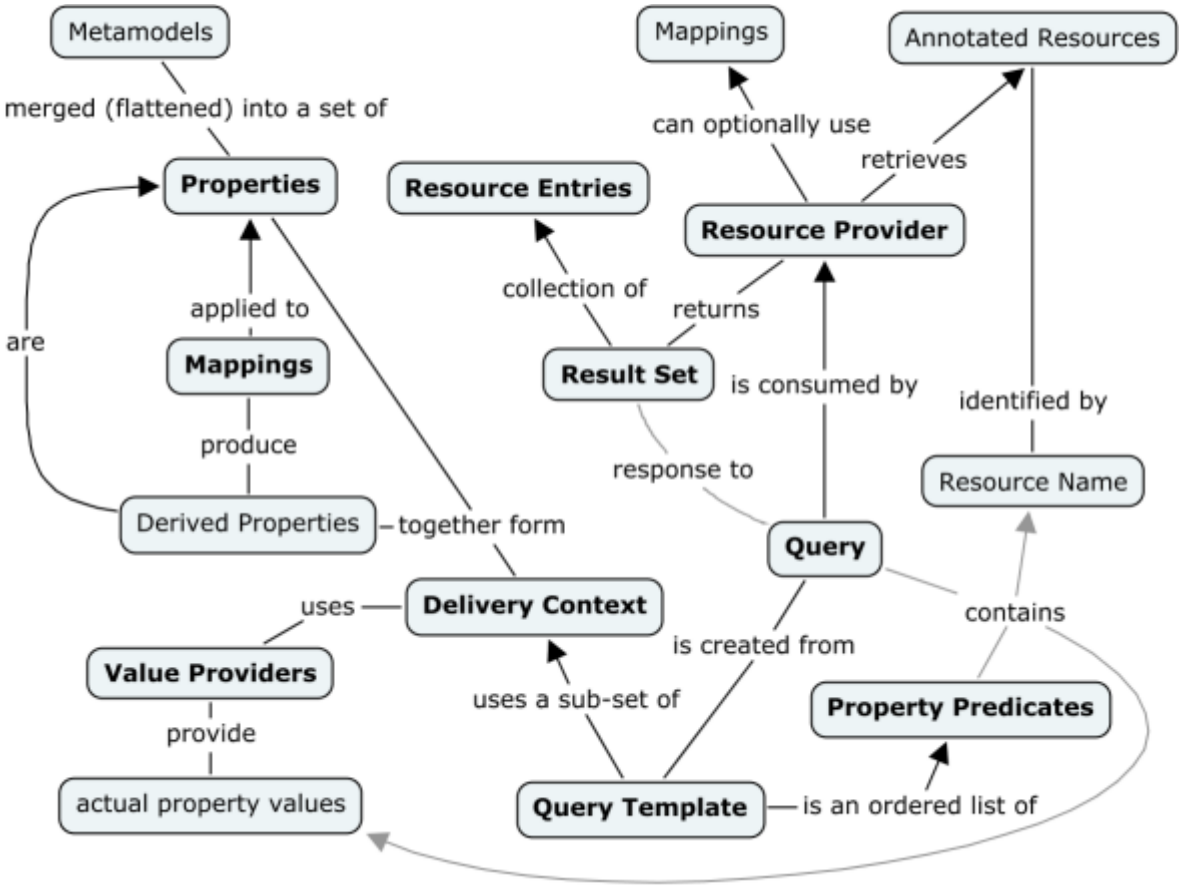
Versatile 1.0 API Reference

Jaroslav Gergic

Overview

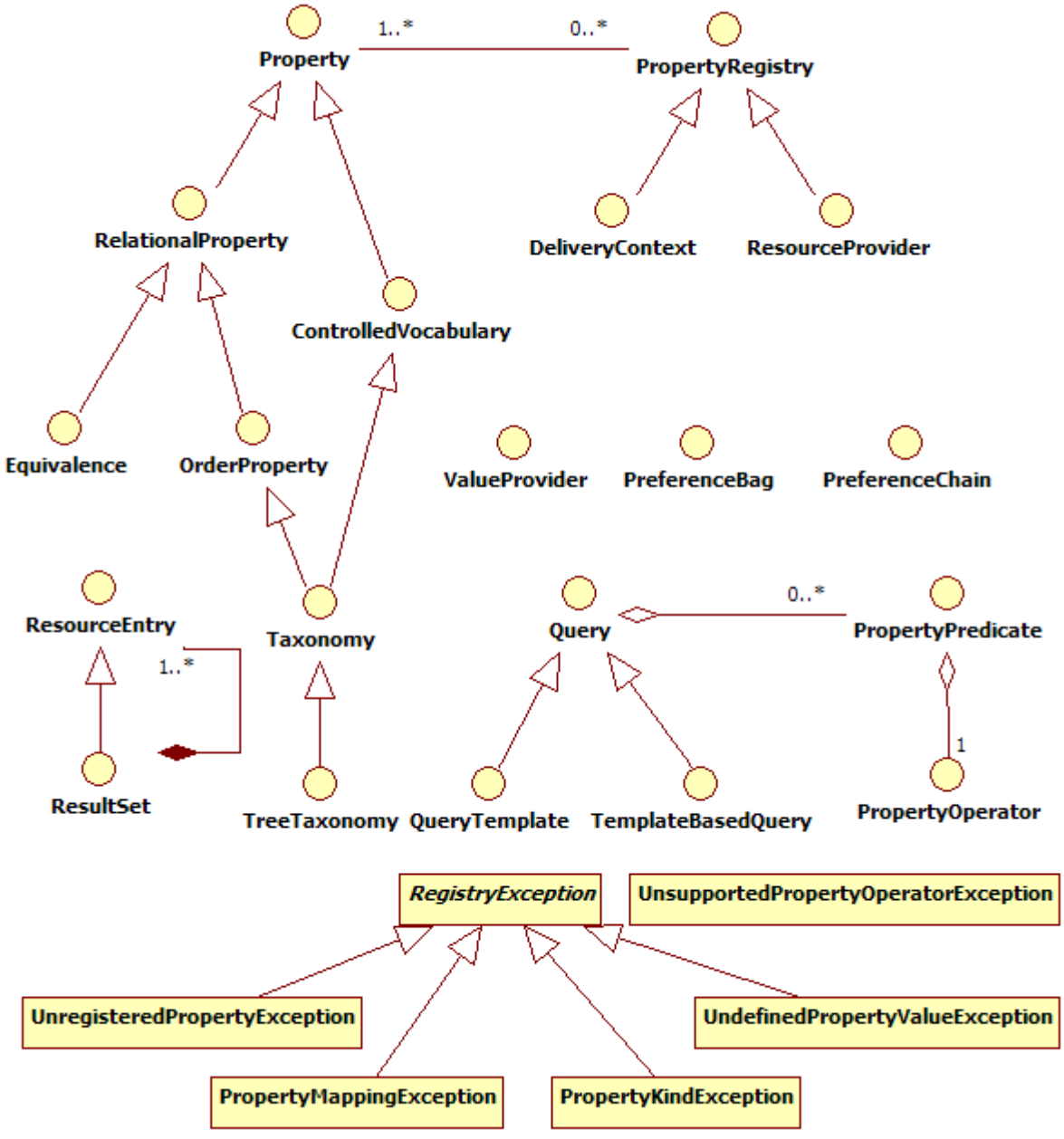
Versatile 1.0 API Reference.

Please visit us at http://dsrg.mff.cuni.cz/~gergic/versatile/.



Package
cz.cuni.versatile.api

The key elements of the Versatile API.



cz.cuni.versatile.api
Interface ControlledVocabulary

All Superinterfaces:
Property

All Subinterfaces:
Taxonomy, TreeTaxonomy

---

public interface **ControlledVocabulary**
extends Property

A controlled vocabulary property. Allows to enumerate all property values.
See Also:
Vocabulary, Taxonomy, Thesaurus, Ontology and a Meta-Model

Method Summary

Set	`getValueSet()` A set of all values a given property can have.
Iterator	`iterator()` An iterator over the value set.

Methods inherited from interface `cz.cuni.versatile.api.Property`

`getLocalName`, `getNamespace`, `getSeparator`, `getType`, `getUniqueName`

Methods

getValueSet

```
public Set getValueSet()
```

A set of all values a given property can have.

Returns:
a `Set` of all values.

---

iterator

```
public Iterator iterator()
```

An iterator over the value set.

Returns:
`Iterator` over the value set.

cz.cuni.versatile.api
Interface DeliveryContext

All Superinterfaces:
PropertyRegistry

public interface **DeliveryContext**
extends PropertyRegistry

`DeliveryContext` serves as the developer's "window to the outer-world". All the parameters needed to make the versioning decisions in the application scope of a given module or a component should be registered in its `DeliveryContext`. During application runtime, the `DeliveryContext` is used to acquire property values needed during instantiation of queries out of the pre-defined query templates. With a properly configured `DeliveryContext` the developer does not need to care about retrieving property values and/or fall-back (default-value) policies as all this should be already built-in in the `DeliveryContext`.

Remarks: The main purpose of using *derived* (mapped) properties in the `DeliveryContext` is to transform raw (typically domain-specific -- e.g. UAProf) meta-data into pre-processed application-specific properties which better correspond to the inherent logic of the application. The transformations in such a case are typically value adding (information adding): e.g. canonicalization, hierarchical classification. Alternatively, the usage of the property mappings can be as simple as property renaming (aliasing) due to the need to use multiple overlapping vocabularies (namespaces).

Method Summary

`Object`	`getValue(Property prop)` Returns the current value of the specified `Property`.
`Object`	`getValue(String uniqueName)` Returns the current value of the specified property.
`ValueProvider`	`getValueProvider(Property prop)` Returns the `ValueProvider` for the specified `Property`.
`boolean`	`hasValue(Property prop)` Checks whether the specified `Property` has value in the current state of the delivery context.
`boolean`	`hasValue(String uniqueName)` Checks whether the property of the specified `uniqueName` has value in the current state of the delivery context.
`void`	`registerProperty(Property prop, ValueProvider vp)` Registers a *leaf* property and its associated value provider.

Methods inherited from interface `cz.cuni.versatile.api.PropertyRegistry`

`getProperties`, `getProperty`, `getPropertyMapping`, `hasProperty`, `hasProperty`, `isMappedProperty`, `registerProperty`, `unregisterProperty`

Methods

registerProperty

```
public void registerProperty(Property prop,
        ValueProvider vp)
```

(continued from last page)

Registers a *leaf* property and its associated value provider. If the `Property` has already been registered, it redefines the `Property`

For registering a *derived* (mapped) property to the `DeliveryContext` use `PropertyRegistry.registerProperty(Property, PropertyMapping)`; the property being registered **must** be in the *range set* of the `PropertyMapping` and consequently, all the properties of the mapping's *domain set* must be already registered in the delivery context.

Parameters:

`prop` - a `Property` to register
`vp` - `ValueProvider` for the `Property`

See Also:

`PropertyRegistry.registerProperty(Property, PropertyMapping)`

---

getValueProvider

```
public ValueProvider getValueProvider(Property prop)
  throws UnregisteredPropertyException,
         PropertyKindException
```

Returns the `ValueProvider` for the specified `Property`.

Parameters:

`prop` - a `Property` instance

Returns:

`ValueProvider` for the specified `Property`

Throws:

`UnregisteredPropertyException` - in case the property is not registered
`PropertyKindException` - in case there is no `ValueProvider` registered for the given `Property`

---

hasValue

```
public boolean hasValue(Property prop)
  throws UnregisteredPropertyException
```

Checks whether the specified `Property` has value in the current state of the delivery context.

