException if there is no predefined attribute in the specified position, or if the true class of the specified value is of an incompatible type for the specified attribute, as determined by the notification type of the underlying channel.

As delineated below, this method is overloaded on all allowed types for parameter *val*.

```
setPredefinedAttr(int pos,short val)
setPredefinedAttr(int pos,int val)
setPredefinedAttr(int pos,long val)
setPredefinedAttr(int pos,float val)
setPredefinedAttr(int pos,double val)
setPredefinedAttr(int pos,boolean val)
setPredefinedAttr(int pos,string val)
setPredefinedAttr(int pos,char[] val)
setPredefinedAttr(int pos,short[] val)
setPredefinedAttr(int pos,int[] val)
setPredefinedAttr(int pos,long[] val)
setPredefinedAttr(int pos,float[] val)
setPredefinedAttr(int pos,double[] val)
setPredefinedAttr(int pos,boolean[] val)
setPredefinedAttr(int pos,byte[] val)
```

getPredefinedAttr*TYPE* by name

```
public type getPredefinedAttrRETURN_TYPE(string
    name)
```

Retrieves the value of a specified attribute of the notification, identified by its name.

name: the name of the predefined attribute

type: the data type of the attribute

returns: the value of a predefined attribute of this object, identified by its name

throws: *getPredefinedAttrTYPE* throws an InvalidNotificationException if the specified name is not the name of a predefined attribute

getPredefinedAttrChar is replicated on all allowed data types, with the *TYPE* forming the suffix of the method name.

```
char getPredefinedAttrChar(string name)
short getPredefinedAttrShort(string name)
int getPredefinedAttrInt(string name)
long getPredefinedAttrLong(string name)
float getPredefinedAttrFloat(string name)
double getPredefinedAttrDouble(string name)
boolean getPredefinedAttrBoolean(string name)
string getPredefinedAttrString(string name)
char[] getPredefinedAttrCharArray(string name)
short[] getPredefinedAttrShortArray(string name)
int[] getPredefinedAttrIntArray(string name)
long[] getPredefinedAttrLongArray(string name)
float[] getPredefinedAttrFloatArray(string name)
double[] getPredefinedAttrDoubleArray(string name)
boolean[] getPredefinedAttrBooleanArray(string name)
byte[] getPredefinedAttrByteArray(string name)
```

getPredefinedAttr*TYPE* by position

```
public type getPredefinedAttrTYPE(int pos)
```

Retrieves the value of a specified attribute of the notification, identified by its position.

Note: The position of an attribute is determined by the sequence of attributes set when the channel was configured by the NetInjector administrator.

pos: the position (starting with 0) of the channel's predefined attribute

type: the data type of the attribute

returns: the value of a predefined attribute of this object, identified by its position

throws: *getPredefinedAttrTYPE* throws an InvalidNotificationException if there is no predefined attribute in the specified position

getPredefinedAttrTYPE is replicated on all allowed return types, with the *TYPE* forming the suffix of the method name.

```
short getPredefinedAttrShort(int pos)
int getPredefinedAttrInt(int pos)
long getPredefinedAttrLong(int pos)
float getPredefinedAttrFloat(int pos)
double getPredefinedAttrDouble(int pos)
boolean getPredefinedAttrBoolean(int pos)
string getPredefinedAttrString(int pos)
char[] getPredefinedAttrCharArray(int pos)
short[] getPredefinedAttrShortArray(int pos)
int[] getPredefinedAttrIntArray(int pos)
long[] getPredefinedAttrLongArray(int pos)
float[] getPredefinedAttrFloatArray(int pos)
```

```
double[] getPredefinedAttrDoubleArray(int pos)
boolean[] getPredefinedAttrBooleanArray(int pos)
byte[] getPredefinedAttrByteArray(int pos)
```

addDiscretionaryAttr

Sets the value of a discretionary attribute of the notification.

```
public void addDiscretionaryAttr(string name,type
    val)
```

name: the name of the discretionary attribute whose value is to be set

type: the data type of the attribute

val: the value to be assigned to the attribute

throws: addDiscretionaryAttr throws an InvalidNotificationException if the specified attribute name belongs to a predefined attribute, or if the specified type ID is invalid, or if the true class of the specified value is incompatible with the specified type ID.

As delineated below, *addDiscretionaryAttr* is overloaded on all allowed types for parameter *val*.

```
addDiscretionaryAttr(string name,char val)
addDiscretionaryAttr(string name,short val)
addDiscretionaryAttr(string name,int val)
addDiscretionaryAttr(string name,long val)
addDiscretionaryAttr(string name,float val)
addDiscretionaryAttr(string name,double val)
addDiscretionaryAttr(string name,boolean val)
addDiscretionaryAttr(string name,string val)
addDiscretionaryAttr(string name,char[] val)
addDiscretionaryAttr(string name,short[] val)
addDiscretionaryAttr(string name,int[] val)
addDiscretionaryAttr(string name,long[] val)
addDiscretionaryAttr(string name,float[] val)
addDiscretionaryAttr(string name,double[] val)
addDiscretionaryAttr(string name,boolean[] val)
addDiscretionaryAttr(string name,byte[] val)
```

getDiscretionaryAttr

```
public Object getDiscretionaryAttr(string name)
```

Retrieves the value of a discretionary attribute of the notification.

name: the name of the discretionary attribute whose value is to be returned

returns: the value of the specified attribute; the true class of the returned object is dependent upon the actual type of the attribute value

throws: *getDiscretionaryAttr* throws an InvalidNotificationException if the specified name is not the name of a discretionary attribute

DiscretionaryAttrNames

```
public string[] DiscretionaryAttrNames()
```

Lists all discretionary attributes by name.

returns: a sequence of the names of all discretionary attributes of this object in the order in which those attributes were added

isDiscretionaryAttr

```
public boolean isDiscretionaryAttr(string name)
```

Indicates whether the object has a discretionary attribute of a particular name.

name: the discretionary attribute name to check

returns: `true` if name is the name of a discretionary attribute of this object; otherwise `false`

OrFilter

— a disjunction of attribute filter expressions

OrFilter objects facilitate notification filtering by disjunctive expressions.

Hierarchy

java.lang.Object
↳com.precache.event.Filter
↳com.precache.event.CompoundFilter
↳com.precache.event.OrFilter

Constructors

OrFilter by ChannelInfo

```
public OrFilter(ChannelInfo c)
```

Constructs an OrFilter object and associates it with a ChannelInfo object, providing access to a channel and validation against the associated channel's configuration.

c : the ChannelInfo object to be associated with the OrFilter

OrFilter by Proxy

```
public OrFilter(Proxy p)
```

Constructs an OrFilter object and associates it with a Proxy object, providing access to a channel and validation against the associated channel's configuration.

p: the proxy to be associated with the OrFilter

Methods

OrFilter inherits *add*, *getFilters*, *remove*, *reset*, and *toString* from class com.precache.event.CompoundFilter.

OrFilter inherits *clone*, *equals*, *getClass*, *hashCode*, *notify*, *notifyAll*, and *wait* from class java.lang.Object.

finalize

```
public void finalize()
```

Frees up resources associated with this object.

overrides: *finalize* in class Object

match

```
public boolean match(Notification n)
```

Verifies whether a notification type matches the OrFilter object.

n: the notification to be matched

returns: `true`, if the notification matches; otherwise `false`

overrides: *match* in class Filter

Clone

```
public Filter Clone()
```

Constructs a clone of the OrFilter object.

overrides: *clone* in class Filter

Proxy

— connectivity to a channel

A Proxy object provides connectivity to a channel and validation of notifications against the channel's Notification type.

Proxy publishes notifications for the channel's subjects and delivers matching notifications to subscriber's via subscriber callbacks.

Hierarchy

java.lang.Object
↳com.precache.event.Proxy

Constructor

`public Proxy(int channel,int pub)`

Constructs a Proxy object for a specified channel and publisher.

channel: the ID of the channel to be accessed

pub: the ID of the application using this object

throws: *Proxy* throws an InvalidChannelException if the specified channel ID is invalid, or if the specified publisher ID is not allowed to access the channel.

Methods

Proxy inherits *clone*, *equals*, *getClass*, *hashCode*, *notify*, *notifyAll*, *toString*, and *wait* from class java.lang.Object

finalize

`public void finalize()`

Frees up resources associated with this object.

overrides: *finalize* in class Object

publish

`public void publish(Notification n)`

Publishes a notification over the associated channel .

n: the notification object to be published

throws: Proxy throws an InvalidNotificationException if the Notification object is incomplete or otherwise invalid for publication over the channel

subscribe

`public SubscriptionHandle subscribe(arguments)`

Registers a subscription for the associated channel.

The *subscribe* method may be invoked in a variety of ways.

```
public SubscriptionHandle subscribe(
   String subjectFilter,
```

```
    Subscriber sub,
    Object closure)

public SubscriptionHandle subscribe(
    String subjectFilter,
    String exp,
    Subscriber sub,
    Object closure)

public SubscriptionHandle subscribe(
    String subjectFilter,
    Filter f,
    Subscriber sub,
    Object closure)

public SubscriptionHandle subscribe(
    String subjectFilter,
    Subscriber sub,
    Object closure,
    Subscriber ctrlSubscriber)

public SubscriptionHandle subscribe(
    String subjectFilter,
    String exp,
    Subscriber sub,
    Object closure,
    Subscriber ctrlSubscriber)

public SubscriptionHandle subscribe(
    String subjectFilter,
    Filter f,
    Subscriber sub,
    Object closure,
    Subscriber ctrlSubscriber)
```

closure: an object passed to the callback method to identify which subscription was matched by a notification

exp: a string filter to be applied to the attributes of notifications sent over the underlying channel; a null value sets the attribute filter to *TRUE*

f: an object filter to be applied to the attributes of notifications sent over the underlying channel; a null value sets the attribute filter to *TRUE*

sub: a Subscriber object that implements a callback method for receiving matching notifications for the registered subscription

subjectFilter: a filter to be applied to the subject of notifications sent over the associated channel

ctrlSubscriber: An optional Subscriber class used to deliver control messages associated with a subscription

returns: a SubscriptionHandle object identifying the registered subscription for other operations

throws: *subscribe* throws an InvalidFilterException if a problem was encountered in attempting to register the subscription with the channel.

subscribe throws an InvalidSubjectException if a valid filtered subject is not found.

ctrlSubscribe

```
public SubscriptionHandle ctrlSubscribe(arguments)
```

The ctrl*Subscribe* method registers a subscription for internal system-level events. (See, also, CtrlNotification.)

The implementations of this method, and their variables, are listed below.

```
public SubscriptionHandle ctrlSubscribe(
    String subjectFilter,
    Subscriber sub,
    Object closure)

public SubscriptionHandle ctrlSubscribe(
    String subjectFilter,
    String exp,
```

```
    Subscriber sub,
    Object closure)

public SubscriptionHandle ctrlSubscribe(
    String subjectFilter,
    Filter f,
    Subscriber sub,
    Object closure)
```

closure: an object passed to the callback method to identify which subscription was matched by a notification

exp: a string filter to be applied to the attributes of notifications sent over the underlying channel; a null value sets the attribute filter to *TRUE*

f: an object filter to be applied to the attributes of notifications sent over the underlying channel; a null value sets the attribute filter to *TRUE*

sub: a Subscriber object that implements a callback method for receiving matching notifications for the registered subscription

subjectFilter: a filter to be applied to the subject of notifications sent over the underlying

returns: a SubscriptionHandle object identifying the registered subscription for other operations

throws: *ctrlSubscribe* throws an InvalidFilterException if a problem was encountered in attempting to register the subscription with the channel.

ctrlSubscribe throws an InvalidSubjectException if a valid filtered subject is not found.

pause

```
public void pause(SubscriptionHandle sh)
```

Halts delivery of matching notifications to the callback associated with a subscription.

Note: The operation is idempotent.

sh: a handle for the subscription to be paused

throws: *pause* throws an InvalidSubscriptionHandleException if the SubscriptionHandle does not identify a subscription currently registered with this object.

suspend

```
public void suspend(SubscriptionHandle sh)
```

Halts matching of notifications against a subscription.

Note: The operation is idempotent.

sh: a handle for the subscription to be suspended

throws: *suspend* throws an InvalidSubscriptionHandleException if the SubscriptionHandle does not identify a subscription currently registered with this object.

resume

```
public void resume(SubscriptionHandle sh)
```

Resumes matching and delivery of notifications against a previously paused or suspended subscription, including notifications received while the subscription was paused.

