zenoh-java

Release 0.4.2-M1

Sep 24, 2020

Contents

The *zenoh-java* library provides a Java zenoh client API for zenoh.

An introduction to zenoh and its concepts is available on zenoh.io.

> **Warning:** zenoh has been subjet to a complete rewrite with major protocol updates between versions 0.4.2 and 0.5.0. The Java API does not yet integrate those changes and is only compatible with version 0.4.2 of the zenoh daemon and the underlying zenoh-c stack.

Note that this library also provides a low-level API (zenoh-net) that gives access to the zenoh protocol primitives and allow some advanced use cases where a fine tuning of the protocol is required.

CHAPTER 1

Zenoh API Reference

1.1 org.eclipse.zenoh

1.1.1 Admin

public class **`Admin`**
The zenoh administration class.

Constructors

Admin

protected **`Admin`** (*Workspace* *w*, String *zid*)

Methods

addBackend

public void **`addBackend`** (String *beid*, Properties *properties*)
Add a backend in the connected zenoh router (i.e. the one you are directly connected to).

Parameters

- **`beid`** – the backend identifier.
- **`properties`** – the properties for backend initialization.

Throws

- ***`ZException`*** – if an error occurs.

addBackend

public void **addBackend**(String *beid*, Properties *properties*, String *zid*)
Add a backend in the specified zenoh router, not necessarily the one you are connected to.

Parameters

- **beid** – the backend identifier.
- **properties** – the properties for backend initialization.
- **zid** – the zenoh router identifier.

Throws

- ***ZException*** – if an error occurs.

addStorage

public void **addStorage**(String *stid*, Properties *properties*)
Add a storage in the connected zenoh router, using an automatically chosen backend.

Parameters

- **stid** – the storage identifier
- **properties** – the properties for storage initialization.

Throws

- ***ZException*** – if an error occurs.

addStorage

public void **addStorage**(String *stid*, Properties *properties*, String *zid*)
Add a storage in the specified zenoh router, using an automatically chosen backend.

Parameters

- **stid** – the storage identifier
- **properties** – the properties for storage initialization.
- **zid** – the zenoh router identifier.

Throws

- ***ZException*** – if an error occurs.

addStorageOnBackend

public void **addStorageOnBackend**(String *stid*, Properties *properties*, String *backend*)
Add a storage in the connected zenoh router, using the specified backend.

Parameters

- **stid** – the storage identifier.
- **properties** – the properties for storage initialization.
- **backend** – the identifier of the backend to use for the storage.

Throws

- ***ZException*** – if an error occurs.

addStorageOnBackend

public void **addStorageOnBackend**(String *stid*, Properties *properties*, String *backend*, String *zid*)
Add a storage in the specified zenoh router, using the specified backend.

Parameters

- **stid** – the storage identifier.
- **properties** – the properties for storage initialization.
- **backend** – the identifier of the backend to use for the storage.
- **zid** – the zenoh router identifier.

Throws

- ***ZException*** – if an error occurs.

getBackend

public Properties **getBackend**(String *beid*)
Get a backend's properties from the connected zenoh router (i.e. the one you are directly connected to).

Parameters

- **beid** – the backend identifier

Throws

- ***ZException*** – if an error occurs.

Returns the backend properties

getBackend

public Properties **getBackend**(String *beid*, String *zid*)
Get a backend's properties from the specified zenoh router, not necessarily the one you are connected to.

Parameters

- **beid** – the backend identifier
- **zid** – the zenoh router identifier.

Throws

- ***ZException*** – if an error occurs.

Returns the backend properties

getBackends

public Map<String, Properties> **getBackends**()
Get all the backends from the connected zenoh router (i.e. the one you are directly connected to).

Throws

- ***ZException*** – if an error occurs.

Returns a map of the backends properties, indexed by the backends identifiers.

getBackends

public Map<String, Properties> **getBackends**(String *zid*)
Get all the backends from the specified zenoh router, not necessarily the one you are connected to.

Parameters

- **zid** – the zenoh router identifier.

Throws

- ***ZException*** – if an error occurs.

Returns a map of the backends properties, indexed by the backends identifiers.

getStorage

public Properties **getStorage**(String *stid*)
Get a storage's properties from the connected zenoh router.

Parameters

- **stid** – the storage identifier

Throws

- ***ZException*** – if an error occurs.

Returns the storage properties

getStorage

public Properties **getStorage**(String *stid*, String *zid*)
Get a storage's properties from the specified zenoh router.

Parameters

- **stid** – the storage identifier
- **zid** – the zenoh router identifier.

Throws

- ***ZException*** – if an error occurs.

Returns the storage properties

getStorages

public Map<String, Properties> **getStorages**()
Get all the storages from the connected zenoh router.

Throws

- ***ZException*** – if an error occurs.

Returns a map of the storages properties, indexed by the storages identifiers.

getStorages

public Map<String, Properties> **getStorages**(String *zid*)
Get all the storages from the specified zenoh router.

Parameters

- **zid** – the zenoh router identifier.

Throws

- ***ZException*** – if an error occurs.

Returns a map of the storages properties, indexed by the storages identifiers.

getStoragesFromBackend

public Map<String, Properties> **getStoragesFromBackend**(String *backend*)
Get all the storages from the specified backend within the connected zenoh router.

Parameters

- **backend** – the backend identifier.

Throws

- ***ZException*** – if an error occurs.

Returns a map of the storages properties, indexed by the storages identifiers.

getStoragesFromBackend

public Map<String, Properties> **getStoragesFromBackend**(String *backend*, String *zid*)
Get all the storages from the specified backend within the specified zenoh router.

Parameters

- **backend** – the backend identifier.
- **zid** – the zenoh router identifier.

Throws

- ***ZException*** – if an error occurs.

Returns a map of the storages properties, indexed by the storages identifiers.

removeBackend

public void **removeBackend**(String *beid*)

Remove a backend from the connected zenoh router (i.e. the one you are directly connected to).

Parameters

- **beid** – the backend identifier.

Throws

- ***ZException*** – if an error occurs.

removeBackend

public void **removeBackend**(String *beid*, String *zid*)

Remove a backend from the specified zenoh router, not necessarily the one you are connected to.