Parameters:

`prop` - a `Property` instance

Returns:

`true` if the property can be evaluated, `false` if the property value is undefined

Throws:

`UnregisteredPropertyException` - in case the specified `Property` in not registered

See Also:

`ValueProvider.hasValue()`

---

hasValue

```
public boolean hasValue(String uniqueName)
  throws UnregisteredPropertyException
```

Checks whether the property of the specified `uniqueName` has value in the current state of the delivery context.

Parameters:

(continued from last page)

`uniqueName` - a unique identifier of the property

Returns:

`true` if the property can be evaluated, `false` if the property value is undefined

Throws:

`UnregisteredPropertyException` - in case the specified `Property` in not registered

See Also:

`ValueProvider.hasValue()`

---

getValue

```
public Object getValue(Property prop)
  throws UnregisteredPropertyException,
         UndefinedPropertyValueException
```

Returns the current value of the specified `Property`. In case of a *leaf* property it directly invokes its `ValueProvider`. In case of a *derived* property it builds a dependency tree based on its potentially nested) property mappings and proceeds with a recursive bottom-up evaluation starting with the *leaf* properties.

Parameters:

`prop` - a `Property` instance

Returns:

the current value of the specified `Property`

Throws:

`UnregisteredPropertyException` - in case the specified `Property` in not registered

`UndefinedPropertyValueException` - in case the property value is undefined

---

getValue

```
public Object getValue(String uniqueName)
  throws UnregisteredPropertyException,
         UndefinedPropertyValueException
```

Returns the current value of the specified property. In case of a *leaf* property it directly invokes its `ValueProvider`. In case of a *derived* property it builds a dependency tree based on its potentially nested) property mappings and proceeds with a recursive bottom-up evaluation starting with the *leaf* properties.

Parameters:

`uniqueName` - a unique identifier of the property

Returns:

the current value of the specified property

Throws:

`UnregisteredPropertyException` - in case the property of the specified `uniqueName` in not registered

`UndefinedPropertyValueException` - in case the property value is undefined

cz.cuni.versatile.api
Interface Equivalence

All Superinterfaces:
RelationalProperty, Property

---

public interface **Equivalence**
extends RelationalProperty

`Equivalence` is a marker interface to denote a commonly used specialization of `RelationalProperty`.

It does not introduce any new methods, however, it imposes restrictions on several methods of its parent `RelationalProperty` interface. The following rules **must hold** for any `Equivalence` implementation to be a **valid** implementation:

- `isReflexive() = true`
- `isSymmetric() = true`
- `isTransitive() = true`
- `contains(Object dom, Object rng)` must be consistent with the above.

See Also:
`RelationalProperty`, Equivalence relation (Wikipedia)

Methods inherited from interface `cz.cuni.versatile.api.RelationalProperty`
`contains`, `isAntisymmetric`, `isAsymetric`, `isIrreflexive`, `isReflexive`, `isSymmetric`, `isTransitive`

Methods inherited from interface `cz.cuni.versatile.api.Property`
`getLocalName`, `getNamespace`, `getSeparator`, `getType`, `getUniqueName`

---

cz.cuni.versatile.api
Interface OrderProperty

All Superinterfaces:
RelationalProperty, Property

All Subinterfaces:
Taxonomy, TreeTaxonomy

---

public interface **OrderProperty**
extends RelationalProperty

`OrderProperty` is an interface to denote a commonly used specialization of `RelationalProperty`.

It encompasses all the various types of *order*: partial order, total order (linear order), strict order It imposes restrictions on several methods of its parent `RelationalProperty` interface. The following rules **must hold** for any `OrderProperty` implementation to be a **valid** implementation:

- `(isStrictOrder() = false) -> (isReflexive() = true)`
- `isAntisymmetric() = true`
- `isTransitive() = true`
- `contains(Object dom, Object rng)` must be consistent with the above.

* @author Jaroslav Gergic

Method Summary

`boolean`	`comparable(Object entryA, Object entryB)` Checks whether two values are comparable given a particular `OrderProperty`.
`Comparator`	`comparator()` Returns a comparator suitable for Java Collections utility classes.
`boolean`	`isPartialOrder()` Checks whether a particular `OrderProperty` defines a partial order.
`boolean`	`isStrictOrder()` Checks whether a particular `OrderProperty` defines a strict order.
`boolean`	`isTotalOrder()` Checks whether a particular `OrderProperty` defines a total order.

Methods inherited from interface `cz.cuni.versatile.api.RelationalProperty`

`contains`, `isAntisymmetric`, `isAsymetric`, `isIrreflexive`, `isReflexive`, `isSymmetric`, `isTransitive`

Methods inherited from interface `cz.cuni.versatile.api.Property`

`getLocalName`, `getNamespace`, `getSeparator`, `getType`, `getUniqueName`

Methods

(continued from last page)

comparator

```
public Comparator comparator()
```

Returns a comparator suitable for Java Collections utility classes.

Remarks: Please note, that unless `isTotalOrder() = true`, we consider an order to be a *partial* order and therefore the comparator instance returned by this method may not be suitable for usage by Java Collections framework, as it always assumes *total* (*linear*) order. If two values are not comparable (due to partial order) the `Comparator.compare()` method should return `0` (zero).

Returns:

a comparator instance

See Also:

`java.util.Collections`
`comparable(Object, Object)`

---

comparable

```
public boolean comparable(Object entryA,
          Object entryB)
```

Checks whether two values are comparable given a particular `OrderProperty`.
Remarks: Please note that if `isTotalOrder() = true` this method **must** always return `true`

Parameters:

`entryA` - a property value to compare
`entryB` - a property value to compare

Returns:

`true if contains(entryA, entryB) or contains(entryB, entryA)`

---

isTotalOrder

```
public boolean isTotalOrder()
```

Checks whether a particular `OrderProperty` defines a total order.

Returns:

`true if for all x,y: comparable(x, y) = true`

---

isPartialOrder

```
public boolean isPartialOrder()
```

Checks whether a particular `OrderProperty` defines a partial order.

Returns:

`not isTotalOrder()` (i.e. an opposite to total order)

---

isStrictOrder

```
public boolean isStrictOrder()
```

Checks whether a particular `OrderProperty` defines a strict order.

Returns:

`not isReflexive()` (i.e. `true` for irreflexive relations)

cz.cuni.versatile.api
Interface PreferenceBag

All Known Implementing Classes:
PreferenceBagImpl

---

public interface **PreferenceBag**
extends Set

A marker interface to denote an unordered set of values. The reason we are not directly using `java.util.Set` is that we want to distinguish properties whose values are sets (type `Set`), which need to be treated as individuals (i.e. comparing entire sets for equality, etc.) versus the cases when a property of another type (e.g. `String`) happens to have multiple values.

In general, a `ValueProvider` for any `Property`, regardless of its type, can return `PreferenceBag`, a collection of multiple values, instead of a single value of the given type. The semantics of a property value being a `PreferenceBag` is equivalent to logical `OR` operator, or even more precisely to the `IN` construct of *SQL*:

```
SELECT * FROM Table WHERE Attribute IN (...)
```

Remarks: `PreferenceBag` corresponds to the RDF:Bag construct. It is used to let the client to enumerate all acceptable options without stating their relative preference.

Methods inherited from interface `java.util.Set`
`add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray`

Methods inherited from interface `java.util.Collection`
`add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray`

---

cz.cuni.versatile.api
Interface PreferenceChain

All Known Implementing Classes:
PreferenceChainImpl

---

public interface **PreferenceChain**
extends List

A marker interface to denote an ordered list of values. The reason we are not directly using `java.util.List` is that we want to distinguish properties whose values are lists (type `List`), which need to be treated as individuals (i.e. comparing entire lists for equality, etc.) versus the cases when a property of another type (e.g. `String`) happens to have multiple values in descending order of significance.

In general, a `ValueProvider` for any `Property`, regardless of its type, can return `PreferenceChain` as a collection of multiple values instead of a single value of a given type. The semantics of a property value being a `PreferenceChain` is equivalent to `if, (else if)[1..(n-1)]` sequence, i.e., the values should be tried one after another in the order of the `PreferenceChain` until a match is found.

Remarks: `PreferenceChain` corresponds to the RDF:Sequence construct. It is used to let the client to enumerate all acceptable options and also to express its preferences regarding the listed options.

Methods inherited from interface `java.util.List`
`add, add, addAll, addAll, clear, contains, containsAll, equals, get, hashCode, indexOf, isEmpty, iterator, lastIndexOf, listIterator, listIterator, remove, remove, removeAll, retainAll, set, size, subList, toArray, toArray`

Methods inherited from interface `java.util.Collection`
`add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray`

---

cz.cuni.versatile.api
Interface Property

All Subinterfaces:
ControlledVocabulary, Taxonomy, TreeTaxonomy, RelationalProperty, Equivalence, OrderProperty, Taxonomy, TreeTaxonomy

---

public interface **Property**
extends

A meta-data property interface. This interface defines the essential attributes common to all Versatile properties. Custom properties can be introduced:

- *statically* by implementing this interface or one of its sub-typed interfaces defined in the `cz.cuni.versatile.api` package
- *dynamically* by using generic implementations of this interface and its sub-types defined in the `cz.cuni.versatile.api` package

---

Method Summary

`String`	`getLocalName()` A local property name within a particular namespace.
`String`	`getNamespace()` A property namespace.
`String`	`getSeparator()` Namespace separator.
`Class`	`getType()` A type which is used to represent values of the property.
`String`	`getUniqueName()` A fully qualified property name.

Methods

getNamespace

`public String getNamespace()`

A property namespace. The namespace can be an arbitrary string, however, it is highly recommended to follow one of the established naming schemas like XML Namespaces (URLs) or Java package names.

Returns:
property namespace

See Also:
Namespaces in XML
Package Names (Java Language Specification)

---

getLocalName

`public String getLocalName()`

(continued from last page)

A local property name within a particular namespace.

Returns:
locally unique property name

getUniqueName

`public String getUniqueName()`

A fully qualified property name. Equivalent to invoking: `getNamespace() + getSeparator() + getLocalName()`

Returns:
fully qualified property name.

getSeparator

`public String getSeparator()`

Namespace separator. A string used to separate namespace and local name when generating fully qualified name. For example '`.`' (dot) in case of following Java naming conventions, '#' in case of following XML naming conventions:

- `com.mycompany.mypackage.MyProperty` (*Note:* the last dot is considered as the separator, the former ones are a part of the namespace)
- `http://www.mycompany.com/mypackage/#MyProperty`

Returns:
separator between the namespace and the locale name.

getType

`public Class getType()`

A type which is used to represent values of the property. All actual property values must be instances of that type. Can be one of the built-in Java types like `java.lang.String`, `java.lang.Integer`, `java.lang.Boolean` or can be any other custom type, e.g. `javax.ccpp.uaprof.DimensionAttribute`.

Returns:
class object representing the type of the property.

cz.cuni.versatile.api
Class PropertyKindException

```
java.lang.Object
   ┗-java.lang.Throwable
      ┗-java.lang.Exception
         ┗-cz.cuni.versatile.api.RegistryException
            ┗-cz.cuni.versatile.api.PropertyKindException
```

All Implemented Interfaces:
Serializable

---

public class **PropertyKindException**
extends RegistryException

Thrown when trying to obtain a `PropertyMapping` for a *leaf* property or wise versa, when trying to retrieve a `ValueProvider` for a *derived* property.