Note: The operation is idempotent.

sh: a handle for the subscription to be resumed

throws: *suspend* throws an InvalidSubscriptionHandleException if the SubscriptionHandle does not identify a subscription currently registered with this object.

unsubscribe

```
public void unsubscribe(SubscriptionHandle sh)
```

Unregisters a subscription from a channel.

sh: a handle for the subscription to be unsubscribed

throws: *suspend* throws an InvalidSubscriptionHandleException if the SubscriptionHandle does not identify a subscription currently registered with this object.

ctrlUnsubscribe

```
public void ctrlUnsubscribe(SubscriptionHandle sh)
```

Unregisters a control subscription from a channel.

sh: a handle for the subscription to be unsubscribed

throws: *ctrlUnsubscribe* throws an InvalidSubscriptionHandleException if the SubscriptionHandle does not identify a control subscription currently registered with this object.

getCtrlChannelInfo

```
public ChannelInfo getCtrlChannelInfo()
```

ChannelInfo objects contain attribute information for notifications.

returns: the control ChannelInfo needed for constructing filters for system events (See, also, CtrlNotification.)

getChannelInfo

```
public ChannelInfo getChannelInfo()
```

ChannelInfo objects contain attribute information for notifications.

returns: the ChannelInfo needed for constructing filters for data notifications

getFilterFactory

```
public FilterFactory getFilterFactory()
```

Creates an instance of the FilterFactory for creating Filter objects.

returns: a FilterFactory object that can be used for building Filter objects for data notifications

getCtrlFilterFactory

```
public FilterFactory getCtrlFilterFactory()
```

Creates an instance of the FilterFactory for creating Filter objects.

returns: a FilterFactory object that can be used for building Filter objects for control notifications

load

```
public static Proxy load(
    java.lang.String filename)
    throws EventException
```

Creates a Proxy object based on the state information stored on a file

filename: The path to the file containing the proxy state supposed to be restored in the new object

throws: *load* throws an EventException if the state cannot be restored from the saved file

save

```
public void save(
    java.lang.String filename)
    throws EventException
```

Saves the current state of the Proxy object in a given file

filename: The path of the file storing thesaved proxy state

throws: *save* throws an EventException if the state cannot be saved in the given file.

getSubscriptions

```
public java.util.Vector getSubscriptions()
```

Retrieves a list of the existing subscriptions

returns: The list of subscriptions handles of all the subsriptions currently associated with a proxy

setClientType

```
public static void setClientType(
    int idChannel,
    boolean bIsAllowedToPublish,
    boolean bIsAllowedToSubscribe)
    throws
    com.precache.channel.InvalidChannelException,
    EventException
```

Specifies whether the client is allowed to subscribe and/or publish on a specific channel

idChannel: the ID of the channel to be accessed

bIsAllowedToPublish: when set to *true* it enables the application to publish on the given channel

bIsAllowedToSubscribe: when set to *true* it enables the application to subscribe on the given channel

throws: *setClientType* throws an InvalidChannelException if the specified channel ID is invalid, or if the operation cannot be performed on the given channel

setClientType throws an EventException if an invalid combionation of flags is used (such as (*false*,*false*))

SimpleFilter

— filters for testing expressions

A SimpleFilter object stores an attribute filter expression based on attribute, name, operator, and operand. SimpleFilter performs a simple expression test and returns a Boolean containing the value of an attribute.

Hierarchy

java.lang.Object
↳com.precache.event.Filter
↳ com.precache.event.SimpleFilter

Methods

SimpleFilter inherits *clone*, *equals*, *getClass*, *hashCode*, *notify*, *notifyAll*, and *wait* from class java.lang.Object

toString

```
public string toString()
```

Converts the filtered object to a string representation.

overrides: *toString* in class Object

returns: the string expression for this object

finalize

```
public void finalize()
```

Frees up resources associated with this object.

overrides: *finalize* in class Object

match

```
public boolean match(Notification n)
```

Verifies whether a Notification matches the SimpleFilter object.

n: the notification to be matched

returns: `true`, if the notification matches; otherwise `false`

overrides: *match* in class Filter

clone

```
public SimpleFilter clone()
```

Constructs a clone of the SimpleFilter object.

overrides: *clone* in class Filter

Subscriber

— the interface for callback operations

A subscriber class that receives Notifications must implement interface Subscriber. The interface defines a callback method, *notify*, which is invoked on behalf of the subscriber by a Proxy object to delivermatching notifications.

Constructor

```
public interface Subscriber
```

Methods

notify

```
public void notify(
   Notification n,
   Object closure)
   throws Exception
```

Called to deliver matching notifications to the object implementing this interface.

n: the notification to be delivered

closure: an object passed to the Proxy's subscribe method to identify which subscription was matched by a notification

SubscriptionHandle

— the subscription identification class

A SubscriptionHandle object identifies a subscription that was registered with a Proxy object and can be used to have the Proxy perform various operations on the subscription.

Hierarchy

java.lang.Object
↳com.precache.event.SubscriptionHandle

Methods

SubscriptionHandle inherits *clone*, *equals*, *getClass*, *hashCode*, *notify*, *notifyAll*, *toString*, and *wait* from class java.lang.Object.

finalize

```
public void finalize()
```

Frees up resources associated with this object.

overrides: *finalize* in class Object

getAttrFilterString

```
public java.lang.String getAttrFilterString()
```

Retrives the string representation of the attribute filter associated with the subscription.

getSubjectFilter

```
public java.lang.String getSubjectFilter()
```

Retrives the string representation of the subject filter associated with the subscription.

getClosure

```
public java.lang.Object getClosure()
```

Retrieves a pointer to the closure associated with the current subscription.

getSubscriber

```
public Subscriber getSubscriber()
```

Retrieves a pointer to the `Subscriber` object associated with the current subscription

C++ Classes

— classes, functions, and exception handlers for the C++ deployment of the Event Agent

Class Hierarchy: *the Event Agent's C++ Class hierarchy*

Archive: *an abstraction for the class archive*

AndFilter: *a conjunction of attribute filter expressions*

ChannelInfo: *an abstraction for the channel configuration*

CompoundFilter: *an abstract class for aggregating filters*

CtrlNotification: *notifications for internal events*

Exception classes: *descriptions of the exception classes*

Filter: *an abstract class for attribute filters*

FilterFactory: *generate filters from expressions*

Notification: *the data for an event to be published*

OrFilter: *a disjunction of attribute filter expressions*

Proxy: *connectivity to a channel*

Serializable: *read and write data to disk*

SimpleFilter: *filters for testing expressions*

Subscriber: *the interface for receiving notifications*

SubscriptionHandle: *the subscription identification class*

Class Hierarchy

— the Event Agent's C++ Class hierarchy

Archive
Serializable
  ↳Filter
    ↳CompoundFilter
      ↳AndFilter
      ↳OrFilter
    ↳SimpleFilter
  ↳Subscriber
  ↳SubscriptionHandle
ChannelInfo
Notification
  ↳CtrlNotification
FilterFactory
Proxy
PC_Exception
  ↳EventException
    ↳InvalidFilterException
    ↳InvalidOperationException
    ↳InvalidSubjectException
    ↳InvalidSubscriptionHandleException
    ↳ProxyStateException
    ↳StateTransitionException
    ↳UnsupportedOperationException

AndFilter

— a conjunction of attribute filter expressions

AndFilter objects facilitate notification filtering by conjunctive expressions.

Hierarchy

precache::util::Serializable
↳precache::event::Filter
↳precache::event::CompoundFilter
↳precache::event::AndFilter

Constructors

AndFilter(const ChannelInfo& *rkChannelInfo*)

`AndFilter(const ChannelInfo& rkChannelInfo)`

Constructs an AndFilter object and associates it with a ChannelInfo object, providing access to a channel's attributes and validation against the associated channel's configuration.

rkChannelInfo **:** the ChannelInfo object to be associated with the AndFilter

AndFilter(const Proxy& *rkProxy*)

`AndFilter(const Proxy& rkProxy)`

Constructs an AndFilter object and associates it with a Proxy object, providing access to a channel's attributes and validation against the associated channel's configuration.

rkProxy **:** the proxy associated with the AndFilter

AndFilter(const AndFilter& *rkAndFilter*)

Constructs a copy of a AndFilter object that is a deep copy of the full tree of Filters rooted at the copied object.

`AndFilter(const AndFilter& rkAndFilter)`

rkAndFilter **:** the AndFilter to be copied

Destructor

`~AndFilter()`

Frees up resources that were allocated for this object.

Methods

Match

`virtual bool Match(Notification& rkNotification)`

Verifies whether a notification type matches the AndFilter object.

rkNotification: the notification to be matched

returns: `true`, if the notification matches; otherwise `false`

Clone

```
virtual Filter* Clone
```

Constructs a clone of the Filter object.

Note: We provide this virtual method rather than a public copy constructor because the cloning must be class-specific, and because many of the true classes are hidden from applications.

Archive

— an abstraction for the class archive

Archive provides the abstraction for storage during the serialization process.

Hierarchy

precache::util::Archive

Constructor

```
Archive(
   const PCSmallString& rkstrFileName,
   bool bOpenForRead)
   throw(precache::util::SerializationException)
```

Opens a class archive for reading or writting.

***rkstrFileName*:** the name of the file storing the archive

Methods

ReadBuf

```
void ReadBuf(
  void *buf,
  int cBufSize)
  throw (precache::util::SerializationException)
```

Reads *cBufSize* bytes of memory in the archive.

buf: a pointer to a memory buffer containing the data to be stored

cBufSize: the number of bytes to be read

throws: *ReadBuf* throws a SerializationException if the archive encounters an error while storing the data.

WriteBuf

```
void WriteBuf(
   const void *buf,
   int cBufSize)
   throw (precache::util::SerializationException)
```

Stores *cBufSize* bytes of memory in the archive.

buf: a pointer to a memory buffer containing the data to be stored

cBufSize: the number of bytes to be stored

throws: *WriteBuf* throws a SerializationException if the archive encounters an error while storing the data.

Read operators

```
Archive& operator>>(
   TypeName &val)
   throw (SerializationException)
```

Archive prodives a number of read operators for accessing archived data:

```
Archive& operator>>(
   precache::util::Serializable* &objVal)
   throw (precache::util::SerializationException)

Archive& operator>>(
   precache::util::Serializable &objVal)
   throw (precache::util::SerializationException)

Archive& operator>>(
   PCSmallString& rstrVal)
   throw (precache::util::SerializationException)

Archive& operator>>(
   PC_STRING &szVal)
   throw (precache::util::SerializationException)

template<class KeyType, class ElementType>
   Archive& operator>>(
   map<KeyType, ElementType> &rmpVal)
   throw (precache::util::SerializationException)

template<class ElementType>
   Archive& operator>>(
   set<ElementType> &rmpVal)
   throw (precache::util::SerializationException)

Archive& operator>>(
   PC_CHAR &val)
   throw (SerializationException)

Archive& operator>>(
   PC_UCHAR &val)
   throw (SerializationException)

Archive& operator>>(PC_SHORT &val)
   throw (SerializationException)

Archive& operator>>(
   PC_USHORT &val)
   throw (SerializationException)

Archive& operator>>(
   PC_INT &val)
   throw (SerializationException)

Archive& operator>>(
```

```
   PC_UINT &val)
   throw (SerializationException)

Archive& operator>>(
   PC_LONG &val)
   throw (SerializationException)

Archive& operator>>(
   PC_ULONG &val)
   throw (SerializationException)

Archive& operator>>(
   PC_DOUBLE &val)
   throw (SerializationException)

Archive& operator>>(
   PC_FLOAT &val)
   throw (SerializationException)
```

val: the value to be read

returns: a reference to the archive object

throws: *Archive* throws a SerializationException if the archive encounters an error while reading the data

Write operators

```
Archive& operator<<(
   const TypeName val)
   throw (SerializationException)
```

Archive prodives a number of write operators for storing archived data:

```
Archive& operator<<(
   precache::util::Serializable *&objVal)
   throw (precache::util::SerializationException)

Archive& operator<<(
   precache::util::Serializable &objVal)
   throw (precache::util::SerializationException)

Archive& operator<<(
   const PCSmallString& rkstrVal)

Archive& operator<<(
   const PC_STRING kszVal)

 template<class KeyType, class ElementType>
   Archive& operator<<(
   map<KeyType, ElementType> &rmpVal)
   throw (precache::util::SerializationException)

 template<class ElementType>
   Archive& operator<<(
   Set<ElementType> &rmpVal)
   throw (precache::util::SerializationException)

Archive& operator<<(
   const PC_CHAR val)
   throw (SerializationException)

Archive& operator<<(
   const PC_UCHAR val)
   throw (SerializationException)

Archive& operator<<(
   const PC_SHORT val)
   throw (SerializationException)

Archive& operator<<(
   const PC_USHORT val)
   throw (SerializationException)

Archive& operator<<(
   const PC_INT val)
   throw (SerializationException)

Archive& operator<<(
   const PC_UINT val)
   throw (SerializationException)
```

```
Archive& operator<<(
   const PC_LONG val)
   throw (SerializationException)

Archive& operator<<(
   const PC_ULONG val)
   throw (SerializationException)

Archive& operator<<(
   const PC_DOUBLE val)
   throw (SerializationException)

Archive& operator<<(
   const PC_FLOAT val)
   throw (SerializationException)
```

val: the value to be written

returns: a reference to the archive object

throws: *Archive* throws a SerializationException if the archive encounters an error while reading the data

ChannelInfo

— an abstraction for the channel configuration

ChannelInfo needs to be associated with various objects such as Filter or Notification.

ChannelInfo provides an abstraction for the channel configuration, consisting of the subject and attributes of a channel, which may be retrieved by a proxy. No methods are exposed.

CompoundFilter

— an abstract class for aggregating filters

CompoundFilter is used to aggregate filters into a single object. CompoundFilter serves as a base class for AndFilter and OrFilter objects.

Hierarchy

precache::util::Serializable
↳precache::event::Filter
↳precache::event::CompoundFilter

subclasses: AndFilter, OrFilter

Constructors

CompoundFilter(*ChannelInfo*)

```
CompoundFilter(
   const ChannelInfo& rkChannelInfo)
```

Constructs a CompoundFilter object and associates it with a ChannelInfo object, providing access to a channel's attributes and validation against the associated channel's configuration.

rkChannelInfo: the ChannelInfo object associated with the filter

CompoundFilter(*CompoundFilter*)

```
CompoundFilter(
   const CompoundFilter& rkCompoundFilter)
```

Constructs a copy of a CompoundFilter object by performing a deep copy of the Filter tree.

rkCompoundFilter: the CompoundFilter to be copied

Destructor

```
~CompoundFilter()
```

Frees up resources that were allocated for this object.

Methods

Add

```
void Add(Filter* pFilter)
   throw(InvalidFilterException)
```

Adds a subfilter to this object.

pFilter: a Filter object containing the added subfilter

throws: *Add* throws an InvalidFilterException if the subfilter has already been added.