Parameters

- **beid** – the backend identifier.
- **zid** – the zenoh router identifier.

Throws

- ***ZException*** – if an error occurs.

removeStorage

public void **removeStorage**(String *stid*)

Remove a storage from the connected zenoh router.

Parameters

- **stid** – the storage identifier.

Throws

- ***ZException*** – if an error occurs.

removeStorage

public void **removeStorage**(String *stid*, String *zid*)

Remove a storage from the specified zenoh router.

Parameters

- **stid** – the storage identifier.
- **zid** – the zenoh router identifier.

Throws

- ***ZException*** – if an error occurs.

1.1.2 Change

public class **Change**
The notification of a change for a resource in zenoh. See *Listener*.

Constructors

Change

protected **Change**(*Path path*, *Kind kind*, *Timestamp timestamp*, *Value value*)

Methods

getKind

public *Kind* **getKind**()
Returns the *Kind* of change.

Returns the kind of change.

getPath

public *Path* **getPath**()
Returns the *Path* of resource that changed.

Returns the resource path.

getTimestamp

public *Timestamp* **getTimestamp**()
Returns the *Timestamp* when change occured.

Returns the timestamp.

getValue

public *Value* **getValue**()
Depending of the change *Kind*, returns:

- if kind is *Kind.PUT*: the new value
- if kind is *Kind.UPDATE*: the delta value
- if kind is *Kind.REMOVE*: null

Returns the new value (complete or delta), or null.

1.1.3 Change.Kind

public enum **Kind**
The kind of Change: either *PUT*, *UPDATE* or *REMOVE*.

Enum Constants

PUT

public static final *Change*.*Kind* **PUT**

REMOVE

public static final *Change*.*Kind* **REMOVE**

UPDATE

public static final *Change*.*Kind* **UPDATE**

Methods

fromInt

protected static *Kind* **fromInt** (int *numVal*)

value

public int **value** ()
Returns the numeric value of the change kind (the same than in zenoh-c).

Returns the numeric value.

1.1.4 Data

public class **Data** implements Comparable<*Data*>
A zenoh data returned by a *Workspace.get(Selector)* query. The Data objects are comparable according to their *Timestamp*. Note that zenoh makes sure that each published path/value has a unique timestamp accross the system.

Constructors

Data

protected **Data** (*Path path*, *Value value*, *Timestamp timestamp*)

Methods

compareTo

public int **compareTo** (*Data o*)

equals

public boolean **equals**(Object *obj*)

getPath

public *Path* **getPath**()
Returns the *Path* of the data.

Returns the path of the data.

getTimestamp

public *Timestamp* **getTimestamp**()
Returns the *Timestamp* of the data.

Returns the timestamp of the data.

getValue

public *Value* **getValue**()
Returns the *Value* of the data.

Returns the value of the data.

hashCode

public int **hashCode**()

1.1.5 Encoding

public enum **Encoding**
A description of the *Value* format, allowing zenoh to know how to encode/decode the value to/from a bytes buffer.

Enum Constants

FLOAT

public static final *Encoding* **FLOAT**

INT

public static final *Encoding* **INT**

JSON

public static final *Encoding* **JSON**
The value is a JSON structure in an UTF-8 string.

PROPERTIES

public static final *Encoding* **PROPERTIES**
The value if a list of keys/values, encoded as an UTF-8 string. The keys/values are separated by ';' character, and each key is separated from its associated value (if any) with a '=' character.

RAW

public static final *Encoding* **RAW**
The value has a RAW encoding (i.e. it's a bytes buffer).

STRING

public static final *Encoding* **STRING**
The value is an UTF-8 string.

Methods

fromFlag

protected static *Encoding* **fromFlag**(short *flag*)

getDecoder

public *Value.Decoder* **getDecoder**()
Returns the *Value.Decoder* for this encoding.

Returns the decoder.

getFlag

public short **getFlag**()
Returns the numeric flag corresponding to the encoding, for usage with zenoh-net.

Returns the encoding flag.

1.1.6 Eval

public interface **Eval**
Interface to be implemented by evaluation functions (see *Workspace.registerEval(Path,Eval)*)

Methods

callback

public *Value* **callback** (*Path path*, Properties *props*)
The callback operation called each time the *Workspace.get(Selector)* operation is called for a Selector matching the Path this Eval is registered with. Zenoh will wrap the returned *Value* into a *Data* and add it to the result of the get() operation.

Parameters

- **path** – the Path with which the Eval has been registered with (in case the same Eval is registered with several Paths).
- **props** – the Properties specified in the Selector used in eval operation.

Returns a Value resulting of the evaluation.

1.1.7 FloatValue

public class **FloatValue** implements *Value*
A *Value* containing a 64-bits signed float (i.e. a Java double).

Fields

Decoder

public static final *Value.Decoder* **Decoder**
The *Value.Decoder* for *StringValue*s.

Constructors

FloatValue

public **FloatValue** (double *v*)
Creates an IntValue containing a int.

Parameters

- **v** – the float

Methods

encode

public static ByteBuffer **encode** (double *v*)

encode

public ByteBuffer **encode** ()

equals

public boolean **equals** (Object *obj*)

getEncoding

public *Encoding* **getEncoding** ()

getFloat

public double **getFloat** ()
Returns the string from this StringValue

Returns the string

hashCode

public int **hashCode** ()

toString

public String **toString** ()

1.1.8 IntValue

public class **IntValue** implements *Value*
A `Value` containing a 64-bits signed integer (i.e. a Java long).

Fields

Decoder

public static final *Value*.*Decoder* **Decoder**
The `Value.Decoder` for `StringValue`s.

Constructors

IntValue

public **IntValue** (long *v*)
Creates an IntValue containing a long.

Parameters

- **v** – the long

Methods

encode

public static ByteBuffer **encode**(long *v*)

encode

public ByteBuffer **encode**()

equals

public boolean **equals**(Object *obj*)

getEncoding

public *Encoding* **getEncoding**()

getInt

public long **getInt**()
Returns the string from this StringValue

Returns the string

hashCode

public int **hashCode**()

toString

public String **toString**()

1.1.9 Listener

public interface **Listener**
Interface to be implemented for subscriptions (see *Workspace.subscribe(Selector,Listener)*)

Methods

onChanges

public void **onChanges**(List<*Change*> *changes*)
The callback operation called for all changes on subscribed paths.