Constructor Summary

public	PropertyKindException()
public	PropertyKindException(String message)
public	PropertyKindException(Throwable cause)
public	PropertyKindException(String message, Throwable cause)

Methods inherited from class `java.lang.Throwable`

`fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString`

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

PropertyKindException

`public` **`PropertyKindException`**`()`

(continued from last page)

PropertyKindException

```
public PropertyKindException(String message)
```

PropertyKindException

```
public PropertyKindException(Throwable cause)
```

PropertyKindException

```
public PropertyKindException(String message,
                             Throwable cause)
```

cz.cuni.versatile.api
Class PropertyMappingException

```
java.lang.Object
   ┗-java.lang.Throwable
      ┗-java.lang.Exception
         ┗-cz.cuni.versatile.api.RegistryException
            ┗-cz.cuni.versatile.api.PropertyMappingException
```

All Implemented Interfaces:
Serializable

---

public class **PropertyMappingException**
extends RegistryException

Thrown when

- the direction (*domain* versus *range*) of the `PropertyMapping` being registered is incorrect with the respect to underlying `PropertyRegistry` semantics.
- the `PropertyMapping` does not have the `Property` being registered neither in its *domain* nor *range*
- the dependencies of the given `PropertyMapping` can not satisfied by the already registered properties of the underlying `PropertyRegistry`

Constructor Summary

public	PropertyMappingException()
public	PropertyMappingException(String message)
public	PropertyMappingException(Throwable cause)
public	PropertyMappingException(String message, Throwable cause)

Methods inherited from class `java.lang.Throwable`

`fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString`

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

PropertyMappingException

`public` **`PropertyMappingException`**`()`

(continued from last page)

PropertyMappingException

```
public PropertyMappingException(String message)
```

Parameters:
message

PropertyMappingException

```
public PropertyMappingException(Throwable cause)
```

Parameters:
cause

PropertyMappingException

```
public PropertyMappingException(String message,
                                Throwable cause)
```

Parameters:
message
cause

cz.cuni.versatile.api
Interface PropertyOperator

All Known Implementing Classes:
PropertyOperators

public interface **PropertyOperator**
extends

`PropertyOperator` interface represents a relational or a functional operator used in `PropertyPredicate` to express meta-data constraints and preferences.
See Also:
`PropertyPredicate`

Method Summary

`int`	`getId()` The unique ID of the property operator.
`String`	`getName()` A human-readable property operator name - just for logging and debugging.
`boolean`	`isAssertive()` Flag whether the operator is *assertive* or whether it allows for an approximate matching (*fall -back, constraint relaxing*)
`boolean`	`isExtrinsic()` Flag whether the operator is *extrinsic Extrinsic* operators rely on the relations externally provided by the properties (e.g.
`boolean`	`isIntrinsic()` Flag whether the operator is *intrinsic*.

Methods

getId

`public int getId()`

The unique ID of the property operator. Used for manipulating the property predicates programmatically (to support `switch/case` statement).

Returns:
the unique ID of the property operator

getName

`public String getName()`

A human-readable property operator name - just for logging and debugging.

Returns:
a human-readable property operator name.

isIntrinsic

`public boolean isIntrinsic()`

Flag whether the operator is *intrinsic*. *Intrinsic* operators use methods of the actual property values for comparison, so they rely on the existing methods of Java objects (e.g. `Object.equals()` or `Comparable.compareTo()`)

Remarks: An operator is either *intrinsic* or *extrinsic*, never both.

Returns:
`true` is the property operator is *intrinsic*

isExtrinsic

`public boolean isExtrinsic()`

Flag whether the operator is *extrinsic Extrinsic* operators rely on the relations externally provided by the properties (e.g. `OrderProperty`, `Equivalence`)

Remarks: An operator is either *intrinsic* or *extrinsic*, never both.

Returns:
`true` is the property operator is *extrinsic*

isAssertive

`public boolean isAssertive()`

Flag whether the operator is *assertive* or whether it allows for an approximate matching (*fall-back*, *constraint relaxing*)

Returns:
`true` if the operator does not allow for any constraint relaxing

cz.cuni.versatile.api
Interface PropertyPredicate

public interface **PropertyPredicate**
extends

`PropertyPredicate` represents a single meta-data constraint or a preference. An ordered list of property predicates together with additional settings forms a `QueryTemplate`. A `QueryTemplate` can instantiate a particular multi-variant resource `Query`.

See Also:
`QueryTemplate`, `Query`

Method Summary

`Object`	`getArguments()` Additional arguments (besides the `Property` and the *property value*) required to evaluate the predicate.
`PropertyOperator`	`getOperator()` A `PropertyOperator` this predicate applies to its associated `Property`
`Property`	`getProperty()` A `Property` this predicate applies to.
`Object`	`getPropertyValue()` Returns the actual `Property` value.
`void`	`setPropertyValue(Object value)` Sets the actual `Property` value.

Methods

getProperty

`public Property getProperty()`

A `Property` this predicate applies to.

Returns:
a `Property` this predicate applies to

getOperator

`public PropertyOperator getOperator()`

A `PropertyOperator` this predicate applies to its associated `Property`

Returns:
a `PropertyOperator` used by this predicate

See Also:
`PropertyOperators`

(continued from last page)

getArguments

`public Object getArguments()`

Additional arguments (besides the `Property` and the *property value*) required to evaluate the predicate.

Returns:
an operator-specific data structure capturing the additional operator arguments, can be `null`

See Also:
`PropertyOperators`

---

getPropertyValue

`public Object getPropertyValue()`

Returns the actual `Property` value. The value is determined at the time of `Query` instantiation which means that if this method is invoked on a `PropertyPredicate` while being a part of a `QueryTemplate`, the method most likely returns `null`.

Returns:
the actual `Property` value at the time of `Query` instantiation.

---

setPropertyValue

`public void setPropertyValue(Object value)`

Sets the actual `Property` value. This method is invoked during `Query` creation process when the actual property values are being substituted from the `DeliveryContext`.

Parameters:
`value` - the actual `Property` value

See Also:
`QueryTemplate.newQuery(String)`

cz.cuni.versatile.api
Interface PropertyRegistry

All Subinterfaces:
DeliveryContext, ResourceProvider

public interface **PropertyRegistry**
extends

A common super-type for all interfaces which need to keep track of properties and their dependencies via property mappings.

Method Summary

Set	getProperties() Returns a read-only view of the currently registered properties.
Property	getProperty(String uniqueName) Returns a Property corresponding to the given uniqueName
PropertyMapping	getPropertyMapping(Property prop) Returns the PropertyMapping for the given Property.
boolean	hasProperty(Property prop) Check whether the given Property is already registered to this registry.
boolean	hasProperty(String uniqueName) Check whether a property with the given uniqueName is already registered to this registry.
boolean	isMappedProperty(Property prop) Checks, whether the given Property is a *leaf* or a property *derived* using a PropertyMapping.
void	registerProperty(Property prop, PropertyMapping pm) Registers a *derived* property and its value-providing PropertyMapping.
void	unregisterProperty(Property prop) Unregisters the given Property from the registry.

Methods

hasProperty

```
public boolean hasProperty(Property prop)
```

Check whether the given Property is already registered to this registry.

Parameters:
prop - a property instance

Returns:
true if the property is already registered, false otherwise

(continued from last page)

hasProperty

```
public boolean hasProperty(String uniqueName)
```

Check whether a property with the given `uniqueName` is already registered to this registry.

Parameters:
`uniqueName` - a unique identifier of the property

Returns:
`true` if the property is already registered, `false` otherwise

See Also:
`Property.getUniqueName()`

---

getProperty

```
public Property getProperty(String uniqueName)
  throws UnregisteredPropertyException
```

Returns a `Property` corresponding to the given `uniqueName`

Parameters:
`uniqueName` - a unique identifier of the property

Returns:
a `Property` corresponding to the given `uniqueName`

Throws:
`UnregisteredPropertyException` - in case a property the given `uniqueName` is not registered

---

isMappedProperty

```
public boolean isMappedProperty(Property prop)
  throws UnregisteredPropertyException
```

Checks, whether the given `Property` is a *leaf* or a property *derived* using a `PropertyMapping`.

Parameters:
`prop` - a property instance

Returns:
`true` if the given property is a *derived* property, `false` otherwise

Throws:
`UnregisteredPropertyException` - in case the property is not registered

---

getPropertyMapping

```
public PropertyMapping getPropertyMapping(Property prop)
  throws UnregisteredPropertyException,
         PropertyKindException
```

Returns the `PropertyMapping` for the given `Property`.

Parameters:
`prop` - a `Property` instance

Returns:

(continued from last page)

`PropertyMapping` registered for the given `Property`.

Throws:

`UnregisteredPropertyException` - in case the property is not registered
`PropertyKindException` - in case there is no `PropertyMapping` registered for the given `Property`

---

registerProperty

```
public void registerProperty(Property prop,
          PropertyMapping pm)
  throws PropertyMappingException
```

Registers a *derived* property and its value-providing `PropertyMapping`. If the `Property` has already been registered, it redefines the `Property`

Parameters:

`prop` - a `Property` to register
`pm` - a `PropertyMapping` for the given `Property`

Throws:

`PropertyMappingException` - in case the *domain* and/or *range* of the mapping is incorrect with the respect to the semantics of the underlying `PropertyRegistry` implementation.

---

unregisterProperty

```
public void unregisterProperty(Property prop)
  throws UnregisteredPropertyException
```

Unregisters the given `Property` from the registry.

Parameters:

`prop` - a `Property` to unregister

Throws:

`UnregisteredPropertyException` - in case the property is not registered

---

getProperties

```
public Set getProperties()
```

Returns a read-only view of the currently registered properties.

Returns:

a read-only view of the currently registered properties.

cz.cuni.versatile.api
Interface Query

All Subinterfaces:
QueryTemplate, TemplateBasedQuery

---

public interface **Query**
extends

Multi-variant resource query (a data structure abstraction). `Query` is a data structure encapsulating all the information necessary to retrieve a resource possibly existing in many different variants and flavors:

- *resource name* (mandatory) which uniquely identifies a *resource* in the scope of a particular `ResourceProvider` instance; unlike property names, resource names are not globally unique in Versatile.
  (*resource* means a versioned entity, not a particular version/variant of that entity)
- *ordered list of property predicates* (optional, empty by default) which specifies additional meta-data constraints and/or preferences (constraints are expressed using *assertive* operators, preferences using *constraint-relaxing* operators)
- *n-best size* (optional, default = 1) which determines how many results should be returned (maximum)
- *score threshold* (optional, default = 0.0) which determines the lowest acceptable score value, all values are accepted by default
- *scoring factor* (optional, default = 0.99) which determines relative significance of property predicates when calculating the score

Remarks:

- `Query` instances are typically created as a spin-off of the `QueryTemplate` object.
- `Query` is meant to be an immutable object: once it gets instantiated, it never changes
- `Query` implementations should override `Object.equals()` and implement an efficient comparison algorithm to detect whether two queries are identical
- the two points above are to ensure `ResourceProvider` implementations can more easily implement caching for improved performance.

See Also:
`ResourceProvider`, `QueryTemplate`, `TemplateBasedQuery`

Field Summary

`public static final`	`BIASED_SCORING_FACTOR` Biased scoring factor. Value: **`0.1`**
`public static final`	`DEFAULT_MATCH_SCORE` Default match score is `0.0`, which means all values. Value: **`0.0`**
`public static final`	`DEFAULT_N_BEST` Default N-best is 1-best. Value: **`1`**
`public static final`	`DEFAULT_SCORING_FACTOR` Default scoring factor. Value: **`0.99`**
`public static final`	`EXACT_MATCH_SCORE` Exact match score is `1.0`, which means no constraint relaxing (fall-back) activity took place. Value: **`1.0`**

public static final	NEUTRAL_SCORING_FACTOR NEUTRAL_SCORING_FACTOR = 1.0 makes all property predicates equally significant. Value: **1.0**

Method Summary

int	getNBest() Returns the N-best (the expected size of the result set) setting for this query.
List	getPredicates() An ordered list of meta-data constraints.
String	getResourceName() Resource name which uniquely identifies a resource within a scope of a particular ResourceProvider instance.
double	getScoreThreshold() Returns the resource score threshold setting for this query.
double	getScoringFactor() Returns the scoring factor used by the scoring function for this query.