Remove

```
void Remove(Filter* pFilter)
   throw(InvalidFilterException)
```

Removes a subfilter from the object.

pFilter: a Filter object subfilter to be removed

Reset

```
void Reset()
   throw(InvalidFilterException)
```

Removes all existing subfilters for the object.

SubFilters

```
rgpFilters_t& SubFilters
```

returns: *SubFilters* returns a vector object storing the constituent filters in the order in which they were added.

ToString

```
virtual const PC_CHAR* ToString()
```

Returns the filter object as a string representation.

CtrlNotification

— notifications for internal events

CtrlNotification delivers notifications for internal events such as an asynchronous exception or subscription confirmations.

Control notifications have a fixed attribute configuration for *name*, *message*, and *level*.

name: *PC_STRING*

message: *PC_STRING*

level: *PC_LONG*

CtrlNotification inherits all the Notification class functionality.

Hierarchy

precache::event::Notification
↳precache::event::CtrlNotification

Level attribute values

PC_INFO_LEVEL

The control notification has three attributes: name, message, and level. The level attribute has three possible values: *0* = *PC_INFO_LEVEL*; *1* = *PC_CTRL_LEVEL*; *2* = *PC_ERROR_LEVEL*.

```
const PC_UINT PC_INFO_LEVEL
```

PC_INFO_LEVEL control notifications deliver status updates and debugging information. They do not play an active role in API usage; their delivery should not affect the behavior of an application.

PC_CTRL_LEVEL

```
const PC_UINT PC_CTRL_LEVEL
```

PC_CTRL_LEVEL control notifications report asynchronous system events that might be relevant for application behavior. Applications should handle such events in order to rely on a deterministic behavior.

PC_ERROR_LEVEL

```
const PC_UINT PC_ERROR_LEVEL
```

PC_ERROR_LEVEL notifies the application of the occurrence of a critical asynchronous event such us an unhandled exception thrown by an internal Proxy thread.

Note: Failure to handle *PC_ERROR_LEVEL* triggers program termination.

Methods

GetName

```
PC_CHAR* GetName()
```

GetName retrieves the value of the *name* attribute of the control notification.

GetMessage

```
PC_CHAR* GetMessage()
```

GetMessage retrieves the value of the *message* attribute of the control notification.

GetLevel

```
PC_UINT GetLevel() const
```

GetLevel retrieves the value of the *level* attribute of the control notification. The possible values passed are:

0: PC_INFO_LEVEL

1: PC_CTRL_LEVEL

2: PC_ERROR_LEVEL

SetName

```
void SetName(const PC_CHAR* kszName)
```

SetName changes the value of the *name* attribute of the control notification.

SetMessage

```
void SetMessage(const PC_CHAR* kszMessage)
```

SetMessage changes the content of the *message* attribute of the control notification.

SetLevel

```
void SetLevel(PC_UINT unLevel)
```

SetLevel changes the severity level of the *level* attribute of the control notification.

Exception classes

— descriptions of the exception classes

This section describes all the NetInjector Event Agent exception classes.

PC_Exception

The base class for all exception classes. *PC_Exception* logs exceptions if the application implementing this class has been compiled with the macro *EXCEPTION_LOGGING* defined.

Hierarchy

precache::util::PC_Exception

Fields

***m_kstrMessage*:** the diagnostic message associated with this object.

Constructor

```
PCException (
   const PCSmallString& rkstrMessage,
   const PCSmallString& rkstrFile,
   const PC_INT nLine,
   PC_LogPriority kidLogPriority)
```

Constructs a *PC_Exception* object with a diagnostic message.

***rkstrMessage*:** the message for this object

***rkstrFile*:** the name of the file in which this object was thrown

***nLine*:** the line number within rkstrFile at which this object was thrown

***kidLogPriority*:** the priority level for logging; see logging API for more information

InvalidChannelException

An *InvalidChannelException* occurs if the application encounters a problem accessing a channel, whether due to an invalid channel or client ID, or because of network connection problems.

Hierarchy

precache::util::PC_Exception
↳precache::channel::InvalidChannelException

Methods

```
class PC_EXPORT InvalidChannelException
```

Constructs an InvalidChannelException object with a diagnostic message.

ChannelInfoException

Hierarchy

precache::util::PC_Exception
↳precache::event::EventException
↳precache::event::ChannelInfoException

Constructs an ChannelInfoException object with a diagnostic message.

```
class PC_EXPORT ChannelInfoException
```

EventException

Hierarchy

precache::util::PC_Exception
↳precache::event::EventException

subclasses: InvalidFilterException, InvalidNotificationException, InvalidOperationException, InvalidSubjectException, InvalidSubscriptionHandleException, ProxyStateException, StateTransitionException, UnsupportedOperationException

Constructor

```
EventException(
    const string& rkstrMessage,
    const string& rkstrFile,
    PC_INT nLine)
```

Constructs an EventException with a diagnostic message.

rkstrMessage: the message for this object

rkstrFile: the name of the file in which this object was thrown

nLine: the line number within *rkstrFile* at which this object was thrown

InvalidFilterException

An *InvalidFilterException* occurs if the application attempts to construct a Filter object with an invalid attribute filter expression.

Hierarchy

precache::util::PC_Exception
↳precache::event::EventException
↳precache::event::InvalidFilterException

Methods

Constructs an InvalidFilterException object with a diagnostic message.

```
class PC_EXPORT InvalidFilterException
```

InvalidNotificationException

An *InvalidNotificationException* occurs if the application attempts to construct a Notification object with attribute values inconsistent with the notification type of the associated channel.

Hierarchy

precache::util::PC_Exception
↳precache::event::EventException
↳precache::event::InvalidNotificationException

Methods

Constructs an InvalidNotificationException object with a diagnostic message.

```
class PC_EXPORT InvalidNotificationException
```

InvalidOperationException

An *InvalidOperationException* occurs if the application attempts to apply an invalid attribute values/test operation combination.

Hierarchy

precache::util::PC_Exception
↳precache::event::EventException
↳precache::event::InvalidOperationException

Methods

Constructs an InvalidOperationException object with a diagnostic message.

```
class PC_EXPORT InvalidOperationException
```

InvalidSubjectException

An *InvalidSubjectException* occurs if the application references a nonexistent subject in a Notification object or an invalid subject filter in a subscription, as determined by the associated channel.

Hierarchy

precache::util::PC_Exception
↳precache::event::EventException
↳precache::event::InvalidSubjectException

Methods

```
class PC_EXPORT InvalidSubjectException
```

Constructs an InvalidSubjectException object with a diagnostic message.

InvalidSubscriptionHandleException

An *InvalidSubscriptionHandleException* occurs if the application references an obsolete or invalid SubscriptionHandle object.

Hierarchy

precache::util::PC_Exception
↳precache::event::EventException
↳precache::event::InvalidSubscriptionHandleException

Methods

```
class PC_EXPORT InvalidSubscriptionHandleException
```

Constructs an InvalidSubscriptionHandleException object with a diagnostic message.

ProxyStateException

ProxyStateException is thrown by the Proxy state management routines.

Hierarchy

precache::util::PC_Exception
↳precache::event::EventException
↳precache::event::ProxyStateException

Methods

```
class PC_EXPORT ProxyStateException
```

Constructs a ProxyStateException object with a diagnostic message.

StateTransitionException

A *StateTransitionException* indicates that something went wrong while the Proxy was switching from one state to another.

Hierarchy

precache::util::PC_Exception
↳precache::event::EventException
↳precache::event::StateTransistionException

Methods

```
class PC_EXPORT StateTransitionException
```

Constructs an StateTransitionException object with a diagnostic message.

UnsupportedOperationException

An UnsupportedOperationException occurs if the application attempts to perform an operation not permitted under the current proxy state.

Hierarchy

precache::util::PC_Exception
↳precache::event::EventException
↳precache::event::UnsupportedOperationException

Methods

```
class PC_EXPORT UnsupportedOperationException
```

Constructs an UnsupportedOperationException object with a diagnostic message.

Filter

— an abstract class for attribute filters

`class Filter()`

Class Filter is the base class for all filters, including SimpleFilter and CompoundFilter objects. Filter objects provide access to a channel's attributes and validation against the associated channel's configuration.

Hierarchy

precache::util::Serializable
↳precache::event::Filter

subclasses: CompoundFilter, SimpleFilter

Destructor

`~Filter()`

Frees up resources that were allocated for the Filter object.

Methods

Match

`virtual bool Match(Notification& rkNotification)`

Verifies whether a Notification type matches the Filter object.

rkNotification: the notification to be matched

returns: `true`, if the notification matches; otherwise `false`

Clone

`virtual Filter* Clone()`

Constructs a clone of the Filter object.

We provide this virtual method rather than a public copy constructor because the cloning must be class-specific, and because many of the true classes are hidden from applications.

ToString

`virtual const PC_CHAR ToString()`

Converts the filtered object to a string representation.

FilterFactory

— generate filters from expressions

The FilterFactory is a singleton class that generates Filter objects from string attribute filter expressions.

Destructor

```
~FilterFactory()
```

Frees up resources associated with this object

Methods

CreateFilter

```
Filter* CreateFilter(const PC_CHAR* kszFilterExp)
    throw(InvalidFilterException)
```

Creates a tree of Filter objects from a string attribute filter expression and associates the objects with a ChannelInfo.

kszFilterExp: the attribute filter expression to be parsed

rkChannelInfo: the *ChannelInfo* with which the constructed Filter objects will be associated, providing access to a channel and validation against the associated channel's configuration

returns: a pointer to the newly-allocated tree of *Filter* objects

throws: *CreateFilter* throws an InvalidFilterException if there are syntactic or semantic errors in *kszFilterExp*.

Notification

— the data for an event to be published

A Notification object stores information about a single event that is to be published over a channel.

The object stores the event's subject and the attribute values for the associated channel.

Hierarchy

precache::event::Notification

subclasses: CtrlNotification

Constructors

Notification by Proxy

```
Notification(
    const Proxy& rkProxy,
    const PCSmallString& rkstrSubject)
    throw(InvalidSubjectException,
    InvalidNotificationException)
```

Constructs a notification object from a subject string and associates this notification with a ChannelInfo.

rkProxy: the *Proxy* with which this object will be associated, providing access to a channel and validation against the channel's subject tree and notification type

rkstrSubject: a subject for the notification

throws: *Notification* throws an InvalidSubjectException if the specified subject is invalid for the underlying channel.

Notification throws an InvalidNotificationException if there is a problem setting up the notification message.

Notification by ChannelInfo

```
Notification(
    const ChannelInfo& rkChannelInfo,
    const PCSmallString& rkstrSubject,
    const PC_CHAR* kszAttrVals)
    throw(InvalidSubjectException,
    InvalidNotificationException)
```

Constructs a Notification object from a subject string and associates this notification with a ChannelInfo.

rkChannelInfo: the ChannelInfo with which this object will be associated, providing access to a channel and validation against the channel's subject tree and notification type

rkstrSubject: a subject for the notification

kszAttrVals: a comma-separated sequence of attribute assignments of the form `name=(type)value`

throws: *Notification* throws an InvalidSubjectException if the specified subject is invalid for the channel.

Notification throws an InvalidNotificationException if the specified attribute assignments are syntactically incorrect or are inconsistent with the notification type of the channel, or if there is a problem setting up the notification message.

Notification copy

```
Notification(const Notification& rkNotification)
    throw(InvalidNotificationException)
```

Constructs a copy of a Notification object.

rkNotification: the notification to be copied

throws: *copy* throws an InvalidNotificationException if the attempt to copy was unsuccessful

Notification assignment

```
Notification& operator=(
    const Notification& rNotification)
    throw(InvalidNotificationException)
```

Assigns one Notification object to another.

rNotification: the notification to be assigned

throws: *assignment* throws an InvalidNotificationException if the attempt to assign was unsuccessful.

Destructor

```
~Notification()
```

Frees up resources that were allocated for this object.

Methods

Subject

```
const PCSmallString& Subject()
```

Returns the subject of the notification.

SubjectMapping

```
vector<PC_UINT16> SubjectMapping()
```

The integer mapping of the subject associated with the notification.

Print

```
void Print()
```

Prints a text rendering of the notification's contents on the standard output stream

SetPredefinedAttr — Generic

by name

```
virtual void SetPredefinedAttr(
   const PCSmallString& rkstrName,
   const void* pkVal,
   PC_TypeCode idTypeCode,
   PC_UINT unSizeIfArray = PC_ATTR_SIZE_ARG_DEFAULT)
   throw(InvalidNotificationException)
```

Sets the values of a specified attribute of the notification, identified by its name.

rkstrName: the name of the predefined attribute whose value is to be set

pkVal: an untyped pointer to a variable containing the value of the specified attribute (or the address of its first element, if it is a string or array value); the caller must ensure that the variable is of the correct type

idTypeCode: the type code of *pkVal*

unSizeIfArray: if *idTypeCode* designates an array type, then this parameter is the number of elements in the array name

throws: *setPredefinedAttr* throws an InvalidNotificationException if *rkstrName* is not the name of a predefined attribute, or if *ridTypeCode* is not the same as the type of the attribute as determined by the notification type of the channel.

by position

Sets the values of a specified attribute of the notification, identified by its position.

```
virtual void SetPredefinedAttr(
   PC_UINT unPos,
   const void* pkVal,
   PC_TypeCode idTypeCode,
   PC_UINT unSize = PC_ATTR_SIZE_ARG_DEFAULT)
   throw(InvalidNotificationException)
```

unPos: the position of the predefined attribute whose value is to be set, with respect to the sequence of predefined attributes defined by the notification type of the underlying channel, starting with 0

pkVal: the value to be assigned to the attribute

idTypeCode: the type code of *pkVal*

unSizeIfArray: if *idTypeCode* designates an array type, then this parameter is the number of elements in the array

throws: *SetPredefinedAttr* throws an InvalidNotificationException if there is no predefined attribute in the specified position, or if the type of *val* is not the same as the type of the attribute as determined by the notification type of the channel, or, in the case of floating-point types, if *val* is a NaN.