Parameters

- **changes** – the list of changes.

1.1.10 Path

public class **Path** implements Comparable<*Path*>

A zenoh Path is a set of strings separated by '/' , as in a filesystem path. A Path cannot contain any '*' character. Examples of paths: "/zenoh/examples/test" , "/com/adlink/building/fr/floor/1/office/2" ...

A path can be absolute (i.e. starting with a '/') or relative to a *Workspace*.

See also: *Workspace.put(Path,Value)*

Constructors

Path

public **Path** (String *p*)

Create a Path from a string such as "/zenoh/examples/test".

Parameters

- **p** – the string

Methods

addPrefix

public *Path* **addPrefix** (*Path prefix*)

Returns a new Path made of the concatenation of the specified prefix and this Path.

Parameters

- **prefix** – the prefix to add.

Returns a new Path made of the prefix plus this Path.

compareTo

public int **compareTo** (*Path p*)

equals

public boolean **equals** (Object *object*)

hashCode

public int **hashCode** ()

isRelative

static boolean **isRelative** (String *p*)

isRelative

public boolean **isRelative**()
Returns true if the Path is relative (i.e. it doesn't start with '/')

Returns true if the Path is relative.

length

public int **length**()
The Path length (i.e. the length of the Path as a string)

Returns the length

toString

public String **toString**()

1.1.11 PropertiesValue

public class **PropertiesValue** implements *Value*
A *Value* containing Properties.

Fields

Decoder

public static final *Value.Decoder* **Decoder**
The *Value.Decoder* for *PropertiesValue*s.

Constructors

PropertiesValue

public **PropertiesValue**(Properties *p*)
Creates a PropertiesValue containing some Properties.

Parameters

- **p** – the properties

Methods

encode

public ByteBuffer **encode**()

equals

public boolean **equals** (Object *obj*)

getEncoding

public *Encoding* **getEncoding** ()

getProperties

public Properties **getProperties** ()
Returns the properties from this PropertiesValue

Returns the properties

hashCode

public int **hashCode** ()

toString

public String **toString** ()

1.1.12 RawValue

public class **RawValue** implements *Value*
A `Value` containing a `ByteBuffer`.

Fields

Decoder

public static final *Value.Decoder* **Decoder**
The `Value.Decoder` for `RawValue`s.

Constructors

RawValue

public **RawValue** (ByteBuffer *buf*)
Creates a RawValue containing a `ByteBuffer`.

Parameters

- **buf** – the bytes buffer.

Methods

encode

public ByteBuffer **encode**()

equals

public boolean **equals**(Object *obj*)

getBuffer

public ByteBuffer **getBuffer**()
Returns the `ByteBuffer` from this RawValue

Returns the bytes buffer.

getEncoding

public *Encoding* **getEncoding**()

hashCode

public int **hashCode**()

toString

public String **toString**()

1.1.13 Selector

public final class **Selector** implements Comparable<*Selector*>
A zenoh Selector is a string which is the conjunction of an path expression identifying a set of keys and some optional parts allowing to refine the set of `Path`s and associated `Value`s. Structure of a selector:

```
/s1/s2/../sn?x>1&y<2&..&z=4(p1=v1;p2=v2;..;pn=vn)#a;x;y;..;z
|          | |            | |                   |  |        |
|-- expr --| |-- filter --| |--- properties ---|  |fragment|
```

where:

- **expr**: is a path expression. I.e. a string similar to a `Path` but with character '*' allowed. A single '*' matches any set of characters in a path, except '/'. While "**" matches any set of characters in a path, including '/'. A path expression can be absolute (i.e. starting with a '/') or relative to a `Workspace`.
- **filter**: a list of predicates separated by '&' allowing to perform filtering on the `Value` associated with the matching keys. Each predicate has the form "*field*''*operator*''*value*" where:
  - **field** is the name of a field in the value (is applicable and is existing. otherwise the predicate is false)
  - **operator** is one of a comparison operators: '' , <= , >= , = , !=

  - **value** is the the value to compare the field's value with

- **fragment**: a list of fields names allowing to return a sub-part of each value. This feature only applies to structured values using a "self-describing" encoding, such as JSON or XML. It allows to select only some fields within the structure. A new structure with only the selected fields will be used in place of the original value.

NOTE: the filters and fragments are not yet supported in current zenoh version.

See also: *Workspace.get(Selector)*, *Workspace.subscribe(Selector,Listener)*

Constructors

Selector

public **Selector** (String *s*)
Create a Selector from a string such as "/zenoh/examples/**?(name=Bob)".

Parameters

- **p** – the string

Methods

addPrefix

public *Selector* **addPrefix** (*Path prefix*)
Returns a new Selector made of the concatenation of the specified prefix and this Selector.

Parameters

- **prefix** – the prefix to add.

Returns a new Selector made of the prefix plus this Selector.

compareTo

public int **compareTo** (*Selector s*)

equals

public boolean **equals** (Object *object*)

getFilter

public String **getFilter** ()
Return the filter expression of this Selector. I.e. the substring after the '?' character and before the '(' character (or until end of string).

Returns the filter expression or an empty string if not present.

getFragment

public String **getFragment**()
Return the fragment expression of this Selector. I.e. the substring after the '#' character.

Returns the fragment expression or an empty string if not present.

getOptionalPart

protected String **getOptionalPart**()

getPath

public String **getPath**()
Return the path expression of this Selector. I.e. the substring before the '?' character.

Returns the path expression.

getProperties

public String **getProperties**()
Return the properties expression of this Selector. I.e. the substring enclosed between '(' and ')' after the '?' character.

Returns the properties expression or an empty string if not present.

hashCode

public int **hashCode**()

isRelative

public boolean **isRelative**()
Returns true if the Selector is relative (i.e. it doesn't start with '/')

Returns true if the Selector is relative.

toString

public String **toString**()

1.1.14 StringValue

public class **StringValue** implements *Value*
A *Value* containing an UTF-8 String.