Fields

DEFAULT_N_BEST

`public static final int` **`DEFAULT_N_BEST`**

Default N-best is 1-best.
Constant value: **1**

DEFAULT_MATCH_SCORE

`public static final double` **`DEFAULT_MATCH_SCORE`**

Default match score is `0.0`, which means all values.
Constant value: **0.0**

EXACT_MATCH_SCORE

`public static final double` **`EXACT_MATCH_SCORE`**

Exact match score is `1.0`, which means no constraint relaxing (fall-back) activity took place.
Constant value: **1.0**

See Also:
PropertyOperators.BEST_MATCH

DEFAULT_SCORING_FACTOR

`public static final double` **`DEFAULT_SCORING_FACTOR`**

Default scoring factor. `DEFAULT_SCORING_FACTOR = 0.99`, which is semantically very close `NEUTRAL_SCORING_FACTOR`, but the small penalty of this scoring factor effectively prevents ambiguity of the result scores.
Constant value: **0.99**

NEUTRAL_SCORING_FACTOR

`public static final double` **`NEUTRAL_SCORING_FACTOR`**

`NEUTRAL_SCORING_FACTOR = 1.0` makes all property predicates equally significant. the score calculated for an individual result entry by the scoring function will be `1/(1 + m)`, where `m` is the *magnitude* of the vector defined as the distance of the result entry from the original query in an N-dimensional *Euclidean* space, each property corresponding to one dimension.

Remarks: This setting can easily generate ambiguous results - multiple result entries with the same score.
Constant value: **`1.0`**

BIASED_SCORING_FACTOR

`public static final double` **`BIASED_SCORING_FACTOR`**

Biased scoring factor. `BIASED_SCORING_FACTOR = 0.1` strongly inclines towards the properties at the beginning of the predicate list, i.e. these are the most significant. This is analogous to the order of significance of the individual digits in a decimal number.
Constant value: **`0.1`**

Methods

getNBest

`public int` **`getNBest`**`()`

Returns the N-best (the expected size of the result set) setting for this query. N-best mostly applies in combination with *bestMatch* or another constraint relaxing operator which affects the *score* of the `ResultSet` entries. Assertive operators have no effect on the score rating. N-best represents the maximum number of result entries which can be returned.

Returns:
the N-best setting for this query (a positive integer)

getScoreThreshold

`public double` **`getScoreThreshold`**`()`

Returns the resource score threshold setting for this query. Result items with a lower score will be skipped from the result set. The score threshold is a number in the interval `<Query#DEFAULT_MATCH_SCORE, Query#EXACT_MATCH_SCORE>` (`<0.0, 1.0>`).

Returns:
the resource score threshold setting for this query.

getScoringFactor

`public double` **`getScoringFactor`**`()`

Returns the scoring factor used by the scoring function for this query. The scoring factor determines relative significance of property predicates. It has a significant impact on results in cases when there are multiple property predicates using *constraint-relaxing* operator like *bestMatch*. The scoring factor is a number in the interval `(0, Query#NEUTRAL_SCORING_FACTOR>`.

Returns:
a scoring factor used when evaluating this query

See Also:
`QueryTemplate.setScoringFactor(double)`

getResourceName

`public String getResourceName()`

Resource name which uniquely identifies a resource within a scope of a particular `ResourceProvider` instance. Unlike property names, resource names are not necessarily globally unique in Versatile).

Returns:
resource name (a unique resource key)

getPredicates

`public List getPredicates()`

An ordered list of meta-data constraints. The `PropertyPredicates` are ordered in descending order of their significance, the significance of the predicates affects the resource score in case there are one or more predicates using *bestMatch* or another constraint relaxing operator.

Returns:
an ordered list of `PropertyPredicate` objects, the list can be empty

See Also:
`PropertyPredicate`

cz.cuni.versatile.api
Interface QueryTemplate

All Superinterfaces:
Query

---

public interface **QueryTemplate**
extends Query

`QueryTemplate` represents the Versatile API to the underlying "meta-data query language". In a typical case, the actual `Query` objects are not created directly but they are instantiated using `newQuery` method of the `QueryTemplate` class. The instances of the `QueryTemplate` class serve the following purposes:

1. define a set of meta-data constraints and preferences (an ordered list of predicates)
2. define the query evaluation preferences (N-best, score threshold, scoring factor)
3. allow to re-use the same settings for multiple different queries (with different resource name)
4. allow to fully automate property value evaluation and substitution during query instantiation (automatically retrieving property values for each predicate from the associated delivery context)

The interface provides a generic method for adding property predicates (`addPredicate`) and a set of methods allowing a short-hand notation for all the Versatile predefined operators.
See Also:
`PropertyOperators`

Fields inherited from interface `cz.cuni.versatile.api.Query`
`BIASED_SCORING_FACTOR`, `DEFAULT_MATCH_SCORE`, `DEFAULT_N_BEST`, `DEFAULT_SCORING_FACTOR`, `EXACT_MATCH_SCORE`, `NEUTRAL_SCORING_FACTOR`

Method Summary

`void`	`add_Equal(String uniqueName)` Add a meta-data predicate using a given property and the `PropertyOperators#EQ` (*equals*) intrinsic operator.
`void`	`add_GE(String uniqueName)` Add a meta-data predicate using a given property and the `PropertyOperators#GE` (*greater than or equals*) intrinsic operator.
`void`	`add_GT(String uniqueName)` Add a meta-data predicate using a given property and the `PropertyOperators#GT` (*greater than*) intrinsic operator.
`void`	`add_LE(String uniqueName)` Add a meta-data predicate using a given property and the `PropertyOperators#LE` (*less than or equals*) intrinsic operator.
`void`	`add_LT(String uniqueName)` Add a meta-data predicate using a given property and the `PropertyOperators#LT` (*less than*) intrinsic operator.
`void`	`addAssert(String uniqueName)` Add a meta-data predicate using a given property and the `PropertyOperators#ASSERT` (*assert*) extrinsic operator.

void	addAssertInv(String uniqueName) Add a meta-data predicate using a given property and the PropertyOperators#ASSERT_INV (*assertInv*) extrinsic operator.
void	addAssertLevel(String uniqueName, int level) Add a meta-data predicate using a given property and the PropertyOperators#ASSERT_LEVEL (*assertLevel*) extrinsic operator.
void	addBestMatch(String uniqueName) Add a meta-data predicate using a given property and the PropertyOperators#BEST_MATCH (*bestMatch*) extrinsic operator.
void	addComparable(String uniqueName) Add a meta-data predicate using a given property and the PropertyOperators#COMPARABLE (*comparable*) extrinsic operator.
void	addEquivalent(String uniqueName) Add a meta-data predicate using a given property and the PropertyOperators#EQUIVALENT (*equivalent*) extrinsic operator.
void	addIsAncestor(String uniqueName) Add a meta-data predicate using a given property and the PropertyOperators#IS_ANCESTOR (*isAncestor*) extrinsic operator.
void	addIsChild(String uniqueName) Add a meta-data predicate using a given property and the PropertyOperators#IS_CHILD (*isChild*) extrinsic operator.
void	addIsDescendant(String uniqueName) Add a meta-data predicate using a given property and the PropertyOperators#IS_DESCENDANT (*isDescendant*) extrinsic operator.
void	addIsParent(String uniqueName) Add a meta-data predicate using a given property and the PropertyOperators#IS_PARENT (*isParent*) extrinsic operator.
void	addPredicate(PropertyPredicate pp) A generic method for adding a meta-data predicate.
DeliveryContext	getDeliveryContext() Returns the associated delivery context used to evaluate property values and substitute them to the property predicates during query instantiation.
TemplateBasedQuery	newQuery(String resourceName) Query factory method.
void	setNBest(int nbest) Sets the N-best size for queries based on this template.
void	setScoreThreshold(double threshold) Sets the score threshold for queries based on this template.
void	setScoringFactor(double scoringFactor) Sets the scoring factor for queries based on this template.

Methods inherited from interface cz.cuni.versatile.api.Query

getNBest, getPredicates, getResourceName, getScoreThreshold, getScoringFactor

(continued from last page)

Methods

setNBest

```
public void setNBest(int nbest)
```

Sets the N-best size for queries based on this template.

Parameters:

`nbest` - a positive integer

See Also:

`Query.DEFAULT_N_BEST`

---

setScoreThreshold

```
public void setScoreThreshold(double threshold)
```

Sets the score threshold for queries based on this template.

Parameters:

`threshold` - a number in the interval `<Query#DEFAULT_MATCH_SCORE, Query#EXACT_MATCH_SCORE>` (`<0.0, 1.0>`)

See Also:

`Query.DEFAULT_MATCH_SCORE`
`Query.EXACT_MATCH_SCORE`

---

setScoringFactor

```
public void setScoringFactor(double scoringFactor)
```

Sets the scoring factor for queries based on this template. The scoring factor determines relative significance of property predicates. It has a significant impact on results in cases when there are multiple property predicates using *constraint-relaxing* operator like *bestMatch*.

Parameters:

`scoringFactor` - a number in the interval `(0, Query#NEUTRAL_SCORING_FACTOR>`.

See Also:

`Query`

---

getDeliveryContext

```
public DeliveryContext getDeliveryContext()
```

Returns the associated delivery context used to evaluate property values and substitute them to the property predicates during query instantiation.

Returns:

the associated delivery context

---

addPredicate

```
public void addPredicate(PropertyPredicate pp)
  throws UnregisteredPropertyException
```

A generic method for adding a meta-data predicate. It allows to build queries using extension property operators which are not pre-defined in Versatile.

(continued from last page)

Parameters:
`pp` - a meta-data predicate describing one particular aspect of a desired resource.

Throws:
`UnregisteredPropertyException` - if the associated `DeliveryContext` does not recognize a given property (an attribute of the `PropertyPredicate` parameter)

See Also:
`PropertyOperators`

---

add_Equal

```
public void add_Equal(String uniqueName)
  throws UnregisteredPropertyException
```

Add a meta-data predicate using a given property and the `PropertyOperators#EQ` (*equals*) intrinsic operator.

Parameters:
`uniqueName` - a unique name of the `Property`

Throws:
`UnregisteredPropertyException` - if the associated `DeliveryContext` does not recognize a given property

See Also:
`PropertyOperators.EQ`

---

add_GT

```
public void add_GT(String uniqueName)
  throws UnregisteredPropertyException
```

Add a meta-data predicate using a given property and the `PropertyOperators#GT` (*greater than*) intrinsic operator.

Parameters:
`uniqueName` - a unique name of the `Property`

Throws:
`UnregisteredPropertyException` - if the associated `DeliveryContext` does not recognize a given property

See Also:
`PropertyOperators.GT`

---

add_LT

```
public void add_LT(String uniqueName)
  throws UnregisteredPropertyException
```

Add a meta-data predicate using a given property and the `PropertyOperators#LT` (*less than*) intrinsic operator.