GetPredefinedAttr — Generic

by name

```
virtual void GetPredefinedAttr(
    const PCSmallString& rkstrName,
    void*& rpVal,
    PC_TypeCode idTypeCode)
    throw(InvalidNotificationException)

virtual void GetPredefinedAttr(
    const PCSmallString& rkstrName,
    void*& rpVal,
    PC_TypeCode idTypeCode,
    PC_UINT& runSizeIfArray)
    throw(InvalidNotificationException)
```

Returns the value of a predefined attribute of this object, identified by *name*. The first signature returns nonarrayed attributes values; the second signature returns an array of values.

rkstrName: the name of the predefined attribute whose value is to be returned

rpVal: variable to hold an untyped pointer to the value of the specified attribute, or for array-valued attributes to hold an untyped copy of the array pointer itself

idTypeCode: the type code of *rpVal*

runSizeIfArray: if *idTypeCode* designates an array type, then this parameter is the number of elements in the array name

returns: the value of a predefined attribute of this object, identified by its name

throws: *SetPredefinedAttr* throws an InvalidNotificationException if *rkstrName* is not the name of a predefined attribute, or if *ridTypeCode* is not the same as the type of the attribute as determined by the notification type of the channel.

by position

```
virtual void GetPredefinedAttr(
    PC_UINT unPos,
    void*& rpVal,
    PC_TypeCode idTypeCode)
    throw(InvalidNotificationException)

virtual void GetPredefinedAttr(
    PC_UINT unPos,
    void*& rpVal,
    PC_TypeCode idTypeCode,
    PC_UINT& runSizeIfArray)
    throw(InvalidNotificationException)
```

Returns the value of a predefined attribute of this object, identified by *position*. The first signature returns nonarrayed attributes values; the second signature returns an array of values.

unPos: the position of the predefined attribute whose value is to be returned, with respect to the sequence of predefined attributes defined by the notification type of the channel, starting with 0

rpVal: variable to hold an untyped pointer to the value of the specified attribute, or for array-valued attributes to hold an untyped copy of the array pointer itself

idTypeCode: the type code of *rpVal*

runSizeIfArray: if *idTypeCode* designates an array type, then this parameter is the number of elements in the array name

returns: an untyped pointer to the value of the specified attribute

throws: *GetPredefinedAttr* throws an InvalidNotificationException if there is no predefined attribute in the specified position.

SetPredefinedAttr — Typed

by name

```
void SetPredefinedAttr(
    const PC_SmallString& rkstrName,
    type val)
    throw(InvalidNotificationException)
```

```
void SetPredefinedAttr(
    const PC_SmallString& rkstrName,
    type* val,
    PC_UINT unSize)
    throw(InvalidNotificationException)
```

Sets the value of a predefined attribute of this object, identified by its name. The first signature sets nonarrayed attributes values; the second signature sets an array of values.

As delineated below, this method is overloaded on all allowed types for parameter *val*.

```
SetPredefinedAttr(
    const PC_SmallString& rkstrName,
    PC_CHAR val)
```

```
SetPredefinedAttr(
    const PC_SmallString& rkstrName,
    PC_LONG val)
```

```
SetPredefinedAttr(
    const PC_SmallString& rkstrName,
    PC_DOUBLE val)
```

```
SetPredefinedAttr(
    const PC_SmallString& rkstrName,
    PC_CHAR_ARRAY val)
```

```
SetPredefinedAttr(
    const PC_SmallString& rkstrName,
    PC_LONG_ARRAY val,
    PC_UINT unSize)
```

```
SetPredefinedAttr(
    const PC_SmallString& rkstrName,
    PC_DOUBLE_ARRAY val,
    PC_UINT unSize)
```

```
SetPredefinedAttrByteArray
    (const PCSmallString& rkstrName,
    const PC_BYTE_ARRAY val,
    PC_UINT unSize)
```

rkstrName: the name of the predefined attribute whose value is to be set

type: the data type of the attribute

val: an untyped pointer to a variable containing the value of the specified attribute (or the address of its first element, if it is a string or array value); the caller must ensure that the variable is of the correct type

unSize: the number of elements in *val*

throws: *SetPredefinedAttr* throws an InvalidNotificationException if *rkstrName* is not the name of a predefined attribute, or if the type of *val* is not the same as the type of the attribute as determined by the notification type of the channel, or, in the case of floating-point types, if *val* is a NaN.

by position

```
void SetPredefinedAttr(
   const PC_UINT unPos,
   type val)
   throw(InvalidNotificationException)

void SetPredefinedAttr(
   const PC_UINT unPos,
   type* val,
   PC_UINT unSize)
   throw(InvalidNotificationException)
```

Sets the value of a predefined attribute of this object, identified by its position. The first signature returns nonarrayed attributes values; the second signature returns an array of values.

As delineated below, this method is overloaded on all allowed types for parameter *val*.

```
SetPredefinedAttr(
   const PC_UINT unPos,
   PC_CHAR val)

SetPredefinedAttr(
   const PC_UINT unPos,
   PC_LONG val)

SetPredefinedAttr(
   const PC_UINT unPos,
   PC_DOUBLE val)

SetPredefinedAttr(
   const PC_UINT unPos,
   PC_CHAR_ARRAY val)

SetPredefinedAttr(
   const PC_UINT rkstrName,
   PC_LONG_ARRAY val,
   PC_UINT unSize)

SetPredefinedAttr(
   const PC_UINT rkstrName,
   PC_DOUBLE_ARRAY val,
   PC_UINT unSize)

SetPredefinedAttrByteArray
   (PC_UINT unPos,
   const PC_BYTE_ARRAY val,
   PC_UINT unSize)
```

unPos: the position of the predefined attribute whose value is to be set, with respect to the sequence of predefined attributes defined by the notification type of the underlying channel, starting with position 0

type: the data type of the attribute

val: the value to be assigned to the attribute

unSize: the number of elements in *val*

throws: *SetPredefinedAttr* throws an InvalidNotificationException if *rkstrName* is not the name of a predefined attribute, or if the type of *val* is not the same as the type of the attribute as determined by the notification type of the channel, or, in the case of floating-point types, if *val* is a NaN.

GetPredefinedAttr — Typed

by name

```
void GetPredefinedAttr(
   const PC_SmallString& rkstrName,
   type& rVal)
   throw(InvalidNotificationException)

void GetPredefinedAttr(
   const PC_SmallString& rkstrName,
   type* rVal,
   PC_UINT& runSize)
```

```
    throw(InvalidNotificationException)
```

Returns the value of a predefined attribute of this object, identified by its name. The first signature returns nonarrayed attributes values; the second signature returns an array of values.

As delineated below, this method is overloaded on all allowed types for parameter *val*.

```
GetPredefinedAttr(
    const PC_SmallString& rkstrName,
    PC_CHAR& rVal)

GetPredefinedAttr(
    const PC_SmallString& rkstrName,
    PC_LONG& rVal)

GetPredefinedAttr(
    const PC_SmallString& rkstrName,
    PC_DOUBLE& rVal)

GetPredefinedAttr(
    const PC_SmallString& rkstrName,
    PC_CHAR_ARRAY& rVal)

GetPredefinedAttr(
    const PC_SmallString& rkstrName,
    PC_LONG_ARRAY rVal,
    PC_UINT& runSize)

GetPredefinedAttr(
    const PC_SmallString& rkstrName,
    PC_DOUBLE_ARRAY rVal,
    PC_UINT& runSize)

GetPredefinedAttrByteArray
    (const PCSmallString& rkstrName,
    PC_BYTE_ARRAY& rVal,
    PC_UINT& runSize)
```

rkstrName: the name of the predefined attribute whose value is to be returned

type: the data type of the attribute

rVal: the value of the specified attribute

runSize: the number of elements in *rVal*

throws: *GetPredefinedAttr* throws an InvalidNotificationException if *rkstrName* is not the name of a predefined attribute, or if the type of val is not the same as the type of the attribute as determined by the notification type of the channel.

by position

```
void GetPredefinedAttr(
    const PC_UINT unPos,
    type& rVal)
    throw(InvalidNotificationException)

void GetPredefinedAttr(
    const PC_UINT unPos,
    type* rVal)
    throw(InvalidNotificationException)
```

Returns the value of a predefined attribute of this object, identified by its position. The first signature returns nonarrayed attributes values; the second signature returns an array of values.

As delineated below, this method is overloaded on all allowed types for parameter *val*.

```
GetPredefinedAttr(
    const PC_UINT unPos,
    PC_CHAR& rVal)

GetPredefinedAttr(
    const PC_UINT unPos,
    PC_LONG& rVal)
```

```
GetPredefinedAttr(
   const PC_UINT unPos,
   PC_DOUBLE& rVal)

GetPredefinedAttr(
   const PC_UINT unPos,
   PC_CHAR_ARRAY& rVal)

GetPredefinedAttr(
   const PC_UINT unPos,
   PC_LONG_ARRAY& rVal,
   PC_UINT& runSize)

GetPredefinedAttr(
   const PC_UINT unPos,
   PC_DOUBLE_ARRAY& rVal,
   PC_UINT& runSize)

GetPredefinedAttrByteArray
   (PC_UINT unPos,
   PC_BYTE_ARRAY& rVal,
   PC_UINT& runSize)
```

unPos: the position of the predefined attribute whose value is to be returned, with respect to the sequence of predefined attributes defined by the notification type of the channel, starting with 0

type: the data type of the attribute

rVal: the value of the specified attribute

runSize: the number of elements in *rVal*

throws: *GetPredefinedAttr* throws an InvalidNotificationException if *rkstrName* is not the name of a predefined attribute, or if the type of *val* is not the same as the type of the attribute as determined by the notification type of the channel.

AddDiscretionaryAttr — Generic

```
virtual void AddDiscretionaryAttr(
   const PC_SmallString& rkstrName,
   const void* pkVal,
   PC_TypeCode idTypeCode,
   PC_UINT unSizeIfArray = PC_ATTR_SIZE_ARG_DEFAULT)
   throw(InvalidNotificationException)
```

Sets the value of a discretionary attribute of this object.

rkstrName: the name of the discretionary attribute whose value is to be set

pkVal: an untyped pointer to a variable containing the value of the specified attribute (or the address of its first element, if it is a string or array value)

Note: the caller must ensure that the variable is of the correct type

idTypeCode: the type code of *pkVal*

unSizeIfArray: if *idTypeCode* designates an array type, then this parameter is the number of elements in the array

throws: *AddDiscretionaryAttr* throws an InvalidNotificationException if *rkstrName* belongs to a predefined attribute.

AddDiscretionaryAttr — Typed

```
void AddDiscretionaryAttr(
   const PC_SmallString& rkstrName,
   type val)
   throw(InvalidNotificationException)

void AddDiscretionaryAttr(
   const PC_SmallString& rkstrName,
```

```
    const type* val,
    PC_UINT unSize)
    throw(InvalidNotificationException)
```

Sets the value of a discretionary attribute of this object, identified by its name. The first signature sets nonarrayed attributes values; the second signature sets an array of values.

As delineated below, this method is overloaded on all allowed types for parameter *val*.

```
AddDiscretionaryAttr(
    const PC_SmallString& rkstrName,
    PC_CHAR val)

AddDiscretionaryAttr(
    const PC_SmallString& rkstrName,
    PC_LONG val)

AddDiscretionaryAttr(
    const PC_SmallString& rkstrName,
    PC_DOUBLE val)

AddDiscretionaryAttr(
    const PC_SmallString& rkstrName,
    PC_LONG val)

AddDiscretionaryAttr(
    const PC_SmallString& rkstrName,
    PC_CHAR_ARRAY val)
```

Note: This method sets an array of attribute values, without specifying the number of elements in parameter *val*.

```
AddDiscretionaryAttr(
    const PC_SmallString& rkstrName,
    PC_LONG_ARRAY val,
    PC_UINT unSize)

AddDiscretionaryAttr(
    const PC_SmallString& rkstrName,
    PC_DOUBLE_ARRAY val,
    PC_UINT unSize)

AddDiscretionaryAttr(
    const PC_SmallString& rkstrName,
    PC_BYTE_ARRAY val,
    PC_UINT unSize)
```

rkstrName: the name of the discretionary attribute whose value is to be set

type: the data type of the attribute

val: the value of the specified attribute

unSize: the number of elements in *val*

throws: *AddDiscretionaryAttr* throws an InvalidNotificationException if *rkstrName* belongs to a predefined attribute, or, in the case of floating-point types, if *val* is a NaN.

GetDiscretionaryAttr

```
void GetDiscretionaryAttr(
    const PC_SmallString& rkstrName,
    void*& rpVal,
    PC_TypeCode& ridTypeCode,
    PC_UINT& runSizeIfArray)
    throw(InvalidNotificationException)
```

GetDiscretionaryAttr returns the value of a discretionary attribute of this object. It is the only method for retrieving discretionary attribute values.

GetDiscretionaryAttr is not overloaded by attribute value type because notification recipients are not guaranteed to have *a priori* awareness of discretionary attribute characteristics.

rkstrName: the name of the discretionary attribute whose value is to be returned

rpVal: variable to hold an untyped pointer to the value of the specified attribute, or for array-valued attributes to hold an untyped copy of the array pointer itself

ridTypeCode: the type code of *rpVal*

runSizeIfArray: if *idTypeCode* designates an array type, then this parameter is the number of elements in the array

throw: *GetDiscretionaryAttr* throws an InvalidNotificationException if *rkstrName* is not the name of a discretionary attribute, or if there is a a problem retrieving the attribute.