Fields

Decoder

public static final *Value.Decoder* **Decoder**
The `Value.Decoder` for `StringValue`s.

Constructors

StringValue

public **StringValue** (String *s*)
Creates a StringValue containing a `String`.

Parameters

- **s** – the string

Methods

encode

public static ByteBuffer **encode** (String *s*)

encode

public ByteBuffer **encode** ()

equals

public boolean **equals** (Object *obj*)

getEncoding

public *Encoding* **getEncoding** ()

getString

public String **getString** ()
Returns the string from this StringValue

Returns the string

hashCode

public int **hashCode** ()

toString

public String **toString**()

1.1.15 SubscriptionId

public final class **SubscriptionId**

The identifier of a subscription.

See also: *Workspace.subscribe(Selector,Listener)*, *Workspace.unsubscribe(SubscriptionId)*

Constructors

SubscriptionId

protected **SubscriptionId**(*Subscriber sub*)

Methods

getZSubscriber

protected *Subscriber* **getZSubscriber**()

1.1.16 Value

public interface **Value**

Interface of a Value that, associated to a *Path*, can be published into zenoh via *Workspace.put(Path,Value)*, or retrieved via *Workspace.get(Selector)* or via a subscription (*Workspace.subscribe(Selector,Listener)*).

Methods

encode

public ByteBuffer **encode**()

Returns a new `ByteBuffer` containing the Value encoded as a sequence of bytes.

Returns the encoded Value.

getEncoding

public *Encoding* **getEncoding**()

Return the *Encoding* of this Value.

Returns the encoding.

1.1.17 Value.Decoder

public interface **Decoder**
Interface of a Value decoder, able to transform a `ByteBuffer` into a Value.

Methods

decode

public *Value* **decode** (ByteBuffer *buf*)
Decode a Value that is encoded in a `ByteBuffer`.

Parameters

- **buf** – the ByteBuffer containing the encoded Value.

Returns the Value.

getEncodingFlag

public short **getEncodingFlag** ()
Returns the flag of the *`Encoding`* that this Decoder supports. (@see Encoding#getFlag()).

Returns the encoding flag.

1.1.18 Workspace

public class **Workspace**
A Workspace to operate on Zenoh.

Constructors

Workspace

protected **Workspace** (*Path path*, *Session session*, ExecutorService *threadPool*)

Methods

get

public Collection<*Data*> **get** (*Selector selector*)
Get a selection of path/value from Zenoh.

Parameters

- **selector** – the *`Selector`* expressing the selection.

Throws

- ***ZException*** – if get failed.

Returns a collection of path/value.

put

public void **put** (*Path path*, *Value value*)
Put a path/value into Zenoh.

Parameters

- **path** – the *Path*. Can be absolute or relative to the workspace.
- **value** – the *Value*.

Throws

- ***ZException*** – if put failed.

put

public void **put** (*Path path*, String *value*)
Put a path/String into Zenoh.

Parameters

- **path** – the *Path*. Can be absolute or relative to the workspace.
- **value** – the value `java.lang.String`.

Throws

- ***ZException*** – if put failed.

put

public void **put** (*Path path*, long *value*)
Put a path/integer into Zenoh.

Parameters

- **path** – the *Path*. Can be absolute or relative to the workspace.
- **value** – the integer value as a long

Throws

- ***ZException*** – if put failed.

put

public void **put** (*Path path*, double *value*)
Put a path/float into Zenoh.

Parameters

- **path** – the *Path*. Can be absolute or relative to the workspace.
- **value** – the float value as a double.

Throws

- ***ZException*** – if put failed.

put

public void **put** (*Path path*, ByteBuffer *value*)
Put a path/ByteBuffer into Zenoh using the RAW encoding.

Parameters

- **path** – the *Path*. Can be absolute or relative to the workspace.
- **value** – the ByteBuffer.

Throws

- ***ZException*** – if put failed.

registerEval

public void **registerEval** (*Path path*, *Eval eval*)
Registers an evaluation function under the provided *Path*. The function will be evaluated in a dedicated thread, and thus may call any other Workspace operation.

Parameters

- **path** – the *Path* where the function can be triggered using *get(Selector)*
- **eval** – the evaluation function

Throws

- ***ZException*** – if registration failed.

remove

public void **remove** (*Path path*)
Remove a path/value from Zenoh.

Parameters

- **path** – the *Path* to be removed. Can be absolute or relative to the workspace.

Throws

- ***ZException*** – if remove failed.

subscribe

public *SubscriptionId* **subscribe** (*Selector selector*, *Listener listener*)
Subscribe to a selection of path/value from Zenoh.

Parameters

- **selector** – the *Selector* expressing the selection.
- **listener** – the *Listener* that will be called for each change of a path/value matching the selection.

Throws

- ***ZException*** – if subscribe failed.

Returns a *SubscriptionId*.

unregisterEval

public void **unregisterEval**(*Path path*)
Unregister a previously registered evaluation function.

Parameters

- **path** – the `Path` where the function has been registered

Throws

- ***ZException*** – if unregistration failed.

unsubscribe

public void **unsubscribe**(*SubscriptionId subid*)
Unregisters a previous subscription.

Parameters

- **subid** – the `SubscriptionId` to unregister

Throws

- ***ZException*** – if unsusbscribe failed.

update

public void **update**(*Path path*, *Value value*)
Update a path/value into Zenoh.

Parameters

- **path** – the `Path`. Can be absolute or relative to the workspace.
- **value** – a delta to be applied on the existing value.

Throws

- ***ZException*** – if update failed.

1.1.19 Zenoh

public class **Zenoh**
The Zenoh client API.

Methods

admin

public *Admin* **admin**()
Returns the `Admin` object that provides helper operations to administer Zenoh.

login

public static *Zenoh* **login**()

Establish a zenoh session through dynamic discovered peers/routers/brokers.

Throws

- ***ZException*** – if login fails.

Returns a *Zenoh* object.

login

public static *Zenoh* **login**(String *locator*)

Establish a zenoh session via a provided locator. Locator is a string representing the network endpoint to which establish the session. If the provided locator is `null`, login will perform some dynamic discovery and try to establish the session automatically. When not `null`, the locator must have the format: `tcp/<ip>:<port>` (for instance `tcp/127.0.0.1:7447`).