Parameters:
`uniqueName` - a unique name of the `Property`

Throws:
`UnregisteredPropertyException` - if the associated `DeliveryContext` does not recognize a given property

See Also:
`PropertyOperators.LT`

add_GE

```
public void add_GE(String uniqueName)
  throws UnregisteredPropertyException
```

Add a meta-data predicate using a given property and the `PropertyOperators#GE` (*greater than or equals*) intrinsic operator.

Parameters:

`uniqueName` - a unique name of the `Property`

Throws:

`UnregisteredPropertyException` - if the associated `DeliveryContext` does not recognize a given property

See Also:

`PropertyOperators.GE`

add_LE

```
public void add_LE(String uniqueName)
  throws UnregisteredPropertyException
```

Add a meta-data predicate using a given property and the `PropertyOperators#LE` (*less than or equals*) intrinsic operator.

Parameters:

`uniqueName` - a unique name of the `Property`

Throws:

`UnregisteredPropertyException` - if the associated `DeliveryContext` does not recognize a given property

See Also:

`PropertyOperators.LE`

addAssert

```
public void addAssert(String uniqueName)
  throws UnregisteredPropertyException,
         UnsupportedPropertyOperatorException
```

Add a meta-data predicate using a given property and the `PropertyOperators#ASSERT` (*assert*) extrinsic operator.

Parameters:

`uniqueName` - a unique name of the `Property`

Throws:

`UnregisteredPropertyException` - if the associated `DeliveryContext` does not recognize a given property
`UnsupportedPropertyOperatorException` - is the operator is not applicable to a given property type

See Also:

`PropertyOperators.ASSERT`

addAssertInv

```
public void addAssertInv(String uniqueName)
  throws UnregisteredPropertyException,
         UnsupportedPropertyOperatorException
```

(continued from last page)

Add a meta-data predicate using a given property and the `PropertyOperators#ASSERT_INV` (*assertInv*) extrinsic operator.

Parameters:

`uniqueName` - a unique name of the `Property`

Throws:

`UnregisteredPropertyException` - if the associated `DeliveryContext` does not recognize a given property
`UnsupportedPropertyOperatorException` - is the operator is not applicable to a given property type

See Also:

`PropertyOperators.ASSERT_INV`

---

addAssertLevel

```
public void addAssertLevel(String uniqueName,
        int level)
  throws UnregisteredPropertyException,
        UnsupportedPropertyOperatorException
```

Add a meta-data predicate using a given property and the `PropertyOperators#ASSERT_LEVEL` (*assertLevel*) extrinsic operator.

Parameters:

`uniqueName` - a unique name of the `Property`

Throws:

`UnregisteredPropertyException` - if the associated `DeliveryContext` does not recognize a given property
`UnsupportedPropertyOperatorException` - is the operator is not applicable to a given property type

See Also:

`PropertyOperators.ASSERT_LEVEL`

---

addEquivalent

```
public void addEquivalent(String uniqueName)
  throws UnregisteredPropertyException,
        UnsupportedPropertyOperatorException
```

Add a meta-data predicate using a given property and the `PropertyOperators#EQUIVALENT` (*equivalent*) extrinsic operator.

Parameters:

`uniqueName` - a unique name of the `Property`

Throws:

`UnregisteredPropertyException` - if the associated `DeliveryContext` does not recognize a given property
`UnsupportedPropertyOperatorException` - is the operator is not applicable to a given property type

See Also:

`PropertyOperators.EQUIVALENT`

---

addComparable

```
public void addComparable(String uniqueName)
  throws UnregisteredPropertyException,
        UnsupportedPropertyOperatorException
```

Add a meta-data predicate using a given property and the `PropertyOperators#COMPARABLE` (*comparable*) extrinsic operator.

(continued from last page)

Parameters:

`uniqueName` - a unique name of the `Property`

Throws:

`UnregisteredPropertyException` - if the associated `DeliveryContext` does not recognize a given property
`UnsupportedPropertyOperatorException` - is the operator is not applicable to a given property type

See Also:

`PropertyOperators.COMPARABLE`

---

addIsParent

```
public void addIsParent(String uniqueName)
  throws UnregisteredPropertyException,
         UnsupportedPropertyOperatorException
```

Add a meta-data predicate using a given property and the `PropertyOperators#IS_PARENT` (*isParent*) extrinsic operator.

Parameters:

`uniqueName` - a unique name of the `Property`

Throws:

`UnregisteredPropertyException` - if the associated `DeliveryContext` does not recognize a given property
`UnsupportedPropertyOperatorException` - is the operator is not applicable to a given property type

See Also:

`PropertyOperators.IS_PARENT`

---

addIsChild

```
public void addIsChild(String uniqueName)
  throws UnregisteredPropertyException,
         UnsupportedPropertyOperatorException
```

Add a meta-data predicate using a given property and the `PropertyOperators#IS_CHILD` (*isChild*) extrinsic operator.

Parameters:

`uniqueName` - a unique name of the `Property`

Throws:

`UnregisteredPropertyException` - if the associated `DeliveryContext` does not recognize a given property
`UnsupportedPropertyOperatorException` - is the operator is not applicable to a given property type

See Also:

`PropertyOperators.IS_CHILD`

---

addIsAncestor

```
public void addIsAncestor(String uniqueName)
  throws UnregisteredPropertyException,
         UnsupportedPropertyOperatorException
```

Add a meta-data predicate using a given property and the `PropertyOperators#IS_ANCESTOR` (*isAncestor*) extrinsic operator.

Parameters:

`uniqueName` - a unique name of the `Property`

Throws:

(continued from last page)

`UnregisteredPropertyException` - if the associated `DeliveryContext` does not recognize a given property
`UnsupportedPropertyOperatorException` - is the operator is not applicable to a given property type

See Also:
`PropertyOperators.IS_ANCESTOR`

---

addIsDescendant

```
public void addIsDescendant(String uniqueName)
  throws UnregisteredPropertyException,
         UnsupportedPropertyOperatorException
```

Add a meta-data predicate using a given property and the `PropertyOperators#IS_DESCENDANT` (*isDescendant*) extrinsic operator.

Parameters:
`uniqueName` - a unique name of the `Property`

Throws:
`UnregisteredPropertyException` - if the associated `DeliveryContext` does not recognize a given property
`UnsupportedPropertyOperatorException` - is the operator is not applicable to a given property type

See Also:
`PropertyOperators.IS_DESCENDANT`

---

addBestMatch

```
public void addBestMatch(String uniqueName)
  throws UnregisteredPropertyException,
         UnsupportedPropertyOperatorException
```

Add a meta-data predicate using a given property and the `PropertyOperators#BEST_MATCH` (*bestMatch*) extrinsic operator.

Parameters:
`uniqueName` - a unique name of the `Property`

Throws:
`UnregisteredPropertyException` - if the associated `DeliveryContext` does not recognize a given property
`UnsupportedPropertyOperatorException` - is the operator is not applicable to a given property type

See Also:
`PropertyOperators.BEST_MATCH`

---

newQuery

```
public TemplateBasedQuery newQuery(String resourceName)
```

(continued from last page)

`Query` factory method. It takes a unique resource name as its input (`resourceName`) and creates an immutable `Query` object using the current `QueryTemplate` settings and its current predicate list. An implementation of this method performs the following tasks:

1. clone the list of the existing property predicates and query template settings (N-best, score threshold, scoring factor)
2. scan the list of all property predicates and build a set of properties used by this query
3. use the associated `DeliveryContext` to provide values for all the properties, each property is evaluated only once, even if it appears in more than one predicate
4. substitute the property values to the predicates
5. instantiate a `Query` using the data gathered in the above steps

Remarks:

- Given the above process, the individual `Query` instances can not be re-used repeatedly over time as the property values may have changed in the meantime

Parameters:

`resourceName` - a resource name uniquely identifying a particular resource within the scope of the target `ResourceProvider`

Returns:

a query object ready to be passed to the intended `ResouceProvider`

cz.cuni.versatile.api
Class RegistryException

```
java.lang.Object
   ┗-java.lang.Throwable
      ┗-java.lang.Exception
         ┗-cz.cuni.versatile.api.RegistryException
```

All Implemented Interfaces:
Serializable

Direct Known Subclasses:
PropertyKindException, PropertyMappingException, UndefinedPropertyValueException, UnregisteredPropertyException

---

public abstract class **RegistryException**
extends Exception

Thrown for `PropertyRegistry` related issues. Please refer to its sub-classes for individual fault conditions.

Constructor Summary

public	RegistryException()
public	RegistryException(String message)
public	RegistryException(Throwable cause)
public	RegistryException(String message, Throwable cause)

Methods inherited from class `java.lang.Throwable`

`fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString`

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

RegistryException

`public` **`RegistryException`**`()`

(continued from last page)

RegistryException

```
public RegistryException(String message)
```

RegistryException

```
public RegistryException(Throwable cause)
```

RegistryException

```
public RegistryException(String message,
                         Throwable cause)
```

cz.cuni.versatile.api
Interface RelationalProperty

All Superinterfaces:
Property

All Subinterfaces:
Equivalence, OrderProperty, Taxonomy, TreeTaxonomy

---

public interface **RelationalProperty**
extends Property

RelationalProperty is a property which defines a binary relation over the set of its values.

The `RelationalProperty` interface declares methods allowing to query algebraic relational properties (*reflexivity*, *symmetry* and *transitivity*). It also defines `contains(Object dom, Object rng)` method which checks if a pair of property values is *related* given a particular property definition.

The algebraic definitions of relational properties used thereafter adhere to the definitions given at the *Binary Relation* article of Wikipedia.

See Also:
Relations over a set (*Binary relation*, Wikipedia), `RelationalOperator`, `RelationalOperatorsRegistry`

Method Summary

`boolean`	`contains(Object dom, Object rng)` Query whether two property values are *related* in terms of a given relational property.
`boolean`	`isAntisymmetric()` Antisymmetry check.
`boolean`	`isAsymetric()` Asymetry check.
`boolean`	`isIrreflexive()` Irreflexivity check.
`boolean`	`isReflexive()` Reflexivity check.
`boolean`	`isSymmetric()` Symmetry check.
`boolean`	`isTransitive()` Transitivity check.

Methods inherited from interface `cz.cuni.versatile.api.Property`

`getLocalName`, `getNamespace`, `getSeparator`, `getType`, `getUniqueName`

Methods

(continued from last page)

isReflexive

`public boolean isReflexive()`

Reflexivity check. Query whether a relational property is *reflexive*:

`for all x: contains(x, x) = true`
(An opposite to *irreflexive* property.)

Returns:
`true` if a particular relational property is reflexive

---

isIrreflexive

`public boolean isIrreflexive()`

Irreflexivity check. Query whether a relational property is *irreflexive*:

`for all x: contains(x, x) = false`
(An opposite to *reflexive* property.)

Returns:
`true` if a particular relational property is irreflexive

---

isSymmetric

`public boolean isSymmetric()`

Symmetry check. Query whether a relational property is *symmetric*:

`for all x,y: (contains(x, y) = true) -> (contains(y, x) = true)`

Returns:
`true` if a particular relational property is symmetric

---

isAntisymmetric

`public boolean isAntisymmetric()`

Antisymmetry check. Query whether a relational property is *antisymmetric*:

`for all x,y: ((contains(x, y) = true and contains(y, x) = true) -> x = y`

Returns:
`true` if a particular relational property is antisymmetric

---

isAsymetric

`public boolean isAsymetric()`

Asymetry check. Query whether a relational property is *asymmetric*:

`for all x,y: (contains(x, y) = true) -> (contains(y, x) = false)`

Returns:
`true` if a particular relational property is asymmetric

---

isTransitive

`public boolean isTransitive()`

(continued from last page)

Transitivity check. Query whether a relational property is *transitive*:

```
for all x,y,z: (contains(x, y) = true and contains(y, z) = true) -> (contains(x, z) = true)
```

Returns:

`true` if a particular relational property is transitive

---

contains

```
public boolean contains(Object dom,
         Object rng)
```

Query whether two property values are *related* in terms of a given relational property. The input to this method is an ordered pair,the order is significant, unless the relational property declares it is a *symmetric* property.