DiscretionaryAttrNames

```
const rgstrNames_t DiscretionaryAttrNames()
    throw(InvalidNotificationException)
```

rgstrNames_t: the data type of the return

rkstrNames_t: an array of strings, containing discretionary attribute names

returns: the names of all discretionary attributes of this object.

throws: *DiscretionaryAttrNames* throws an InvalidNotificationException if there is a problem retrieving the discretionary attributes

IsDiscretionaryAttr

```
bool IsDiscretionaryAttr(const string& rkstrName)
```

Indicates whether this object has a discretionary attribute of a particular name.

rkstrName: the discretionary attribute name to check

returns: *true* if *rkstrName* is the name of a discretionary attribute of this object; otherwise, *false*

OrFilter

— a disjunction of attribute filter expressions

OrFilter objects facilitate notification filtering by conjunctive expressions.

Hierarchy

precache::util::Serializable
↳precache::event::Filter
↳precache::event::CompoundFilter
↳precache::event::OrFilter

Constructors

OrFilter by ChannelInfo

```
OrFilter(const ChannelInfo& rkChannelInfo)
```

Constructs an OrFilter object and associates it with a ChannelInfo object, providing access to a channel's attributes and validation against the associated channel's configuration.

rkChannelInfo: the ChannelInfo object to be associated with the OrFilter

OrFilter by Proxy

```
OrFilter(const Proxy& rkProxy)
```

Constructs an OrFilter object and associates it with a Proxy object, providing access to a channel's attributes and validation against the associated channel's configuration.

rkProxy **:** the proxy associated with the OrFilter

OrFilter copy

Constructs a copy of a OrFilter object that is a deep copy of the full tree of Filters rooted at the copied object.

```
OrFilter(const OrFilter& rkOrFilter)
```

rkOrFilter **:** the OrFilter to be copied

Destructor

```
~OrFilter()
```

Frees up resources that were allocated for this object.

Methods

Match

```
virtual bool Match(Notification& rkNotification)
```

Verifies whether a notification type matches the OrFilter object.

rkNotification: the notification to be matched

returns: `true`, if the notification matches; otherwise `false`

Clone

```
virtual Filter* Clone
```

Constructs a clone of the Filter object.

Note: We provide this virtual method, rather than a public copy constructor, because the cloning must be class-specific, and because many of the true classes are hidden from applications.

Proxy

— connectivity to a channel

A Proxy object provides connectivity to a channel and validation of notifications against the channel's Notification type.

Proxy publishes notifications for the channel's subjects and delivers matching notifications to subscriber's via subscriber callbacks.

Hierarchy

precache::util::Serializable
↳precache::event::Proxy

Constructor

```
Proxy(
    PC_UINT idChannel,
    PC_UINT idClient)
    throw(InvalidChannelException,
    precache::util::ThreadException, EventException)
```

Constructs a Proxy object for a specified channel and publisher.

idChannel: the ID of the channel to be accessed

idClient: the ID of the application using this object

throws: *Proxy* throws an InvalidChannelException if the specified channel ID is invalid, or if the specified publisher ID is not allowed to access the channel.

Destructor

```
~Proxy()
```

Frees up resources allocated to this object.

Methods

Publish

```
void Publish(
   const Notification& rkNotification)
   throw(InvalidNotificationException)
```

Publishes a notification over the associated channel.

rkNotification: the Notification object to be delivered

throws: Proxy throws an InvalidNotificationException if the Notification object is incomplete or otherwise invalid for publication over the channel.

Subscribe

```
SubscriptionHandle Subscribe(arguments)
   throw(InvalidFilterException,
   InvalidSubjectException)
```

Registers a subscription for the associated channel.

This method may be invoked in several ways, providing a wide range of return values and facilitating finely-tuned filtering.

The various implementations of this method, and their variables, are listed below.

```
SubscriptionHandle Subscribe(
   const PCSmallString& rkstrSubjectFilter,
   Subscriber* pSubcriber,
   void* pClosure,
   Subscriber* pCtrlSubscriber = PC_NULL)
```

```
SubscriptionHandle Subscribe(
   const PCSmallString& rkstrSubjectFilter,
   const PC_CHAR* kszAttrFilter,
   Subscriber* pSubcriber,
   void* pClosure,
   Subscriber* pCtrlSubscriber = PC_NULL)
```

```
SubscriptionHandle Subscribe(
   const PCSmallString& rkstrSubjectFilter,
   Filter* pAttrFilter,
   Subscriber* pSubcriber,
   void* pClosure,
   Subscriber* pCtrlSubscriber = PC_NULL)
```

pClosure: an object passed to the callback method to identify which subscription was matched by a notification

pCtrlSubscriber: An optional Subscriber class used to deliver control messages associated with a subscription

pAttrFilter: an object filter to be applied to the attributes of notifications sent over the channel; a null value sets the attribute filter to *TRUE*

pSubscriber: a Subscriber object that implements a callback method for receiving matching notifications for the registered subscription

rkstrSubjectFilter: a filter to be applied to the subject of notifications sent over the channel

kszAttrFilter: a string filter to be applied to the attributes of notifications sent over the channel; a null value sets the attribute filter to *true*

returns: a SubscriptionHandle object identifying the registered subscription for other operations

throws: *Subscribe* throws an InvalidFilterException if a problem was encountered attempting to register the subscription with the channel.

Subscribe throws an InvalidSubjectException if the specified subject filter is syntactically incorrect or invalid for the channel.

CtrlSubscribe

```
SubscriptionHandle CtrlSubscribe(arguments)
    throw(InvalidFilterException,
    InvalidSubjectException)
```

The ctrl*Subscribe* method registers a subscription for internal system-level events. (See, also, CtrlNotification.)

This method may be invoked in several ways, providing a range of return values and facilitating finely-tuned filtering.

CtrlSubscribe throws an InvalidSubjectException if the specified subject filter is syntactically incorrect or invalid for the channel.

The implementations of this method, and their variables, are listed below.

```
SubscriptionHandle CtrlSubscribe(
    const PCSmallString& rkstrSubjectFilter,
    Subscriber* pSubcriber,
    void* pClosure)
```

```
SubscriptionHandle CtrlSubscribe(
    const PCSmallString& rkstrSubjectFilter,
    const PC_CHAR* kszAttrFilter,
    Subscriber* pSubcriber,
    void* pClosure)
```

```
SubscriptionHandle CtrlSubscribe(
    const PCSmallString& rkstrSubjectFilter,
    Filter* pAttrFilter,
    Subscriber* pSubcriber,
    void* pClosure)
```

pClosure: an object passed to the callback method to identify which subscription was matched by a notification

pSubscriber: a Subscriber object that implements a callback method for receiving matching notifications for the registered subscription

pAttrFilter: an object filter to be applied to the attributes of notifications sent over the channel; a null value sets the attribute filter to *TRUE*

rkstrSubjectFilter: a filter to be applied to the subject of notifications sent over the underlying channel

kszAttrFilter: a string filter to be applied to the attributes of notifications sent over the channel; a null value sets the attribute filter to true

returns: a SubscriptionHandle object identifying the registered subscription for other operations

throws: *CtrlSubscribe* throws an InvalidFilterException if a problem was encountered in attempting to register the subscription with the channel.

Pause

```
void Pause(
    SubscriptionHandle& rhSubscription)
    throw(InvalidSubscriptionHandleException)
```

Halts delivery of matching notifications to the callback associated with a subscription.

Note: The operation is idempotent.

rhSubscription: a handle for the subscription to be paused

throws: *Pause* throws an InvalidSubscriptionHandleException if the SubscriptionHandle does not identify a subscription currently registered with this object.

Suspend

```
void Suspend(
    SubscriptionHandle& rhSubscription)
    throw(InvalidSubscriptionHandleException)
```

Halts matching of notifications against a subscription.

Note: The operation is idempotent.

rhSubscription: a handle for the subscription to be suspended

throws: *Suspend* throws an InvalidSubscriptionHandleException if the SubscriptionHandle does not identify a subscription currently registered with this object.

Resume

```
void Resume(
    SubscriptionHandle& rhSubscription)
    throw(InvalidSubscriptionHandleException)
```

Resumes matching and delivery of notifications against a previously paused or suspended subscription, including notifications received while the subscription was paused.

Note: The operation is idempotent.

rhSubscription: a handle for the subscription to be resumed

throws: *Suspend* throws an InvalidSubscriptionHandleException if the SubscriptionHandle does not identify a subscription currently registered with this object.

Unsubscribe

```
void Unsubscribe(
    SubscriptionHandle& rhSubscription)
    throw(InvalidSubscriptionHandleException)
```

Unregisters a subscription from a channel.

rhSubscription: a handle for the subscription to be unsubscribed

throws: *Unsubscribe* throws an InvalidSubscriptionHandleException if the SubscriptionHandle does not identify a subscription currently registered with this object.

CtrlUnsubscribe

```
void CtrlUnsubscribe(
    SubscriptionHandle& rhSubscription)
    throw(InvalidSubscriptionHandleException)
```

Unregisters a subscription from a channel.

rhSubscription: a handle for the subscription to be unsubscribed

throws: *CtrlUnsubscribe* throws an InvalidSubscriptionHandleException if the SubscriptionHandle does not identify a subscription currently registered with this object.

GetCtrlChannelInfo

```
ChannelInfo* GetCtrlChannelInfo() const
```

ChannelInfo objects contain attribute information for notifications.

returns: the ChannelInfo needed for the data channel underlying this object.

GetChannelInfo

```
ChannelInfo* GetChannelInfo() const
```

ChannelInfo objects contain attribute information for notifications.

returns: the ChannelInfo needed for the data channel underlying this object.

GetFilterFactory

```
FilterFactory* GetFilterFactory()
```

Creates an instance of the FilterFactory for creating Filter objects.

returns: a FilterFactory object that can be used for building Filter objects for data Notifications

GetCtrlFilterFactory

```
FilterFactory* GetCtrlFilterFactory()
```

Creates an instance of the FilterFactory for creating Filter objects.

hSubscription: The subscription handle for which data is retrieved

unMinutes: the number of minutes into the past from which cached notifications are to be retrieved

returns: a FilterFactory object that can be used for building Filter objects for Control Notifications

Save

```
void Save(
    const PC_CHAR* filename)
    throw(precache::util::SerializationException)
```

Saves the current state of the Proxy object in a given file. The subscribe and closure objects must be serializable.

filename: The path of the file where the proxy state would be saved

throws: Save throws a SerializationException is thrown if the state cannot be saved in the given file.

Load

```
static Proxy* Load(
    PC_STRING filename)
    throw(precache::util::SerializationException)
```

Creates a Proxy object based on the state information stored in a file.

filename: The path to the file containing the proxy state supposed to be restored in the new object

throws: *Load* throws a SerializationException if the state cannot be restored from the given file.

GetPredefinedAttrNames

```
void GetPredefinedAttrNames(
   rgstrNames_t &rgstrAttrNames)
```

Retrieves a list of predefined attributes names

rgstrNames_t: the data type of the return

rgstrAttrNames: A string array containing the names of the predefined attributes

The position of each attribute name in the array is the same with the attribute position in the channel configuration.

GetPredefinedAttrTypes

Retrieves a list of predefined attributes types.

```
void GetPredefinedAttrTypes(
   rgAttrTypeCode_t &rgAttrTypeCode)
```

rgAttrTypeCode_t: the data type of the return

rgAttrTypeCode: An array containing the type codes of the predefined attributes

The position of each attribute type code in the array is the same with the attribute position in the channel configuration.

SetStorageFileName

```
void SetStorageFileName(
   PC_STRING szFileName)
```

Defines the file name where the proxy state should be stored upon shutdown.

szFileName: The paths to the file where the proxy state should be saved

List

```
List <SubscriptionHandle> GetSubscriptions()
```

Retrieves a list of the existing subscriptions.

returns: the list of subscriptions handles of all the subsriptions currently associated with a proxy

Serializable

— read and write data to disk

Serializable provides a base class for all the classes that need to be serialized using precache serialization mechanism.

Hierarchy

precache::util::Serializable

Macros

DECLARE_SERIALIZABLE

`DECLARE_SERIALIZABLE(`*`ClassName`*`)`

DECLARE_SERIALIZABLE should be invoked within any *Serializable* class definition in order to define the class members needed for serialization.

IMPLEMENT_SERIALIZABLE

`IMPLEMENT_SERIALIZABLE(`*`ClassNameSpace,ClassName`*`)`

`SIMPLE_IMPLEMENT_SERIALIZABLE(`*`ClassName`*`)`

These two implementation macros should be invoked within the class implementation file, which contains the class method implementation, in order to initialize the class members used for serialization.

Methods

GetName

```
virtual const PCSmallString GetName()
```

Returns the class name of the current object. *GetName* is implicitly defined by DECLARE_SERIALIZABLE macro.

Serialize

```
virtual void Serialize(Archive &arch)
   throw(precache::util::SerializationException)
```

Serializes an object to or from an archive. *Serialize* must be implicitly redefined by each serilizable class.

arch **:** Represents the serialization archive

SimpleFilter

— filters for testing expressions

A SimpleFilter object stores an attribute filter expression based on attribute, name, operator, and operand. SimpleFilter performs a simple expression test and returns a Boolean containing the value of an attribute.

Hierarchy

precache::util::Serializable
↳precache::event::Filter
↳ precache::event::SimpleFilter

Constructors

by ChannelInfo

for predefined attributes

```
SimpleFilter(
    const ChannelInfo& rkChannelInfo,
    const PCSmallString& rkstrAttrIdent,AttrTestOp_t
    idOp,
    PC_TypeCode idAttrType,
    const PCSmallString& kstrAttrName,
    PC_UINT unAttrPos,
    PC_CHAR operand)
    throw(InvalidFilterException)
```

Creates a new, parsed SimpleFilter object for a test of a single predefined attribute and associates this object with a ChannelInfo.