Parameters

- **locator** – the locator or `null`.

Throws

- ***ZException*** – if login fails.

Returns a *Zenoh* object.

login

public static *Zenoh* **login**(Properties *properties*)

Establish a zenoh session through dynamic discovered peers/routers/brokers.

Parameters

- **properties** – the Properties to be used for this session (e.g. "user", "password"...). Can be `null`.

Throws

- ***ZException*** – if login fails.

Returns a *Zenoh* object.

login

public static *Zenoh* **login**(String *locator*, Properties *properties*)

Establish a zenoh session via a provided locator. Locator is a string representing the network endpoint to which establish the session. If the provided locator is `null`, login will perform some dynamic discovery and try to establish the session automatically. When not `null`, the locator must have the format: `tcp/<ip>:<port>` (for instance `tcp/127.0.0.1:7447`).

Parameters

- **locator** – the locator or `null`.
- **properties** – the Properties to be used for this session (e.g. "user", "password"...). Can be `null`.

Throws

- ***ZException*** – if login fails.

Returns a *Zenoh* object.

logout

public void **logout**()
Terminates the Zenoh session.

workspace

public *Workspace* **workspace**()
Creates a Workspace on the root path. All relative *Selector* or *Path* used with this Workspace will be relative to this path.

Notice that all subscription listeners and eval callbacks declared in this workspace will be executed by the I/O thread. This implies that no long operations or other call to Zenoh shall be performed in those callbacks.

Returns a *Workspace*.

workspace

public *Workspace* **workspace**(*Path path*)
Creates a Workspace using the provided path. All relative *Selector* or *Path* used with this Workspace will be relative to this path.

Notice that all subscription listeners and eval callbacks declared in this workspace will be executed by the I/O thread. This implies that no long operations or other call to Zenoh shall be performed in those callbacks.

Parameters

- **path** – the Workspace's path.

Returns a *Workspace*.

workspaceWithExecutor

public *Workspace* **workspaceWithExecutor**(*Path path*)
Creates a Workspace using the provided path. All relative *Selector* or *Path* used with this Workspace will be relative to this path.

Notice that all subscription listeners and eval callbacks declared in this workspace will be executed by a CachedThreadPool. This is useful when listeners and/or callbacks need to perform long operations or need to call other Zenoh operations.

Parameters

- **path** – the Workspace's path.

Returns a *Workspace*.

workspaceWithExecutor

public *Workspace* **workspaceWithExecutor**()
Creates a Workspace using the provided path. All relative *Selector* or *Path* used with this Workspace will be relative to this path.

Notice that all subscription listeners and eval callbacks declared in this workspace will be executed by a CachedThreadPool. This is useful when listeners and/or callbacks need to perform long operations or need to call other Zenoh operations.

Returns a *Workspace*.

1.2 org.eclipse.zenoh.core

1.2.1 Timestamp

public class **Timestamp** implements Comparable<*Timestamp*>
Data structure representing a unique timestamp.

Constructors

Timestamp

protected **Timestamp**(long *time*, byte[] *clockId*)

Methods

compareTo

public int **compareTo**(*Timestamp o*)

equals

public boolean **equals**(Object *obj*)

getClockId

public byte[] **getClockId**()

Returns the unique identifier of the clock that created this Timestamp.

getTime

public long **getTime**()
Return the time as a 64-bit long, where:

- The higher 32-bit represent the number of seconds since midnight, January 1, 1970 UTC
- The lower 32-bit represent a fraction of 1 second.

WARNING: this time cannot be used with `java.util.Date.Date(long)`. Rather use *`getTimeAsInstant()`*.

Returns the time as a 64-bits NTP time.

getTimeAsInstant

public Instant **`getTimeAsInstant`**()

Returns the Timestamp's creation time as a `java.time.Instant`.

hashCode

public int **`hashCode`**()

toString

public String **`toString`**()

1.2.2 ZException

public class **`ZException`** extends Exception
An Exception raised by Zenoh.

Fields

ZN_FAILED_TO_OPEN_SESSION

public static final Integer **`ZN_FAILED_TO_OPEN_SESSION`**

ZN_INSUFFICIENT_IOBUF_SIZE

public static final Integer **`ZN_INSUFFICIENT_IOBUF_SIZE`**

ZN_INVALID_ADDRESS_ERROR

public static final Integer **`ZN_INVALID_ADDRESS_ERROR`**

ZN_IO_ERROR

public static final Integer **`ZN_IO_ERROR`**

ZN_MESSAGE_PARSE_ERROR

public static final Integer **`ZN_MESSAGE_PARSE_ERROR`**

ZN_PAYLOAD_HEADER_PARSE_ERROR

public static final Integer **ZN_PAYLOAD_HEADER_PARSE_ERROR**

ZN_PROPERTIES_PARSE_ERROR

public static final Integer **ZN_PROPERTIES_PARSE_ERROR**

ZN_PROPERTY_PARSE_ERROR

public static final Integer **ZN_PROPERTY_PARSE_ERROR**

ZN_RESOURCE_DECL_ERROR

public static final Integer **ZN_RESOURCE_DECL_ERROR**

ZN_TX_CONNECTION_ERROR

public static final Integer **ZN_TX_CONNECTION_ERROR**

ZN_UNEXPECTED_MESSAGE

public static final Integer **ZN_UNEXPECTED_MESSAGE**

Z_ARRAY_PARSE_ERROR

public static final Integer **Z_ARRAY_PARSE_ERROR**

Z_NO_ERROR_CODE

public static final Integer **Z_NO_ERROR_CODE**

Z_STRING_PARSE_ERROR

public static final Integer **Z_STRING_PARSE_ERROR**

Z_VLE_PARSE_ERROR

public static final Integer **Z_VLE_PARSE_ERROR**

Constructors

ZException

public **ZException**(String *message*)

ZException

public **ZException**(String *message*, Throwable *cause*)

ZException

public **ZException**(String *message*, int *errorCode*)

ZException

public **ZException**(String *message*, int *errorCode*, Throwable *cause*)

Methods

getErrorCode

public int **getErrorCode**()

getErrorCodeName

public String **getErrorCodeName**()

toString

public String **toString**()

CHAPTER 2

Zenoh-net API Reference

2.1 org.eclipse.zenoh.net

2.1.1 DataHandler

public interface **DataHandler**

A callback interface to be implemented for the reception of subscribed/stored resources. See *Session.declareSubscriber(String,SubMode,DataHandler)* and *Session.declareStorage(String,StorageHandler)*.