The `contains()` method **must adhere** to the relational properties advertised in terms of the `is*()` methods. If the behavior of the `contains()` method is not consistent with the `is*()` query methods, the implementation of the relational property is considered a **defective** implementation.

When testing/validating relational properties, one may try to employ relational closure operators, apply them on the the property and measure whether the property's `contains()` gives the same results as the `contains()` method of the instance closure operators were applied to.

Parameters:

`dom` - the first member of the ordered pair (a member of the "domain" set)

`rng` - the second member of the ordered pair (a member of the "range" set)

Returns:

`true` is an ordered pair `[dom, rng]` is a member of the binary relation corresponding to a given `RelationalProperty`

See Also:

`RelationalOperator`

cz.cuni.versatile.api
Interface ResourceEntry

All Subinterfaces:
ResultSet

public interface **ResourceEntry**
extends

`ResourceEntry` represents an element of the N-best result list produced by a `ResourceProvider` in a response to a particular `Query`. The `ResourceEntry` object contains not only the resource itself but also its meta-data annotations (property/value pairs). The ability to provide resources alongside their meta-data is especially important when using constraint-relaxing (fall-back) types of searches, when the results returned may only approximate the `Query`.

Method Summary

`int`	`getIndex()` The zero-based index of this resource entry in the parent `ResultSet`.
`Map`	`getProperties()` Return the meta-data annotations of the resource (property/value pairs).
`ResultSet`	`getResultSet()` Returns the parent `ResultSet` of this `ResourceEntry`.
`double`	`getScore()` Return the relative quality score of this resource entry with the respect to the `Query`.
`Object`	`getValue()` Returns the actual resource (the value of the `ResourceEntry`).

Methods

getResultSet

`public ResultSet getResultSet()`

Returns the parent `ResultSet` of this `ResourceEntry`.

Returns:
the parent `ResultSet` of this `ResourceEntry`

getValue

`public Object getValue()`

Returns the actual resource (the value of the `ResourceEntry`).

Returns:
the actual resource (the value of the `ResourceEntry`).

(continued from last page)

getProperties

`public Map getProperties()`

Return the meta-data annotations of the resource (property/value pairs). The keys of the map are `Property` instances, the values of the map are the property values assigned as meta-data annotations to the resource represented by this entry.

Returns:
the meta-data annotations of the resource (property/value pairs).

---

getScore

`public double getScore()`

Return the relative quality score of this resource entry with the respect to the `Query`. The number must be from the interval `(Query#DEFAULT_MATCH_SCORE, Query#EXACT_MATCH_SCORE>` which corresponds to `(0.0, 1.0>`.

Returns:
the quality score of this resource entry.

See Also:
`Query`

---

getIndex

`public int getIndex()`

The zero-based index of this resource entry in the parent `ResultSet`.

Returns:
zero-based index of this resource entry in the parent `ResultSet`.

cz.cuni.versatile.api
Interface ResourceProvider

All Superinterfaces:
PropertyRegistry

---

public interface **ResourceProvider**
extends PropertyRegistry

`ResourceProvider` is an abstraction of a multi-variant resource repository or a class factory. The implementations of the interface are assumed to be specialized for a particular role or an environment. For example:

- a string resource bundle with multiple properties meta-data annotations attached to each variant of a resource (thus supporting many variants of the same resource string)
- a class and or component factory (producing pre-configured individual instances according to selected properties of the delivery context)
- a meta-class factory (meta-component factory) (resource name being an interface name and selected properties of the delivery context used to look-up the most suitable class/component factory for a given situation)

In a typical configuration, we envision multiple purpose-specific `ResourceProvider` instances to correspond to a single general-purpose `DeliveryContext` within a particular application scope (e.g. a module or a component). On the other hand, resource providers are more-likely to be re-used across multiple application scopes.

Remarks: The main purpose of using the property mappings in the `ResourceProvider` is to allow for a kind of *reverse* mapping: The mappings in the `DeliveryContext` transform meta-data from domain-specific to application-centric data structures. The `ResourceProvider` may need to transform these data back as it consumes queries using the application-centric properties, while the resource repository can have the data annotated using the original raw meta-data or a third-party meta-data. Of course, in a perfect world, the resource repositories are cleansed and tagged by the application-centric annotations, but this may not be always possible in practice.

An important consequence of the above is that when registering a mapped property in the `ResourceProvider`, the semantics of the `PropertyRegistry.registerProperty(Property, PropertyMapping)` is exactly the opposite to the `DeliveryContext`: we are not transforming **to** the mapped property but instead **from** the property being mapped to the built-in properties of the `ResourceProvider`. (Naturally, there is no need to register the *leaf* properties in the case of the `ResourceProvider` as they are exactly those built-in properties specific to a particular `ResourceProvider` implementation and/or instance.)

See Also:
`PropertyRegistry.registerProperty(Property, PropertyMapping)`

Method Summary

`ResultSet`	`get(Query q)` Get an N-best collection of resources matching the given `Query`.
`ResultSet`	`get(String resourceName, QueryTemplate qt)` Get an N-best collection of resources matching the given `resourceName` and query template.
`Object`	`getValue(String resourceName, QueryTemplate qt)` Get the closest match without the attached meta-data annotations.

Methods inherited from interface `cz.cuni.versatile.api.PropertyRegistry`

`getProperties`, `getProperty`, `getPropertyMapping`, `hasProperty`, `hasProperty`, `isMappedProperty`, `registerProperty`, `unregisterProperty`

Methods

get

```
public ResultSet get(Query q)
  throws UnregisteredPropertyException,
         MissingResourceException
```

Get an N-best collection of resources matching the given `Query`. The method always returns at least a collection which one entry (or fires an exception).

Parameters:

`q` - a `Query` to process

Returns:

an N-best collection of resources matching the given `Query`

Throws:

`UnregisteredPropertyException` - in case a property in the query is not known (registered) to the `ResourceProvider`
`MissingResourceException` - in case there are no resource entries matching the `Query`

get

```
public ResultSet get(String resourceName,
         QueryTemplate qt)
  throws UnregisteredPropertyException,
         MissingResourceException
```

Get an N-best collection of resources matching the given `resourceName` and query template.

(Syntax sugar: Semantically equivalent to calling `ResourceProvider.get(qt.newQuery(resourceName))`.)

Parameters:

`resourceName` - a unique identifier of a resource in the scope of the `ResourceProvider`
(*resource* means a versioned entity, not a particular version/variant of that entity)
`qt` - a `QueryTemplate` specifying the meta-data constraints (preferences) and other query settings

Returns:

an N-best collection of resources matching the given `Query`

Throws:

`UnregisteredPropertyException` - in case a property in the query is not known (registered) to the `ResourceProvider`
`MissingResourceException` - in case there are no resource entries matching the `Query`

getValue

```
public Object getValue(String resourceName,
         QueryTemplate qt)
  throws UnregisteredPropertyException,
         MissingResourceException
```

Get the closest match without the attached meta-data annotations.

(Syntax sugar: Semantically equivalent to calling `ResourceProvider.get(qt.newQuery(resourceName)).getValue()`.)

Parameters:

(continued from last page)

`resourceName` - a unique identifier of a resource in the scope of the `ResourceProvider` (*resource* means a versioned entity, not a particular version/variant of that entity)
`qt` - a `QueryTemplate` specifying the meta-data constraints (preferences) and other query settings

Returns:

the value from the first `ResourceEntry` of the N-best `ResultSet`

Throws:

`UnregisteredPropertyException` - in case a property in the query is not known (registered) to the `ResourceProvider`
`MissingResourceException` - in case there are no resource entries matching the `Query`

cz.cuni.versatile.api
Interface ResultSet

All Superinterfaces:
ResourceEntry

---

public interface **ResultSet**
extends ResourceEntry

`ResultSet` the N-best result list produced by a `ResourceProvider` in a response to a particular `Query`. The `ResultSet` is ordered in descending order with the respect to the *score* of the individual `ResourceEntry` items. `ResultSet` itself implements the interface `ResourceEntry` and thus exposes facets to its users:

1. an ordered collection of `ResourceEntry` items
2. a shortcut accessor to the first (0-index) `ResourceEntry`

This approach has been chosen because of the default N-best size, which is equal to 1, and at the same time, a `ResultSet` always contains at least one item. In a typical situation, the user does not need (and does not want) to deal with a collection of result items and just wants to pick the first one (the closest match) item.

Method Summary

`ResourceEntry`	`get(int i)` Returns a `ResourceEntry` at a given index.
`Query`	`getQuery()` Returns the `Query` this `ResultSet` is a response to.
`Iterator`	`iterator()` Returns the descending order iterator which reflects the underlying score-ordered collection.
`int`	`size()` Returns the size of the `ResultSet`.

Methods inherited from interface `cz.cuni.versatile.api.ResourceEntry`

`getIndex`, `getProperties`, `getResultSet`, `getScore`, `getValue`

Methods

size

`public int size()`

Returns the size of the `ResultSet`.

Returns:
the size of the `ResultSet`.

---

iterator

`public Iterator iterator()`

(continued from last page)

Returns the descending order iterator which reflects the underlying score-ordered collection.

Returns:
the descending order iterator which reflects the underlying score-ordered collection.

---

get

`public ResourceEntry get(int i)`

Returns a `ResourceEntry` at a given index.

Parameters:
`i` - a zero-based index, must not exceed `size() -1`

Returns:
a `ResourceEntry` at a given index.

Throws:
`IndexOutOfBoundsException` - in case the given index exceeded the size of the result set.

---

getQuery

`public Query getQuery()`

Returns the `Query` this `ResultSet` is a response to.

Returns:
the `Query` this `ResultSet` is a response to.

cz.cuni.versatile.api
Interface Taxonomy

All Superinterfaces:
OrderProperty, RelationalProperty, Property

All Subinterfaces:
TreeTaxonomy

---

public interface **Taxonomy**
extends ControlledVocabulary, OrderProperty

A taxonomy property. Taxonomy represents a hierarchical structure of concepts and sub-concepts (terms/sub-terms).

- Versatile requires each taxonomy to have a *root* (*top*) node of the hierarchy.
- A taxonomy does not necessarily be a tree structure – a node can have multiple parents.
- There is an inherent *partial* order imposed by the hierarchical structure: a node is always considered to be strictly less (<) than all its ancestors.
- This definition of order resonates well with the notion of an *all-encompassing* root node of the taxonomy. (the *universal concept* of a concept/sub-concept hierarchy)

See Also:
Vocabulary, Taxonomy, Thesaurus, Ontology and a Meta-Model, `TreeTaxonomy`

Method Summary

`Iterator`	`getAncestorIterator(Object entry, Comparator cmp)` Iterate over the ancestor hierarchy bottom-up in a well-defined (deterministic) order.
`Set`	`getAncestors(Object entry)` Returns ancestors of the context node.
`Set`	`getChildren(Object entry)` Returns children of the context node.
`Set`	`getDescendants(Object entry)` Returns descendants of the context node.
`Object`	`getLCA(Object[] entries)` Least Common Ancestor (LCA).
`Set`	`getParents(Object entry)` Returns parents of the context node.
`Object`	`getRoot()` Returns the root of the taxonomy.
`boolean`	`isAncestor(Object entry, Object descendant)` Ancestry check.
`boolean`	`isChild(Object entry, Object parent)` Child check.
`boolean`	`isDescendant(Object entry, Object ancestor)` Descendancy check.

boolean	isParent(Object entry, Object child) Parent check.
boolean	isRoot(Object entry) Checks whether a given value is the taxonomy root.