Note: The application must ensure that the supplied attribute type code, name and position are valid for the channel defined by ChannelInfo.

rkChannelInfo: the ChannelInfo with which this object will be associated, providing access to a channel and validation against the associated channel's configuration

rkstrAttrIdent: the string expression used to identify the attribute tested by this object

idOp: the opcode for the test operator

idAttrType: the typecode of the predefined attribute being tested

kstrAttrName: the name of the predefined attribute being tested

unAttrPos: the position of the predefined attribute being tested

operand: the operand value

throws: SimpleFilter throws an InvalidFilterException if the requested combination of operator, attribute type, and operand type is invalid or if the attribute position is too large for the current marshaling scheme.

for discretionary attributes

```
SimpleFilter(
    const ChannelInfo& rkChannelInfo,
    AttrTestOp_t idOp,
    const PCSmallString& kstrAttrName,
    PC_CHAR operand)
```

Creates a new, parsed SimpleFilter object for a test of a single discretionary attribute and associates this object with a ChannelInfo.

rkChannelInfo: the ChannelInfo with which this object will be associated, providing access to a channel and validation against the associated channel's configuration

idOp: the opcode for the test operator

kstrAttrName: the name of the discretionary attribute being tested

operand: the operand value

by Proxy

for predefined attributes

```
SimpleFilter(
    const ProxyInfo& rkProxyInfo,
    const PCSmallString& rkstrAttrIdent,
    AttrTestOp_t idOp,
    PC_TypeCode idAttrType,
    const PCSmallString& kstrAttrName,
    PC_UINT unAttrPos,
    PC_CHAR operand)
    throw(InvalidFilterException)
```

Creates a new, parsed SimpleFilter object for a test of a single predefined attribute and associates this object with a Proxy.

Note: The application must ensure that the supplied attribute type code, name and position are valid for the channel defined by Proxy.

rkChannelInfo: the Proxy with which this object will be associated, providing access to a channel and validation against the associated channel's configuration

rkstrAttrIdent: the string expression used to identify the attribute tested by this object

idOp: the opcode for the test operator

idAttrType: the typecode of the predefined attribute being tested

kstrAttrName: the name of the predefined attribute being tested

unAttrPos: the position of the predefined attribute being tested

operand: the operand value

throws: SimpleFilter throws an InvalidFilterException if the requested combination of operator, attribute type, and operand type is invalid or if the attribute position is too large for the current marshaling scheme.

for discretionary attributes

```
SimpleFilter(
    const Proxy& rkProxy,
    AttrTestOp_t idOp,
    const PCSmallString& kstrAttrName,
    PC_CHAR operand)
```

Creates a new, parsed SimpleFilter object for a test of a single discretionary attribute and associates this object with a Proxy.

rkChannelInfo: the Proxy with which this object will be associated, providing access to a channel and validation against the associated channel's configuration

idOp: the opcode for the test operator

kstrAttrName: the name of the discretionary attribute being tested

operand: the operand value

Methods

Match

```
virtual bool Match(Notification& rkNotification)
```

Verifies whether a notification type matches the OrFilter object.

rkNotification: the notification to be matched

returns: `true`, if the notification matches; otherwise `false`

Clone

```
virtual Filter* Clone
```

Constructs a clone of the Filter object.

We provide this virtual method rather than a public copy constructor because the cloning must be class-specific, and because many of the true classes are hidden from applications.

ToString

```
virtual string ToString()
```

Converts the filtered object to a string representation.

OperatorID

```
AttrTestOp_t OperatorID() const
```

returns: the typecode ID of the operator

AttributeTypeCode

```
PC_TypeCode AttributeTypeCode() const
```

returns: the typecode of the attribute filtered by this object if a predefined attribute, or the typecode of the operand of this object if a discretionary attribute.

OperandTypeCode

```
PC_TypeCode OperandTypeCode() const
```

returns: the typecode of the operand

OperandValue

```
void* OperandValue() const
```

returns: an untyped pointer to the value of the operand of this object, which for an operand of type PC_STRING is a pointer to the string pointer, not the string pointer itself.

AttributePosition

```
PC_UINT AttributePosition() const
    throw(InvalidFilterException)
```

returns: the position of the predefined attribute tested by this object

throws: *AttributePosition* throws an InvalidFilterException if this object is not a test on a predefined attribute.

AttributeName

```
const string& AttributeName() const
```

returns: for discretionary attributes, a reference to the attribute name; for predefined attributes, a string representation of the position of the attribute (such as $3)

TestsDiscretionaryAttr

```
bool TestsDiscretionaryAttr() const
```

Indicates whether or not this object tests a discretionary attribute.

returns: `true`, if this object tests a discretionary attribute; otherwise, `false`

Subscriber

— the interface for receiving notifications

A subscriber class that receives Notifications must implement interface Subscriber. The interface defines a callback method, *notify*, which is invoked on behalf of the subscriber by a Proxy object to delivermatching notifications.

Hierarchy

precache::util::Serializable
↳precache::event::Subscriber

Desctructor

```
~Subscriber()
```

Frees up resources associated with this object

Methods

Notify

```
virtual void Notify(
   Notification* pNotification,
   precache::util::Serializable* pClosure)
   throw(PCException
```

```
virtual void Notify(
   Notification* pNotification,
```

```
    void* pClosure)
    throw(PCException)
```

Called to deliver matching notifications to the object implementing this interface. Implement the serializable method to save the notification to disk.

pNotification: a pointer to a newly-allocated object containing the notification to be delivered

pClosure: an object passed to the Proxy's Subscribe method to identify which subscription was matched by a notification

SubscriptionHandle

— the subscription identification class

A SubscriptionHandle object identifies a subscription that was registered with a Proxy object and can be used to have the Proxy perform various operations on the subscription.

Clients cannot create these objects directly on their own, but instead must have them created as a result of calls to Proxy::Subscribe.

Hierarchy

precache::util::Serializable
↳precache::event::SubscriptionHandle

Constructors

SubscriptionHandle

This default constructor creates an empty SubscriptionHandle which is not associated with any Proxy or registered with a particular subscription.

This constructor is useful for situations requiring default initialization of elements (such as arrays and STL containers).

```
SubscriptionHandle()
```

Copy

Creates a copy of a SubscriptionHandle object in a safe manner, preventing duplication of handled elements.

```
SubscriptionHandle(
    const SubscriptionHandle& rkhSubscription)
```

rkhSubscription: the SubscriptionHandle to be copied

Destructor

~SubscriptionHandle()

Frees up resources allocated for this object.

Note: While this object can be copied, the destruction of the last remaining instance of the original object also automatically unsubscribes the object's registration of its associated Subscription with its Proxy.

```
~SubscriptionHandle()
```

Methods

Operator=

Assigns one SubscriptionHandle object to another in a safe manner, preventing duplication of handled elements.

```
SubscriptionHandle& Operator=(const
    SubscriptionHandle& rkhSubscription)
```

rkhSubscription: the SubscriptionHandle to be assigned

returns: the assigned SubscriptionHandle

Operator==

```
bool operator==(const SubscriptionHandle&
    rkhSubscription)
```

Compares two SubscriptionHandle objects for equality.

rkhSubscription: the SubscriptionHandle against which this object is to be compared

returns: `true`, if this object equals *rkhSubscription*; otherwise, `false`

operator!=

```
bool operator!=(
    const SubscriptionHandle& rkhSubscription)
```

Compares two SubscriptionHandle objects for inequality.

rkhSubscription: the SubscriptionHandle against which this object is to be compared

returns: `true`, if this object does not equal *rkhSubscription*; otherwise, `false`

GetAttrFilterString

```
const PC_CHAR* GetAttrFilterString ()
```

Retrives the string representation of the attribute filter associated with the subscription

GetSubjectFilter

```
PCSmallString GetSubjectFilter()
```

Retrives the string representation of the subject filter associated with the subscription

GetClosure

```
void* GetClosure()
```

Retrieves a pointer to the cloasure associated with the current subscription

GetSubscriber

```
const Subscriber* GetSubscriber()
```

Retrieves a pointer to the Subscriber object associated with the current subscription

C Types and Functions

— *types and functions for the C deployment of the Event Agent*

Filter: *store and create filters*

Notification: *the data for an event to be published*

Proxy: *connectivity to a channel*

Subscriber: *callback functions*

Subscription handlers: *a subscription interface*

Filter

— store and create filters

Filters provide access to a channel's attributes and validation against the associated channel's configuration.

Type

```
typedef struct PC_evn_SFilter
```

Functions

PC_evn_create_filter

```
PC_Status PC_EXPORT PC_evn_create_filter(
    PC_evn_SFilter* pSAttrFilter,
    PC_evn_Proxy proxy,
    PC_CHAR* szAttrFilter,
    PC_BOOL bIsControl)
```

Creates a filter tree from a string attribute filter expression and associates the tree nodes with a proxy.

pSAttrFilter: a pointer to a variable to hold the newly-allocated filter tree

proxy: the proxy with which the constructed filter tree will be associated, providing access to a channel and validation against the associated channel's configuration

szAttrFilter: the attribute filter expression to be parsed

bIsControl: an indication of whether or not this filter will be used to filter control notifications

returns: *PC_INVALID_FILTER*, if there are syntactic or semantic errors in szAttrFilter (in which case *pSAttrFilter* is set to *PC_NULL*); otherwise *PC_SUCCESS*

PC_evn_create_compound_filter

```
PC_Status PC_EXPORT PC_evn_create_compound_filter(
    PC_evn_SFilter* pSCompoundFilter,
    PC_evn_Filter_Kind idKind,
    PC_evn_Channel_Info channel)
```

Creates a compound filter and associates it with a channel.

pSCompoundFilter: a pointer to a variable to hold the newly-allocated compound filter

idKind: the kind of this compound filter, either *PC_AND_FILTER* or *PC_OR_FILTER*

channel: the channel with which this compound filter will be associated, providing access to a channel and validation against the associated channel's configuration

returns: *PC_INVALID_FILTER*, if idKind is not *PC_AND_FILTER* or *PC_OR_FILTER* (in which case *pSCompoundFilter* is set to *PC_NULL*); otherwise *PC_SUCCESS*

PC_evn_clone_filter

```
PC_Status PC_EXPORT PC_evn_clone_filter(
    PC_evn_SFilter* pSAttrFilterNew,
    PC_evn_SFilter SAttrFilterOld)
```

Creates a clone of a filter.

pSAttrFilterNew: a pointer to a variable to hold the newly-allocated clone

SAttrFilterOld: the filter to be cloned

returns: *PC_SUCCESS*

PC_evn_destroy_filter

```
PC_Status PC_EXPORT PC_evn_destroy_filter(
    PC_evn_SFilter SAttrFilter)
```

Releases the resources associated with a filter.

SAttrFilter: the filter to be destroyed; the argument passed for *SAttrFilter* is no longer valid upon return from the function

returns: *PC_SUCCESS*

PC_evn_get_filter_string

```
PC_Status PC_EXPORT PC_evn_get_filter_string(
    PC_evn_SFilter SAttrFilter,
    PC_CHAR** pszAttrFilter)
```

Returns the string representation of a simple filter.

SAttrFilter: the filter whose string representation is to be returned

pszAttrFilter: a pointer to a variable to hold the newly-allocated string expression for this object; should be destroyed with *PC_evn_destroy_filter_string*

returns: *PC_SUCCESS*

PC_evn_destroy_filter_string

```
PC_Status PC_EXPORT PC_evn_destroy_filter_string(
    PC_CHAR* pszAttrFilter)
```

Destroys the buffer allocated in a previous call to *PC_evn_get_filter_string*.

szAttrFilter: the string buffer to be destroyed

returns: *PC_SUCCESS*

PC_evn_get_subfilters

```
PC_Status PC_EXPORT PC_evn_get_subfilters(
    PC_evn_SFilter SAttrFilter,
    PC_evn_SFilter** prgFilters,
    PC_UINT* punSize)
```

Returns the constituent subfilters of a compound filter.

SAttrFilter: the filter whose constituents are to be returned

prgFilters: a pointer to a variable to hold the newly-allocated array of constituent filters; should be destroyed with *PC_evn_destroy_subfilters*

punSize: a pointer to a variable to hold the number of elements in *prgFilters*

returns: *PC_INVALID_FILTER*, if *SAttrFilter* is not a compound filter (in which case *prgFilters* is set to *PC_NULL* and *punSize* is set to 0); otherwise *PC_SUCCESS*

PC_evn_destroy_subfilters

```
PC_Status PC_EXPORT PC_evn_destroy_subfilters(
```

```
    PC_evn_SFilter* rgFilters)
```

Destroys the buffer allocated in a previous call to *PC_evn_get_subfilters*.

rgFilters: the buffer to be destroyed

returns: *PC_SUCCESS*

PC_evn_match_filter

```
PC_BOOL PC_EXPORT PC_evn_match_filter(
    PC_evn_SFilter SAttrFilter,
    PC_evn_Notification notification)
```

Verifies whether a notification matches a filter.

SAttrFilter: the filter to be used for the match

notification: the notification to be matched

returns: *PC_TRUE*, if the notification matches the filter; otherwise, *PC_FALSE*

PC_evn_add_filter

```
PC_Status PC_EXPORT PC_evn_add_filter(
    PC_evn_SFilter SAttrFilter,
    PC_evn_SFilter SFilterToAdd)
```

Adds a subfilter to a compound filter.

SAttrFilter: the compound filter to which the new subfilter is to be added

SFilterToAdd: the subfilter to add

returns: *PC_INVALID_FILTER*, if *SAttrFilter* names an existing filter.