Methods

handleData

public void **handleData** (String *rname*, ByteBuffer *data*, *DataInfo* *info*)

The method that will be called on reception of data matching the subscribed or stored resource.

Parameters

- **rname** – the resource name of the received data.
- **data** – the received data.
- **info** – the *DataInfo* associated with the received data.

2.1.2 DataInfo

public class **DataInfo**

Data structure containing meta informations about the associated data.

Constructors

DataInfo

protected **DataInfo** (long *flags*, *Timestamp* *tstamp*, int *encoding*, int *kind*)

Methods

getEncoding

public int **getEncoding** ()

Returns the encoding of the data.

getKind

public int **getKind** ()

Returns the kind of the data.

getTimestamp

public *Timestamp* **getTimestamp** ()

Returns the unique timestamp at which the data has been produced.

2.1.3 Eval

public class **Eval**
An Eval (see *Session.declareEval(String,EvalCallback)*).

Constructors

Eval

protected **Eval** (zn_eva_t *eval*)

Methods

undeclare

public void **undeclare** ()
Undeclare the Eval.

Throws

- ***ZException*** – if undeclaration failed.

2.1.4 Publisher

public class **Publisher**
A Publisher (see *Session.declarePublisher(String)*).

Constructors

Publisher

protected **Publisher**(zn_pub_t *pub*)

Methods

streamCompactData

public void **streamCompactData**(ByteBuffer *data*)
Send data in a *compact_data* message for the resource published by this Publisher.

Parameters

- **data** – the data to be sent.

Throws

- ***ZException*** – if write fails.

streamData

public void **streamData**(ByteBuffer *data*)
Send data in a *stream_data* message for the resource published by this Publisher.

Parameters

- **data** – the data to be sent.

Throws

- ***ZException*** – if write fails.

streamData

public void **streamData**(ByteBuffer *data*, short *encoding*, short *kind*)
Send data in a *stream_data* message for the resource published by this Publisher.

Parameters

- **data** – the data to be sent.
- **encoding** – a metadata information associated with the published data that represents the encoding of the published data.
- **kind** – a metadata information associated with the published data that represents the kind of publication.

Throws

- ***ZException*** – if write fails.

undeclare

public void **undeclare**()
Undeclare the Publisher.

Throws

- ***ZException*** – if undeclaration failed.

2.1.5 QueryDest

public class **QueryDest** extends zn_query_dest_t
A data structure defining which storages or evals should be destination of a query (see *Session.query(String,String,ReplyCallback,QueryDest,QueryDest)*).

Methods

all

public static *QueryDest* **all**()

Returns a *QueryDest* with kind *Kind.ZN_ALL*.

bestMatch

public static *QueryDest* **bestMatch**()

Returns a *QueryDest* with kind *Kind.ZN_BEST_MATCH*.

complete

public static *QueryDest* **complete**()

Returns a *QueryDest* with kind *Kind.ZN_COMPLETE*.

complete

public static *QueryDest* **complete**(short *nb*)
Returns a *QueryDest* with kind *Kind.ZN_COMPLETE* and with the number of storages or evals that should be destination of the query.

Parameters

- **nb** – the number of storages or evals that should be destination of the query

Returns a *QueryDest* with kind *Kind.ZN_COMPLETE*.

none

public static *QueryDest* **none**()

Returns a *QueryDest* with kind *Kind.ZN_NONE*.

2.1.6 QueryDest.Kind

public enum **Kind**
The Query destination kind.

Enum Constants

ZN_ALL

public static final *QueryDest*.*Kind* **ZN_ALL**
All storages/evals.

ZN_BEST_MATCH

public static final *QueryDest*.*Kind* **ZN_BEST_MATCH**
The nearest complete storage/eval if there is one, all storages/evals if not.

ZN_COMPLETE

public static final *QueryDest*.*Kind* **ZN_COMPLETE**
Only complete storages/evals.

ZN_NONE

public static final *QueryDest*.*Kind* **ZN_NONE**
no storages/evals.

Methods

fromInt

public static *Kind* **fromInt**(short *numVal*)

value

public short **value**()

2.1.7 QueryHandler

public interface **QueryHandler**
A callback interface to be implemented for handling of queries on storages or evals. See *Session.declareStorage(String,StorageHandler)* and *Session.declareEval(String,QueryHandler)*.

Methods

handleQuery

public void **handleQuery**(String *rname*, String *predicate*, *RepliesSender repliesSender*)

The method that will be called on reception of query matching the stored/evaluated resource selection. The implementation must provide the data matching the resource *rname* by calling the *RepliesSender.sendReplies(Resource[])* method with the data as argument. The *RepliesSender.sendReplies(Resource[])* method MUST be called but accepts empty data array. This call can be made in the current Thread or in a different Thread.

Parameters

- **rname** – the resource name of the queried data.
- **predicate** – a string provided by the querier refining the data to be provided.
- **repliesSender** – a *RepliesSender* on which the *sendReplies()* function MUST be called with the provided data as argument.

2.1.8 RepliesSender

public final class **RepliesSender**

Class to be used in a *QueryHandler* implementation to send back replies to a query.

Methods

sendReplies

public void **sendReplies**(*Resource*[] *replies*)

Send back the replies to the query associated with this *RepliesSender* object.

Parameters

- **replies** – the replies.

2.1.9 ReplyHandler

public interface **ReplyHandler**

A callback interface to be implemented for the reception of replies for a query. See *Session.query(String,String,ReplyHandler)* and *Session.query(String,String,ReplyHandler,QueryDest,QueryDest)*.

Methods

handleReply

public void **handleReply**(*ReplyValue reply*)

The method that will be called on reception of replies to the query sent by *Session.query(String,String,ReplyHandler)* or *Session.query(String,String,ReplyHandler,QueryDest,QueryDest)*.