Methods inherited from interface cz.cuni.versatile.api.ControlledVocabulary

getValueSet, iterator

Methods inherited from interface cz.cuni.versatile.api.Property

getLocalName, getNamespace, getSeparator, getType, getUniqueName

Methods inherited from interface cz.cuni.versatile.api.OrderProperty

comparable, comparator, isPartialOrder, isStrictOrder, isTotalOrder

Methods inherited from interface cz.cuni.versatile.api.RelationalProperty

contains, isAntisymmetric, isAsymetric, isIrreflexive, isReflexive, isSymmetric, isTransitive

Methods inherited from interface cz.cuni.versatile.api.Property

getLocalName, getNamespace, getSeparator, getType, getUniqueName

Methods

getRoot

```
public Object getRoot()
```

Returns the root of the taxonomy.

Returns:
root node of the taxonomy.

getParents

```
public Set getParents(Object entry)
```

Returns parents of the context node.

Parameters:
`entry` - a context node to compare to

Returns:
a `Set` of parents of the context node

getChildren

```
public Set getChildren(Object entry)
```

Returns children of the context node.

(continued from last page)

Parameters:
`entry` - a context node to compare to

Returns:
a `Set` of children of the context node

getAncestors

```
public Set getAncestors(Object entry)
```

Returns ancestors of the context node.

Parameters:
`entry` - a context node to compare to

Returns:
a `Set` of ancestors of the context node

getDescendants

```
public Set getDescendants(Object entry)
```

Returns descendants of the context node.

Parameters:
`entry` - a context node to compare to

Returns:
a `Set` of descendants of the context node

isRoot

```
public boolean isRoot(Object entry)
```

Checks whether a given value is the taxonomy root. Equivalent to: `getRoot().equals(entry)`

Parameters:
`entry` - a context node to compare to

Returns:
`true` if a given node is the taxonomy root, `false` otherwise

isParent

```
public boolean isParent(Object entry,
         Object child)
```

Parent check. (Parent-child relation.)

Parameters:
`entry` - a context node to compare to
`child` - a test node to compare to

Returns:
`true` if **entry** is a parent of **child**, `false` otherwise

(continued from last page)

isChild

```
public boolean isChild(Object entry,
         Object parent)
```

Child check. (Child-parent relation.)

Parameters:

`entry` - a context node to compare to
`parent` - a test node to compare to

Returns:

`true` if **entry** is a child of **parent**, `false` otherwise

---

isAncestor

```
public boolean isAncestor(Object entry,
         Object descendant)
```

Ancestry check. (A transitive closure of `isParent` relation.)

Parameters:

`entry` - a context node to compare to
`descendant` - a test node to compare to

Returns:

`true` if **entry** is an ancestor of **descendant**, `false` otherwise

---

isDescendant

```
public boolean isDescendant(Object entry,
         Object ancestor)
```

Descendancy check. (A transitive closure of `isChild` relation.)

Parameters:

`entry` - a context node to compare to
`ancestor` - a test node to compare to

Returns:

`true` if **entry** is an descendant of **ancestor**, `false` otherwise

---

getLCA

```
public Object getLCA(Object[] entries)
```

Least Common Ancestor (LCA).
Remarks: In cases the entries are not actually related, it returns the *root* node of the taxonomy.

Parameters:

`entries` - an array of nodes

Returns:

returns a node which is the least common ancestor of all nodes on the **entries** list

---

getAncestorIterator

```
public Iterator getAncestorIterator(Object entry,
         Comparator cmp)
```

(continued from last page)

Iterate over the ancestor hierarchy bottom-up in a well-defined (deterministic) order. This method can be used to implement hierarchical *defaulting* (constraint relaxing). To implement this feature we apply two order relations on the set of taxonomy values:

1. the partial order implied by the taxonomy hierarchy (the *primary order*)
2. the total order defined by the additional `Comparator` (the *secondary order*)

The secondary order applies only in cases two nodes are not comparable to each other using the primary order, this happens when a node has more than one parent. This method allows to traverse the taxonomy hierarchy in a well-defined (deterministic) way.
Remarks: the **entry** node is not included.

Parameters:

`entry` - a context node to start from
`cmp` - an comparator to define the secondary order.

Returns:

bottom-up taxonomy `Iterator`

See Also:

`RelationalOperatorsRegistry.getTotalOrder(String)`
`OrderProperty.comparator()`

cz.cuni.versatile.api
Interface TemplateBasedQuery

All Superinterfaces:
Query

---

public interface **TemplateBasedQuery**
extends Query

`TemplateBasedQuery` represents a query created as a spin-off of a `QueryTemplate` by invoking its `newQuery()` method.
See Also:
`QueryTemplate`

Fields inherited from interface `cz.cuni.versatile.api.Query`
`BIASED_SCORING_FACTOR`, `DEFAULT_MATCH_SCORE`, `DEFAULT_N_BEST`, `DEFAULT_SCORING_FACTOR`, `EXACT_MATCH_SCORE`, `NEUTRAL_SCORING_FACTOR`

Method Summary

`QueryTemplate`	`getQueryTemplate()` Returns the `QueryTemplate` this query based on.

Methods inherited from interface `cz.cuni.versatile.api.Query`
`getNBest`, `getPredicates`, `getResourceName`, `getScoreThreshold`, `getScoringFactor`

Methods

getQueryTemplate

`public QueryTemplate getQueryTemplate()`

Returns the `QueryTemplate` this query based on.

Returns:
the `QueryTemplate` this query based on.

cz.cuni.versatile.api
Interface TreeTaxonomy

All Superinterfaces:
Taxonomy, OrderProperty, RelationalProperty, Property

public interface **TreeTaxonomy**
extends Taxonomy

A specialized tree taxonomy. Each node except the root has exactly one parent.

Method Summary

List	getAncestorChain(Object entry) Returns a list ancestors ordered bottom-up starting from the **entry** node.
Iterator	getAncestorIterator(Object entry) Iterate over the ancestor list bottom-up starting from the **entry** node.
Object	getParent(Object entry) Returns the parent of the context node.

Methods inherited from interface cz.cuni.versatile.api.Taxonomy

getAncestorIterator, getAncestors, getChildren, getDescendants, getLCA, getParents, getRoot, isAncestor, isChild, isDescendant, isParent, isRoot

Methods inherited from interface cz.cuni.versatile.api.ControlledVocabulary

getValueSet, iterator

Methods inherited from interface cz.cuni.versatile.api.Property

getLocalName, getNamespace, getSeparator, getType, getUniqueName

Methods inherited from interface cz.cuni.versatile.api.OrderProperty

comparable, comparator, isPartialOrder, isStrictOrder, isTotalOrder

Methods inherited from interface cz.cuni.versatile.api.RelationalProperty

contains, isAntisymmetric, isAsymetric, isIrreflexive, isReflexive, isSymmetric, isTransitive

Methods inherited from interface cz.cuni.versatile.api.Property

getLocalName, getNamespace, getSeparator, getType, getUniqueName

Methods

(continued from last page)

getParent

`public Object **getParent**(Object entry)`

Returns the parent of the context node.

Parameters:
`entry` - a context node to compare to

Returns:
the parent of the **entry** node, `null` in case of the root node.

---

getAncestorChain

`public List **getAncestorChain**(Object entry)`

Returns a list ancestors ordered bottom-up starting from the **entry** node.
Remarks: the **entry** node is not included.

Parameters:
`entry` - a context node to start from

Returns:
an ancestor `List` ordered bottom-up starting with the **entry** node.

---

getAncestorIterator

`public Iterator **getAncestorIterator**(Object entry)`

Iterate over the ancestor list bottom-up starting from the **entry** node.
Remarks: the **entry** node is not included.

Parameters:
`entry` - a context node to start from

Returns:
`Iterator` returning elements in bottom-up order

cz.cuni.versatile.api
Class UndefinedPropertyValueException

```
java.lang.Object
   ┗-java.lang.Throwable
      ┗-java.lang.Exception
         ┗-cz.cuni.versatile.api.RegistryException
            ┗-cz.cuni.versatile.api.UndefinedPropertyValueException
```

All Implemented Interfaces:
Serializable

---

public class **UndefinedPropertyValueException**
extends RegistryException

Thrown when the invocation of `DeliveryContext.getValue(Property)` or `DeliveryContext.getValue(String)` fails. This typically indicates a severe runtime failure or a bad design of the `PropertyMappings` hierarchy and/or the `ValueProvider` chains: in a well-designed application:

- every *leaf* property should implement a robust fall-back policy in its `ValueProvider` chain (up-to a built-in constant level) to ensure the value is alwas defined
- every `PropertyMapping` should be a *total* function in order to ensure its value is defined across its entire *domain*.

Constructor Summary

public	UndefinedPropertyValueException()
public	UndefinedPropertyValueException(String message)
public	UndefinedPropertyValueException(Throwable cause)
public	UndefinedPropertyValueException(String message, Throwable cause)

Methods inherited from class `java.lang.Throwable`

`fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString`

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

UndefinedPropertyValueException

`public` **`UndefinedPropertyValueException`**`()`

(continued from last page)

UndefinedPropertyValueException

```
public UndefinedPropertyValueException(String message)
```

UndefinedPropertyValueException

```
public UndefinedPropertyValueException(Throwable cause)
```

UndefinedPropertyValueException

```
public UndefinedPropertyValueException(String message,
                                       Throwable cause)
```

cz.cuni.versatile.api
Class UnregisteredPropertyException

```
java.lang.Object
   |
   +-java.lang.Throwable
      |
      +-java.lang.Exception
         |
         +-cz.cuni.versatile.api.RegistryException
            |
            +-cz.cuni.versatile.api.UnregisteredPropertyException
```

All Implemented Interfaces:
Serializable

---

public class **UnregisteredPropertyException**
extends RegistryException

Thrown when the given `Property` is not registered in the target `PropertyRegistry`.
See Also:
`PropertyRegistry`

Constructor Summary

public	UnregisteredPropertyException()
public	UnregisteredPropertyException(String message)
public	UnregisteredPropertyException(Throwable cause)
public	UnregisteredPropertyException(String message, Throwable cause)

Methods inherited from class `java.lang.Throwable`

`fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString`

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

UnregisteredPropertyException

`public` **`UnregisteredPropertyException`**`()`

(continued from last page)

UnregisteredPropertyException

```
public UnregisteredPropertyException(String message)
```

UnregisteredPropertyException

```
public UnregisteredPropertyException(Throwable cause)
```

UnregisteredPropertyException

```
public UnregisteredPropertyException(String message,
                                     Throwable cause)
```

cz.cuni.versatile.api
Class UnsupportedPropertyOperatorException

```
java.lang.Object
   |-java.lang.Throwable
      |-java.lang.Exception
         |-cz.cuni.versatile.api.UnsupportedPropertyOperatorException
```

All Implemented Interfaces:
Serializable

---

public class **UnsupportedPropertyOperatorException**
extends Exception

Thrown when trying to instantiate a `PropertyPredicate` with a type-incompatible pair of `Property` and an extrinsic `PropertyOperator`.
See Also:
`PropertyOperators`

Constructor Summary

public	`UnsupportedPropertyOperatorException()`
public	`UnsupportedPropertyOperatorException(String message)`
public	`UnsupportedPropertyOperatorException(Throwable cause)`
public	`UnsupportedPropertyOperatorException(String message, Throwable cause)`

Methods inherited from class `java.lang.Throwable`

`fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString`

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

UnsupportedPropertyOperatorException

`public` **`UnsupportedPropertyOperatorException`**`()`

(continued from last page)

UnsupportedPropertyOperatorException

```
public UnsupportedPropertyOperatorException(String message)
```

UnsupportedPropertyOperatorException

```
public UnsupportedPropertyOperatorException(Throwable cause)
```

UnsupportedPropertyOperatorException

```
public UnsupportedPropertyOperatorException(String message,
                                            Throwable cause)
```

cz.cuni.versatile.api
Interface ValueProvider

All Known Implementing Classes:
AbstractValueProvider

---

public interface **ValueProvider**
extends

`ValueProvider` interface represents a concept an attribute value getter, which can be chained in order to implement a particular fall-back strategy.