PC_evn_remove_filter

```
PC_Status PC_EXPORT PC_evn_remove_filter(
    PC_evn_SFilter SAttrFilter,
    PC_evn_SFilter SFilterToRemove)
```

Removes a subfilter from a compound filter.

SAttrFilter: the compound filter from which the subfilter is to be removed

SFilterToAdd: the subfilter to remove.

PC_evn_reset_filter

```
PC_Status PC_EXPORT PC_evn_reset_filter(
    PC_evn_SFilter SAttrFilter)
```

Removes all existing subfilters from a compound filter.

SAttrFilter: the compound filter to be reset

Notification

— the data for an event to be published

A Notification stores information about a single event that is to be published over a channel.

The object stores a subject for the notification as well as values for attributes that provide additional information about the event.

Type

```
typedef void* PC_evn_Notification
```

Functions

PC_evn_create_notification

```
PC_Status PC_EXPORT PC_evn_create_notification(
    PC_evn_Notification* pNotification,
    PC_evn_Proxy proxy,
    PC_CHAR* szSubject,
    PC_CHAR* szAttrVals)
```

Creates a notification object from a subject string and an optional list of attribute assignments, and associates the notification with a proxy.

pNotification: a pointer to a variable to hold the newly-allocated notification

proxy: the proxy with which this object will be associated, providing access to a channel and validation against the channel's subject tree and notification type

szSubject: a subject for the notification

szAttrVals: *PC_NULL*, or a comma-separated sequence of attribute assignments of the form *name=(type)value*

Note: *szAttrVals* must be set to *PC_NULL*.

returns: *PC_INVALID_SUBJECT*, if the specified subject is invalid for the underlying channel (in which case *pNotification* is set to *PC_NULL*); *PC_INVALID_NOTIFICATION*, if there is a problem setting up the notification message or if attribute assignments (the value of *szAttrVals*) have been supplied; otherwise *PC_SUCCESS*.

PC_evn_copy_notification

```
PC_Status PC_EXPORT PC_evn_copy_notification(
    PC_evn_Notification* pNotificationNew,
    PC_evn_Notification notificationOld)
```

Creates a copy of a notification.

pNotificationNew: a pointer to a variable to hold the newly-allocated copy

notificationOld: the notification to be copied

returns: *PC_INVALID_NOTIFICATION*, if the attempt to copy was unsuccessful (in which case *pNotificationNew* is set to *PC_NULL*); otherwise *PC_SUCCESS*

PC_evn_destroy_notification

```
PC_Status PC_EXPORT PC_evn_destroy_notification(
    PC_evn_Notification notification)
```

Releases the resources associated with a notification.

notification: the notification to be destroyed; the argument passed for notification is no longer valid upon return from the function

returns: *PC_SUCCESS*

PC_evn_get_subject

```
PC_Status PC_EXPORT PC_evn_get_subject(
    PC_evn_Notification notification,
    PC_CHAR** pszSubject)
```

Returns the subject of a notification.

notification: the notification whose subject is to be returned

pszSubject: a pointer to a variable to hold the newly-allocated subject string; should be destroyed with *PC_evn_destroy_subject*

returns: *PC_SUCCESS*

PC_evn_destroy_subject

```
PC_Status PC_EXPORT PC_evn_destroy_subject(
    PC_CHAR* szSubject)
```

Destroys the buffer allocated in a previous call to *PC_evn_get_subject*.

szSubject: the subject buffer to be destroyed

returns: *PC_SUCCESS*

PC_evn_get_subject_mapping

```
PC_Status PC_evn_get_subject_mapping(
    PC_evn_Notification notification,
    PC_UINT16** prgunSubjectFields,
    PC_UINT* pcSubjectFields)
```

Returns the mapping of a notification's subject to its array of subject field

notification: the notification whose subject mapping is to be returned

prgunSubjectFields: a pointer to a variable to hold the array of subject field mappings; should be destroyed with *PC_evn_destroy_subject_mapping*

pcSubjectFields: a pointer to a variable to hold the number of fields in the returned array

returns: *PC_SUCCESS*

PC_evn_destroy_subject_mapping

```
PC_Status PC_EXPORT PC_evn_destroy_subject_mapping(
    PC_UINT16* rgunSubjectFields)
```

Destroys the buffer allocated in a previous call to PC_evn_get_subject_mapping.

rgunSubjectFields: the buffer to be destroyed

returns: *PC_SUCCESS*

PC_evn_set_predefined_attr_by_pos

```
PC_Status PC_EXPORT
```

```
PC_evn_set_predefined_attr_by_pos(
PC_evn_Notification notification,
PC_UINT unPos,void* pVal,
PC_TypeCode idTypeCode,
PC_UINT unSize)
```

Sets the value of a predefined attribute of a notification, identified by its position.

Sets the value of a predefined attribute of a notification, identified by its position.

notification: the notification whose attribute is to be set

unPos: the position of the predefined attribute whose value is to be set

pVal: a pointer to the value to be assigned to the attribute (which itself may be a pointer in the case of an array or string value); the caller must ensure that the value is of the correct type

idTypeCode: the typecode of *pVal*

unSize: for array-valued attributes, the size of the array

returns: *PC_INVALID_NOTIFICATION*, if there is no predefined attribute in the specified position, or if *idTypeCode* is not the same as the type of the attribute as determined by the notification type of the channel; otherwise *PC_SUCCESS*

PC_evn_set_predefined_attr_by_name

```
PC_Status PC_EXPORT
    PC_evn_set_predefined_attr_by_name(
    PC_evn_Notification notification,
    PC_CHAR* szName,void* pVal,
    PC_TypeCode idTypeCode,
    PC_UINT unSize)
```

Sets the value of a predefined attribute of a notification, identified by its name.

notification: the notification whose attribute is to be set

szName: the name of the predefined attribute whose value is to be set

pVal: a pointer to the value to be assigned to the attribute (which itself may be a pointer in the case of an array or string value); the caller must ensure that the value is of the correct type

idTypeCode: the typecode of *pVal*

unSize: for array-valued attributes, the size of the array

returns: *PC_INVALID_NOTIFICATION*, if *szName* is not the name of a predefined attribute, or if *idTypeCode* is not the same as the type of the attribute as determined by the notification type of the channel; otherwise *PC_SUCCESS*

PC_evn_get_predefined_attr_by_pos

```
PC_Status PC_EXPORT
    PC_evn_get_predefined_attr_by_pos(
    PC_evn_Notification notification,
    PC_UINT unPos,void** ppVal,
    PC_TypeCode idTypeCode,
    PC_UINT* punSize)
```

Returns the value of a predefined attribute of a notification, identified by its position.

notification: the notification whose attribute is to be returned

unPos: the position of the predefined attribute whose value is to be returned

ppVal: a pointer to a variable to hold an untyped pointer to the value of the specified attribute, or for array-valued attributes to hold an untyped copy of the array pointer itself

idTypeCode: the typecode of *pVal*

punSize: for array-valued attributes, a pointer to a variable to hold the size of the array

returns: *PC_INVALID_NOTIFICATION*, if there is no predefined attribute in the specified position, or if *idTypeCode* is not the same as the type of the attribute as determined by the notification type of the channel; otherwise *PC_SUCCESS*

PC_evn_get_predefined_attr_by_name

```
PC_Status PC_EXPORT
    PC_evn_get_predefined_attr_by_name(
    PC_evn_Notification notification,
    PC_CHAR* szName,void** ppVal,
    PC_TypeCode idTypeCode,
    PC_UINT* punSize)
```

Returns the value of a predefined attribute of a notification, identified by its name.

notification: the notification whose attribute is to be returned

szName: the name of the predefined attribute whose value is to be returned

ppVal: a pointer to a variable to hold an untyped pointer to the value of the specified attribute, or for array-valued attributes to hold an untyped copy of the array pointer itself *idTypeCode* the typecode of *pVal*

punSize: for array-valued attributes, a pointer to a variable to hold the size of the array

returns: *PC_INVALID_NOTIFICATION*, if *szName* is not the name of a predefined attribute, or if *idTypeCode* is not the same as the type of the attribute as determined by the notification type of the underlying channel; otherwise *PC_SUCCESS*

PC_evn_add_discretionary_attr

```
PC_Status PC_EXPORT PC_evn_add_discretionary_attr(
    PC_evn_Notification notification,
    PC_CHAR* szName,void* pVal,
    PC_TypeCode idTypeCode,
    PC_UINT unSize)
```

Sets the value of a discretionary attribute of a notification.

notification: the notification whose attribute is to be set

szName: the name of the discretionary attribute whose value is to be set

pVal: a pointer to the value to be assigned to the attribute (which itself may be a pointer in the case of an array or string value); the caller must ensure that the value is of the correct type

idTypeCode: the typecode of *pVal*

unSize: for array-valued attributes, the size of the array

returns: *PC_INVALID_NOTIFICATION*, if szName is the name of a predefined attribute; otherwise *PC_SUCCESS*

PC_evn_get_discretionary_attr_names

```
PC_Status PC_EXPORT
    PC_evn_get_discretionary_attr_names(
    PC_evn_Notification notification,
    PC_CHAR*** prgszNames,
    PC_UINT* punSize)
```

Returns the names of all discretionary attributes of this object.

notification: the notification whose discretionary attribute names are to be returned

prgszNames: a pointer to a variable to hold the newly-allocated array of names; should be destroyed with *PC_evn_destroy_discretionary_attr_names*

punSize: a pointer to a variable to hold the number of elements in *prgszNames*

returns: *PC_INVALID_NOTIFICATION*, if there is a problem retrieving the discretionary attributes; otherwise *PC_SUCCESS*

PC_evn_is_discretionary_attr

```
PC_BOOL PC_EXPORT PC_evn_is_discretionary_attr(
    PC_evn_Notification notification,
    PC_CHAR* szName)
```

Returns an indication of whether or not this object has a discretionary attribute of a particular name.

notification: the notification whose discretionary attributes are to be checked

szName: the discretionary attribute name to check

returns: *true*, if *szName* is the name of a discretionary attribute of notification; otherwise *false*

PC_evn_destroy_discretionary_attr_names

```
PC_Status PC_EXPORT
    PC_evn_destroy_discretionary_attr_names(
    PC_CHAR** rgszNames,
    PC_UINT unSize)
```

Destroys the buffer allocated in a previous call to *PC_evn_get_discretionary_attr_names*.

rgszNames: the buffer to be destroyed

unSize: the number of elements in rgszNames

returns: *PC_SUCCESS*

PC_evn_get_discretionary_attr

```
PC_Status PC_EXPORT PC_evn_get_discretionary_attr(
    PC_evn_Notification notification,
    PC_CHAR* szName,void** ppVal,
    PC_TypeCode* pidTypeCode,
    PC_UINT* punSize)
```

Returns the value of a discretionary attribute of a notification.

notification: the notification whose attribute is to be returned

szName: the name of the discretionary attribute whose value is to be returned

ppVal: a pointer to a variable to hold an untyped pointer to the value of the specified attribute, or for array-valued attributes to hold an untyped copy of the array pointer itself

pidTypeCode: a pointer to a variable to hold the typecode of *ppVal*

punSize: for array-valued attributes, a pointer to a variable to hold the size of the array

returns: *PC_INVALID_NOTIFICATION*, if *szName* is the name of a discretionary attribute; otherwise *PC_SUCCESS*

Proxy

— connectivity to a channel

A proxy provides connectivity to a channel and validation of notifications and subscriptions against the channel's Notification type.

Proxies also delivers matching notifications to subscriber's via Subscriber callbacks.

Type

```
typedef void* PC_evn_Proxy
```

Functions

PC_evn_create_proxy

```
PC_Status PC_EXPORT PC_evn_create_proxy(
    PC_evn_Proxy* pProxy,
    PC_UINT idChannel,
    PC_UINT idPublisher)
```

Creates a proxy for a specified channel and publisher.

pProxy: a pointer to a variable to hold the newly-allocated proxy

idChannel: the ID of the channel to be accessed

idPublisher: the ID of the application using this object

returns: *PC_INVALID_CHANNEL*, if the specified channel ID is invalid, or if the specified publisher ID is not allowed to access the channel (in which case pProxy is set to *PC_NULL*); otherwise *PC_SUCCESS*

PC_evn_recover_proxy

```
PC_Status PC_EXPORT PC_evn_recover_proxy(
    PC_evn_Proxy* pProxy,
    PC_UINT idChannel,
    PC_UINT idPublisher)
```

Creates a proxy for a specified channel and publisher in recovery mode.

pProxy: a pointer to a variable to hold the newly-allocated proxy

idChannel: the ID of the channel to be accessed

idPublisher: the ID of the application using this object

returns: *PC_INVALID_CHANNEL*, if the specified channel ID is invalid, or if the specified publisher ID is not allowed to access the channel (in which case pProxy is set to *PC_NULL*); otherwise *PC_SUCCESS*

PC_evn_destroy_proxy

```
PC_Status PC_EXPORT PC_evn_destroy_proxy(
    PC_evn_Proxy proxy)
```

Releases the resources associated with a proxy.

proxy: the proxy to be destroyed; the argument passed for proxy is no longer valid upon return from the function

returns: *PC_SUCCESS*

PC_evn_get_channel_info

```
PC_evn_Channel_Info PC_EXPORT
    PC_evn_get_channel_info(
    PC_evn_Proxy proxy,
    PC_BOOL bIsControl)
```

Obtains the information about the channel accessed by a proxy.

proxy: the proxy whose channel information is to be retrieved

bIsControl: an indication of whether the control channel information (*PC_TRUE*) or data channel information (*PC_FALSE*) is desired

returns: the underlying channel information

PC_evn_proxy_save

```
PC_Status PC_EXPORT PC_evn_proxy_save (
    PC_evn_Proxy proxy,
    PC_STRING szFileName)
```

Saves the current proxy state in the file.

proxy: the proxy whose state is being saved

szFileName: the name of the file to be serialized

PC_evn_proxy_load

```
PC_Status PC_EXPORT PC_evn_proxy_load (
    PC_evn_Proxy* pproxy,
    PC_STRING szFileName)
```

Restores a saved proxy from a file

pproxy: a pointer to the proxy variable being loaded

szFileName: the name of the file to be serialized

PC_evn_register_callback

```
int PC_EXPORT PC_evn_register_callback (
    const PC_STRING,
    PC_evn_pfnCallback)
```

Registers the callback such that it would be available for serialization.