Parameters

- **reply** – is the actual reply.

2.1.10 ReplyValue

public class **ReplyValue**

A data structure containing one of the replies to a query (see *ReplyHandler.handleReply(ReplyValue)*).

Constructors

ReplyValue

protected **ReplyValue** (int *kind*, byte[] *srcid*, long *rsn*, String *rname*, ByteBuffer *data*, *DataInfo info*)

ReplyValue

protected **ReplyValue** (*Kind kind*, byte[] *srcid*, long *rsn*, String *rname*, ByteBuffer *data*, *DataInfo info*)

Methods

getData

public ByteBuffer **getData** ()

Returns the received data when `ReplyValue.kind` equals *Kind.ZN_STORAGE_DATA* or *Kind.ZN_EVAL_DATA*.

getInfo

public *DataInfo* **getInfo** ()

Returns some meta information about the received data when `ReplyValue.kind` equals *Kind.ZN_STORAGE_DATA* or *Kind.ZN_EVAL_DATA*.

getKind

public *Kind* **getKind** ()

Returns the Reply message kind.

getRname

public String **getRname** ()

Returns the resource name of the received data when `ReplyValue.kind` equals *Kind.ZN_STORAGE_DATA* or *Kind.ZN_EVAL_DATA*.

getRsn

public long **getRsn**()

Returns the sequence number of the reply from the identified storage or eval when `ReplyValue.kind` equals *Kind.ZN_STORAGE_DATA*, *Kind.ZN_STORAGE_FINAL*, *Kind.ZN_EVAL_DATA* or *Kind.ZN_EVAL_FINAL*.

getSrcId

public byte[] **getSrcId**()

Returns the unique identifier of the storage or eval that sent this reply when `ReplyValue.kind` equals *Kind.ZN_STORAGE_DATA*, *Kind.ZN_STORAGE_FINAL*, *Kind.ZN_EVAL_DATA* or *Kind.ZN_EVAL_FINAL*.

2.1.11 ReplyValue.Kind

public enum **Kind**
The reply message kind.

Enum Constants

ZN_EVAL_DATA

public static final *ReplyValue.Kind* **ZN_EVAL_DATA**
The reply contains some data from an eval.

ZN_EVAL_FINAL

public static final *ReplyValue.Kind* **ZN_EVAL_FINAL**
The reply indicates that no more data is expected from the specified eval.

ZN_REPLY_FINAL

public static final *ReplyValue.Kind* **ZN_REPLY_FINAL**
The reply indicates that no more replies are expected for the query.

ZN_STORAGE_DATA

public static final *ReplyValue.Kind* **ZN_STORAGE_DATA**
The reply contains some data from a storage.

ZN_STORAGE_FINAL

public static final *ReplyValue.Kind* **ZN_STORAGE_FINAL**
The reply indicates that no more data is expected from the specified storage.

Methods

fromInt

public static *Kind* **fromInt** (int *numVal*)

value

public int **value** ()

2.1.12 Resource

public class **Resource**
A resource with a name and a value (data).

Constructors

Resource

public **Resource** (String *rname*, ByteBuffer *data*, int *encoding*, int *kind*)

Methods

getData

public ByteBuffer **getData** ()

Returns the resource value.

getEncoding

public int **getEncoding** ()

Returns the encoding of the resource value.

getKind

public int **getKind** ()

Returns the kind of the resource.

getRname

public String **getRname** ()

Returns the resource name.

2.1.13 Rname

public class **Rname**
Utility class for resource name.

Methods

intersect

public static boolean **intersect** (String *rname1*, String *rname2*)
Return true if the resource name 'rname1' intersect with the resource name 'rname2'.

Parameters

- **rname1** – a resource name
- **rname2** – a resrouce name

2.1.14 Session

public class **Session**
A zenoh-net session.

Methods

LogException

protected static void **LogException** (Throwable *e*, String *message*)

close

public void **close** ()
Close the zenoh-net session.

Throws

- ***ZException*** – if close failed.

declareEval

public *Eval* **declareEval** (String *resource*, *QueryHandler handler*)
Declare an eval able to provide data matching the provided resource name **resource**.

Parameters

- **resource** – the resource to evaluate.
- **handler** – a *QueryHandler* subclass implementing the the callback function that will be called each time a query for data matching the evaluated resource name **resource** is received. The *QueryHandler.handleQuery(String, String,RepliesSender)* function MUST call the provided *RepliesSender.sendReplies(Resource[])* function with the resulting data. *RepliesSender.sendReplies(Resource[])* can be called with an empty array.

Throws

- ***ZException*** – if declaration fails.

Returns the Eval.

declarePublisher

public *Publisher* **declarePublisher** (String *resource*)
Declare a publication for resource name **resource**.

Parameters

- **resource** – the resource name to publish.

Throws

- ***ZException*** – if declaration fails.

Returns the zenoh *Publisher*.

declareStorage

public *Storage* **declareStorage** (String *resource*, *StorageHandler handler*)
Declare a storage for all data matching the provided resource name **resource**.

Parameters

- **resource** – the resource selection to store.
- **handler** – a *StorageHandler* subclass implementing the callback functions that will be called each time a data matching the stored resource name **resource** is received and each time a query for data matching the stored resource name **resource** is received. The `StorageHandler.handleQuery(String, String,RepliesSender)` function MUST call the provided *RepliesSender.sendReplies(Resource[])* function with the resulting data. *RepliesSender.sendReplies(Resource[])* can be called with an empty array.

Throws

- ***ZException*** – if declaration fails.

Returns the zenoh *Storage*.

declareSubscriber

public *Subscriber* **declareSubscriber** (String *resource*, *SubMode mode*, *DataHandler handler*)
Declare a subscribtion for all published data matching the provided resource name **resource**.

Parameters

- **resource** – the resource name to subscribe to.
- **mode** – the subscription mode.
- **handler** – a *DataHandler* subclass implementing the callback function that will be called each time a data matching the subscribed resource name **resource** is received.

Throws

- **ZException** – if declaration fails.

Returns the zenoh-net *Subscriber*.

info

public Map<Integer, byte[]> **info** ()

Returns a map of properties containing various informations about the established zenoh-net session.

open

public static *Session* **open** (String *locator*)

Open a zenoh-net session.

Parameters

- **locator** – a string representing the network endpoint to which establish the session. A typical locator looks like this : `"tcp/127.0.0.1:7447"`. If `null`, open() will scout and try to establish the session automatically.

Throws

- **ZException** – if session etablishment fails.

Returns a Zenoh object representing the openned zenoh session..

open

public static *Session* **open** (String *locator*, Map<Integer, byte[]> *properties*)

Open a zenoh-net session.

Parameters

- **locator** – a string representing the network endpoint to which establish the session. A typical locator looks like this : `"tcp/127.0.0.1:7447"`. If `null`, open() will scout and try to establish the session automatically.
- **properties** – a map of properties that will be used to establish and configure the zenoh session. **properties** will typically contain the `"username"` and `"password"` informations needed to establish the zenoh session with a secured infrastructure. It can be set to `null`.

Throws

- **ZException** – if session etablishment fails.

Returns a Zenoh object representing the openned zenoh session..

query

public void **query** (String *resource*, String *predicate*, *ReplyHandler* *handler*)

Query data matching resource name **resource**.

Parameters

- **resource** – the resource to query.
- **predicate** – a string that will be propagated to the storages and evals that should provide the queried data. It may allow them to filter, transform and/or compute the queried data. .
- **handler** – a *ReplyHandler* subclass implementing the callback function that will be called on reception of the replies of the query.

Throws

- ***ZException*** – if fails.

query

public void **query** (String *resource*, String *predicate*, *ReplyHandler* *handler*, *QueryDest* *dest_storages*, *QueryDest* *dest_evals*)

Query data matching resource name **resource**.

Parameters

- **resource** – the resource to query.
- **predicate** – a string that will be propagated to the storages and evals that should provide the queried data. It may allow them to filter, transform and/or compute the queried data. .
- **handler** – a *ReplyHandler* subclass implementing the callback function that will be called on reception of the replies of the query.
- **dest_storages** – a *QueryDest* indicating which matching storages should be destination of the query.
- **dest_evals** – a *QueryDest* indicating which matching evals should be destination of the query.

Throws

- ***ZException*** – if fails.

writeData

public void **writeData** (String *resource*, java.nio.ByteBuffer *payload*)

Send data in a *write_data* message for the resource **resource**.

Parameters

- **resource** – the resource name of the data to be sent.
- **payload** – the data.

Throws

- ***ZException*** – if write fails.

writeData

public void **writeData** (String *resource*, java.nio.ByteBuffer *payload*, short *encoding*, short *kind*)

Send data in a *write_data* message for the resource **resource**.

Parameters

- **resource** – the resource name of the data to be sent.

- **payload** – the data.
- **encoding** – a metadata information associated with the published data that represents the encoding of the published data.
- **kind** – a metadata information associated with the published data that represents the kind of publication.

Throws

- ***ZException*** – if write fails.

2.1.15 Storage

public class **Storage**
A Storage (see *Session.declareStorage(String,StorageCallback)*).

Constructors

Storage

protected **Storage**(zn_sto_t *sto*)

Methods

undeclare

public void **undeclare**()
Undeclare the Storage.

Throws

- ***ZException*** – if undeclaration failed.

2.1.16 StorageHandler

public interface **StorageHandler** extends *DataHandler*, *QueryHandler*
A callback interface to be implemented by a Storage.

2.1.17 SubMode

public class **SubMode** extends org.eclipse.*zenoh*.swig.zn_sub_mode_t
Subscription mode (used in *Session.declareSubscriber(String,SubMode,SubscriberCallback)*).

Methods

periodicPull

public static *SubMode* **periodicPull**(int *origin*, int *period*, int *duration*)

Returns a periodic pull subscription mode with the specified temporal properties.

periodicPush

public static *SubMode* **periodicPush** (int *origin*, int *period*, int *duration*)

Returns a periodic push subscription mode with the specified temporal properties.

pull

public static *SubMode* **pull** ()

Returns the pull subscription mode.

push

public static *SubMode* **push** ()

Returns the push subscription mode.

2.1.18 SubMode.Kind

public enum **Kind**
The subscription mode kind.

Enum Constants

ZN_PERIODIC_PULL_MODE

public static final *SubMode*.*Kind* **ZN_PERIODIC_PULL_MODE**

ZN_PERIODIC_PUSH_MODE

public static final *SubMode*.*Kind* **ZN_PERIODIC_PUSH_MODE**

ZN_PULL_MODE

public static final *SubMode*.*Kind* **ZN_PULL_MODE**

ZN_PUSH_MODE

public static final *SubMode*.*Kind* **ZN_PUSH_MODE**

Methods

fromInt

public static *Kind* **fromInt** (short *numVal*)

value

public short **value**()

2.1.19 Subscriber

public class **Subscriber**
A Subscriber (see *Session.declareSubscriber(String,SubMode,SubscriberCallback)*).

Constructors

Subscriber

protected **Subscriber**(zn_sub_t *sub*)

Methods

pull

public void **pull**()
Pull data for the *SubMode.Kind.ZN_PULL_MODE* or *SubMode.Kind.ZN_PERIODIC_PULL_MODE* subscribtion. The pulled data will be provided by calling the *DataHandler.handleData(String,java.nio.ByteBuffer,DataInfo)* function provided to the *Session.declareSubscriber(String,SubMode,DataHandler)* function.

Throws

- ***ZException*** – if pull failed.

undeclare

public void **undeclare**()
Undeclare the Subscriber.

Throws

- ***ZException*** – if undeclaration failed.

2.1.20 ZNet

public final class **ZNet**

Fields

INFO_PEER_KEY

public static final Integer **INFO_PEER_KEY**

INFO_PEER_PID_KEY

public static final Integer **INFO_PEER_PID_KEY**

INFO_PID_KEY

public static final Integer **INFO_PID_KEY**

PASSWD_KEY

public static final Integer **PASSWD_KEY**

USER_KEY

public static final Integer **USER_KEY**

Index

H

I

J

K

L

N

O

P

Q

R

S

T

U

V

W

Z