PC_evn_set_client_type

```
PC_evn_set_client_type (
    PC_UINT idChannel,
    PC_BOOL bIsAllowedToPublish,
    PC_BOOL bIsAllowedToSubscribe)
```

Specifies whether the client is allowed to subscribe or publish on a specific channel.

idChannel: the ID of the channel to be accessed

bIsAllowedToPublish: when set to *true* it enables the application to publish on the given channel

bIsAllowedToSubscribe: when set to *true* it enables the application to subscribe on the given channel

Subscriber

— callback functions

In order to receive Notification, subscriber applications implement a callback function, which is invoked by a Proxy to match notifications to the callback's criteria.

Type

```
typedef void PC_evn_pfnCallback(
    PC_evn_Notification notification,
    void* pClosure)
```

notification: the notification delivered to the callback

pClosure: a pointer to the closure object associated with the subscription that the notification matched

Functions

PC_evn_publish

```
PC_Status PC_EXPORT PC_evn_publish (
    PC_evn_Proxy proxy,
    PC_evn_Notification notification)
```

Delivers a notification over a channel via a proxy.

proxy: the proxy to be used to publish the notification

notification: the notification to be published

returns: *PC_INVALID_NOTIFICATION*, if the notification is incomplete or otherwise invalid for publication over the channel, or if a problem was encountered in attempting to publish it over the channel; otherwise *PC_SUCCESS*

PC_evn_subscribe

```
PC_Status PC_EXPORT PC_evn_subscribe(
    PC_evn_Proxy proxy,
    PC_CHAR* szSubjectFilter,
    PC_evn_SFilter SAttrFilter,
    PC_evn_pfnCallback callback,void* pClosure,
    PC_evn_pfnCallback ctrlCallback,
    PC_evn_Subscription_Handle* pShSubscription)
```

Subscribes via a proxy by providing a subject filter string and attribute filter.

proxy: the proxy to be used to register the subscription

szSubjectFilter: a filter to be applied to the subject of notifications sent over the channel

SAttrFilter: an attribute filter to be applied to the attributes of notifications sent over the channel

callback: the function to be invoked to deliver matching notifications for the subscription

pClosure: a pointer that is passed to the callback to allow the receiving application to identify which subscription was matched by the delivered notification, in case the same callback is associated with multiple subscriptions

ctrlCallback: *PC_NULL*, or a function to be invoked to deliver control notifications associated wtih this subscription

pShSubscription: a pointer to variable to hold a newly-allocated handle that can be used to identify the registered subscription in other operations

returns: *PC_INVALID_FILTER*, if a problem was encountered in attempting to register the subscription with the channel (in which case *pShSubscription* is set to *PC_NULL*); *PC_INVALID_SUBJECT*, if the subject filter is syntactically incorrect or names fields that are invalid for the underlying channel (in which case *pShSubscription* is set to *PC_NULL*; otherwise *PC_SUCCESS*

PC_evn_subscribe_with_exp

```
PC_Status PC_EXPORT PC_evn_subscribe_with_exp(
    PC_evn_Proxy proxy,PC_CHAR* szSubjectFilter,
    PC_CHAR* szAttrFilter,
    PC_evn_pfnCallback callback,void* pClosure,
    PC_evn_pfnCallback ctrlCallback,
    PC_evn_Subscription_Handle* pShSubscription)
```

Subscribes via a proxy by providing a subject filter string and attribute filter string.

proxy: the proxy to be used to register the subscription

szSubjectFilter: a filter to be applied to the subject of notifications sent over the channel

szAttrFilter: *PC_NULL*, or an attribute filter to be applied to the attributes of notifications sent over the channel; a null value sets the attribute filter to true

callback: the function to be invoked to deliver matching notifications for this subscription

pClosure: a pointer that is passed to the callback to allow the receiving application to identify which subscription was matched by the delivered notification, in case the same callback is associated with multiple subscriptions

ctrlCallback: PC_NULL, or a function to be invoked to deliver control notifications associated wtih this subscription

pShSubscription: a pointer to variable to hold a newly-allocated handle that can be used to identify the registered subscription in other operations

returns: *PC_INVALID_FILTER*, if a problem was encountered in attempting to register the subscription with the channel (in which case *pShSubscription* is set to *PC_NULL*); PC_INVALID_SUBJECT, if the subject filter is syntactically incorrect or names fields that are invalid for the underlying channel (in which case *pShSubscription* is set to *PC_NULL*; otherwise *PC_SUCCESS*

PC_evn_control_subscribe

```
PC_Status PC_EXPORT PC_evn_control_subscribe(
    PC_evn_Proxy proxy,
    PC_CHAR* szSubjectFilter,
    PC_evn_SFilter SAttrFilter,
    PC_evn_pfnCallback callback,void* pClosure,
    PC_evn_Subscription_Handle* pShSubscription)
```

Subscribes to a control subject via a proxy by providing a subject filter string and attribute filter.

proxy: the proxy to be used to register the subscription

szSubjectFilter: a filter to be applied to the subject of notifications sent over the channel

SAttrFilter: an attribute filter to be applied to the attributes of notifications sent over the channel

callback: the function to be invoked to deliver matching notifications for the subscription

pClosure: a pointer that is passed to the callback to allow the receiving application to identify which subscription was matched by the delivered notification, in case the same callback is associated with multiple subscriptions

pShSubscription: a pointer to variable to hold a newly-allocated handle that can be used to identify the registered subscription in other operations

returns: *PC_INVALID_FILTER*, if a problem was encountered in attempting to register the subscription with the channel (in which case *pShSubscription* is set to *PC_NULL*); *PC_INVALID_SUBJECT*, if the subject filter is syntactically incorrect or names fields that are invalid for the underlying channel (in which case *pShSubscription* is set to *PC_NULL*; otherwise *PC_SUCCESS*

PC_evn_control_subscribe_with_exp

```
PC_Status PC_EXPORT
    PC_evn_control_subscribe_with_exp(
    PC_evn_Proxy proxy,
    PC_CHAR* szSubjectFilter,
    PC_CHAR* szAttrFilter,
    PC_evn_pfnCallback callback,void* pClosure,
    PC_evn_Subscription_Handle* pShSubscription)
```

Subscribes to a control subject via a proxy by providing a subject filter string and attribute filter string.

proxy: the proxy to be used to register the subscription

szSubjectFilter: a filter to be applied to the subject of notifications sent over the channel

szAttrFilter: PC_NULL, or an attribute filter to be applied to the attributes of notifications sent over the channel; a null value sets the attribute filter to true

callback: the function to be invoked to deliver matching notifications for the subscription

pClosure: a pointer that is passed to the callback to allow the receiving application to identify which subscription was matched by the delivered notification, in case the same callback is associated with multiple subscriptions

pShSubscription: a pointer to variable to hold a newly-allocated handle that can be used to identify the registered subscription in other operations

returns: *PC_INVALID_FILTER*, if a problem was encountered in attempting to register the subscription with the channel (in which case *pShSubscription* is set to *PC_NULL*); *PC_INVALID_SUBJECT*, if the subject filter is syntactically incorrect or names fields that are invalid for the underlying channel (in which case *pShSubscription* is set to PC_NULL; otherwise *PC_SUCCESS*

Subscription handlers

— a subscription interface

Subscription handler functions identify a subscription registered with a Proxy. Proxies can invoke subscription handlers to perform various operations on the subscription.

Type

```
typedef struct PC_evn_Subscription_Handle
```

Functions

PC_evn_pause

Halts delivery of matching notifications to the callback associated with a subscription.

Note: The operation is idempotent.

```
PC_Status PC_EXPORT PC_evn_pause(
    PC_evn_Proxy proxy,
    PC_evn_Subscription_Handle ShSubscription)
```

proxy: the proxy with which the subscription was registered

ShSubscription: a handle for the subscription to be paused

returns: *PC_INVALID_SUBSCRIPTION_HANDLE*, if the subscription handle does not identify a subscription currently registered with the proxy; otherwise *PC_SUCCESS*

PC_evn_suspend

```
PC_Status PC_EXPORT PC_evn_suspend(
    PC_evn_Proxy proxy,
    PC_evn_Subscription_Handle ShSubscription)
```

Halts matching of notifications against a subscription.

Note: The operation is idempotent.

proxy: the proxy with which the subscription was registered

ShSubscription: a handle for the subscription to be suspended

returns: *PC_INVALID_SUBSCRIPTION_HANDLE*, if the subscription handle does not identify a subscription currently registered with the proxy; otherwise *PC_SUCCESS*

PC_evn_resume

```
PC_Status PC_EXPORT PC_evn_resume(
    PC_evn_Proxy proxy,
    PC_evn_Subscription_Handle ShSubscription)
```

Resumes matching and delivery of notifications against a previously paused or suspended subscription, including notifications received while the subscription was paused.

proxy: the proxy with which the subscription was registered

ShSubscription: a handle for the subscription to be resumed

returns: *PC_INVALID_SUBSCRIPTION_HANDLE*, if the subscription handle does not identify a subscription currently registered with the proxy; otherwise *PC_SUCCESS*

PC_evn_unsubscribe

```
PC_Status PC_EXPORT PC_evn_unsubscribe(
    PC_evn_Proxy proxy,
    PC_evn_Subscription_Handle ShSubscription)
```

Unregisters a subscription from a channel.

proxy: the proxy with which the subscription was registered

ShSubscription: a handle for the subscription to be unregistered; the argument passed for *hSubscription* is no longer valid upon return from the function

returns: *PC_INVALID_SUBSCRIPTION_HANDLE*, if the subscription handle does not identify a subscription currently registered with the proxy; otherwise *PC_SUCCESS*

PC_evn_control_unsubscribe

```
PC_Status PC_EXPORT PC_evn_control_unsubscribe(
    PC_evn_Proxy proxy,
    PC_evn_Subscription_Handle ShSubscription)
```

Unregisters a control subscription from a channel.

proxy: the proxy with which the subscription was registered

ShSubscription: a handle for the subscription to be unregistered; the argument passed for *hSubscription* is no longer valid upon return from the function

returns: *PC_INVALID_SUBSCRIPTION_HANDLE*, if the subscription handle does not identify a subscription currently registered with the proxy; otherwise *PC_SUCCESS*

PC_evn_get_subscriptions_count

```
PC_Status PC_EXPORT PC_evn_get_subscriptions_count (
    PC_evn_Proxy proxy,
    PC_UINT* pcSubscription)
```

Retrieves the number of the subscriptions associated with a proxy.

proxy: the proxy being queried

pcSubscription: the number of subscriptions (result)

returns: *PC_SUCCESS* is the operation is completed successfuly and *PC_ERROR* otherwise

PC_evn_get_subscriptions

```
PC_Status PC_EXPORT PC_evn_get_subscriptions
    (PC_evn_Proxy proxy, PC_evn_Subscription_Handle
    rgShSubscription[], PC_UINT
    unrgShSubscriptionSize,
    PC_UINT* pcSubscription)
```

Retrieves the subscriptions associated with a proxy.

proxy: the proxy being queried

rgShSubscription: the buffer where the subscriptions will be stored

unrgShSubscriptionSize: the size of the buffer used provided for retrieving the subscriptions

pcSubscription: the number of subscriptions actually read

returns: *PC_SUCCESS* is the operation is completed successfuly and *PC_ERROR* otherwise

PC_evn_get_attr_filter_string

```
PC_Status PC_EXPORT
    PC_evn_get_attr_filter_string (
    PC_evn_Subscription_Handle ShSubscription,
    PC_CHAR** pszAttrFilter)
```

Retrives the string representation of the attribute filter associated with the subscription.

ShSubscription: The handle of the subscription being queried

pszAttrFilter: Upon succesful completion contains a pointer to the result

returns: *PC_SUCCESS* is the operation is completed successfuly and *PC_ERROR* otherwise

PC_evn_get_subject_filter

```
PC_Status PC_EXPORT PC_evn_get_subject_filter (
    PC_evn_Subscription_Handle ShSubscription,
    PC_CHAR** pszSubjectFilter)
```

Retrives the string representation of the subject filter associated with the subscription.

ShSubscription: The handle of the subscription being queried

pszSubjectFilter: Upon succesful completion contains a pointer to the result

returns: *PC_SUCCESS*, if the operation is completed successfuly; otherwise, *PC_ERROR*

PC_evn_get_closure

```
PC_Status PC_EXPORT PC_evn_get_closure (
    PC_evn_Subscription_Handle ShSubscription,
    void** ppClosure)
```

Retrieves a pointer to the closure associated with the current subscription.

ShSubscription: The handle of the subscription being queried

ppClosure: upon succesful completion contains a pointer to the result

returns: *PC_SUCCESS* is the operation is completed successfuly and *PC_ERROR* otherwise

PC_evn_Channel_Info

— an abstraction for the channel configuration

ChannelInfo needs to be associated with various objects such as Filter or Notification.

ChannelInfo provides an abstraction for the channel configuration, consisting of the subject and attributes of a channel, which may be retrieved by a proxy. No functions are exposed.