An implementation of the `ValueProvider` is specific to a particular value source, e.g. a flat-file, an XML file, relational database, `HttpServletRequest`, `HttpSession`, CC/PP profile and other. When instantiating a `ValueProvider`, the user must specify a *unique name* of an attribute and optionally can also provide a *default* -- another instance of the `ValueProvider` interface which is used as a fall-back in the case the attribute value is not set in the former `ValueProvider` instance. This defaulting principle can be applied recursively thus forming a chain of `ValueProvider` instances to implement a particular fall-back strategy: for example, to retrieve a value from command line parameters, if not set, than from a configuration file and eventually to fall-back to a built-in constant.

It is important to note that attribute name can differ for each `ValueProvider` in the chain as the same semantical entity can have different names in different contexts.

Method Summary

`String`	`getAttributeName()` Returns attribute name.
`ValueProvider`	`getDefault()` Returns the next `ValueProvider` in the chain.
`Object`	`getLocalValue()` Returns attribute value.
`Object`	`getValue()` Returns attribute value.
`boolean`	`hasLocalValue()` Checks whether the attribute is currently set.
`boolean`	`hasValue()` Checks whether the attribute is currently set.
`PreferenceChain`	`toPreferenceChain()` Takes the current snapshot of the `ValueProvider` chain returning a list of values in the `ValueProvider` chain.

Methods

getAttributeName

`public String **getAttributeName**()`

Returns attribute name.

(continued from last page)

Returns:
a unique name of the attribute this `ValueProvider` is bound to.

getValue

`public Object getValue()`

Returns attribute value. (Recursive for the entire chain of `ValueProvider` instances.)

Returns:
the current value of the attribute at the time of invocation

hasValue

`public boolean hasValue()`

Checks whether the attribute is currently set. (Recursive for the entire chain of `ValueProvider` instances.)

Returns:
`true` if the value is set for any `ValueProvider` in the chain, `false` otherwise

getLocalValue

`public Object getLocalValue()`

Returns attribute value.

Returns:
the current value of the attribute at the time of invocation

hasLocalValue

`public boolean hasLocalValue()`

Checks whether the attribute is currently set.

Returns:
`true` if the value is set for this `ValueProvider` instance, `false` otherwise

getDefault

`public ValueProvider getDefault()`

Returns the next `ValueProvider` in the chain.

Returns:
the next `ValueProvider` in the chain or `null` when invoked for the tail of the chain.

toPreferenceChain

`public PreferenceChain toPreferenceChain()`

Takes the current snapshot of the `ValueProvider` chain returning a list of values in the `ValueProvider` chain. A value is appended to the chain for those value providers, where `hasLocaleValue()` returns `true`.

Remarks: Any item in the result `PreferenceChain` can be the desired property value as well as a `PreferenceChain` or a `PreferenceBag` of property values as returned by an individual `ValueProvider` in the chain. The purpose of this method is mainly for diagnostics (tuning and debugging).

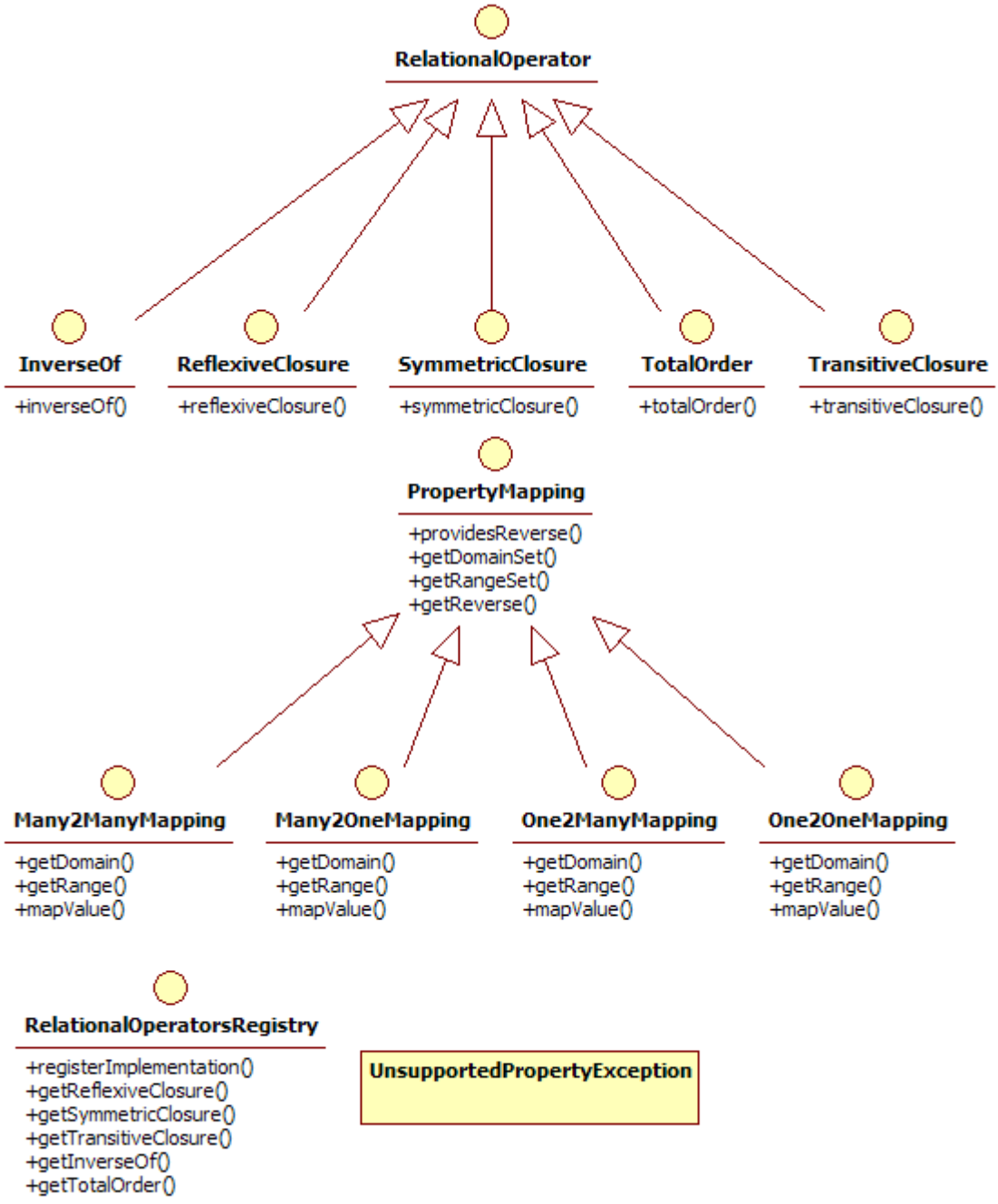
(continued from last page)

Returns:

`PreferenceChain` object - a list of values - can be an empty list.

Package
cz.cuni.versatile.api.relops

Relational operators and property mappings subsection of the Versatile API.



cz.cuni.versatile.api.relops
Interface InverseOf

All Superinterfaces:
RelationalOperator

public interface **InverseOf**
extends RelationalOperator

`InverseOf` implementations inverse `RelationalProperty` properties thus constructing an inverse binary relation for a given relation represented by a `RelationalProperty`. By inverse binary relation we understand a relation, where *domain* and *range* are swapped and
`for all x, y: P.contains(x, y) = true <-> InverseOf(P).contains(y, x) = true`

Note: A generic implementation of this interface is feasible which constructs inverse properties on the fly.

Method Summary

`RelationalProperty`	`inverseOf(RelationalProperty source)` Returns a `RelationalProperty` which corresponds to an inverse binary relation to the source `RelationalProperty`.

Methods

inverseOf

`public RelationalProperty inverseOf(RelationalProperty source)`

Returns a `RelationalProperty` which corresponds to an inverse binary relation to the source `RelationalProperty`.

Parameters:
`source` - a source relational property

cz.cuni.versatile.api.relops
Interface Many2ManyMapping

All Superinterfaces:
PropertyMapping

---

public interface **Many2ManyMapping**
extends PropertyMapping

The most generic property mapping as described in `PropertyMapping` API. It implements `n:m` arity map - a complex transformation between two sets of properties.

Method Summary

`Property[]`	`getDomain()` Returns an ordered list of properties which form a domain of this mapping.
`Property[]`	`getRange()` Returns an ordered list of properties which form a range of this mapping.
`Object[]`	`mapValue(Object[] dom)` Executes the actual transformation - a mapping.

Methods inherited from interface `cz.cuni.versatile.api.relops.PropertyMapping`

`getDomainSet`, `getRangeSet`, `getReverse`, `providesReverse`

Methods

getDomain

`public Property[] getDomain()`

Returns an ordered list of properties which form a domain of this mapping.

Returns:
an ordered list of domain properties

See Also:
`PropertyMapping.getDomainSet()`

---

getRange

`public Property[] getRange()`

Returns an ordered list of properties which form a range of this mapping.

Returns:
an ordered list of range properties

(continued from last page)

mapValue

`public Object[] mapValue(Object[] dom)`

Executes the actual transformation - a mapping.

Parameters:

`dom` - an ordered list of domain properties values (the order must correspond to the `getDomain()` method)

Returns:

an ordered list of range properties values (the order must corresponds to the `getRange()` method)

cz.cuni.versatile.api.relops
Interface Many2OneMapping

All Superinterfaces:
PropertyMapping

---

public interface **Many2OneMapping**
extends PropertyMapping

A mapping which corresponds to an ordinary n-ary function `(P1, P2, P3, ..., Pn) -> Pm`.

It is used if there is a need to combine several simpler properties into a single property and then use this target property in the versioning code.

UAProf example: use `BrowserUA.* (HtmlVersion, XhtmlVersion, BrowserVersion, BrowserName)` and `WapCharacteristics.* (WapVersion, WmlVersion, WmlScriptVersion)` to derive a single `DeviceMarkupClass`, a hierarchical classification (taxonomy), to simplify the decisions based on a device markup capabilities.

Method Summary

`Property[]`	`getDomain()` Returns an ordered list of properties which form a domain of this mapping.
`Property`	`getRange()` Returns a range of this mapping.
`Object`	`mapValue(Object[] dom)` Executes the actual transformation - a mapping.

Methods inherited from interface `cz.cuni.versatile.api.relops.PropertyMapping`

`getDomainSet`, `getRangeSet`, `getReverse`, `providesReverse`

Methods

getDomain

`public Property[] getDomain()`

Returns an ordered list of properties which form a domain of this mapping.

Returns:
an ordered list of domain properties

See Also:
`PropertyMapping.getDomainSet()`

---

getRange

`public Property getRange()`

Returns a range of this mapping.

(continued from last page)

Returns:
a range property

mapValue

```
public Object mapValue(Object[] dom)
```

Executes the actual transformation - a mapping.

Parameters:
`dom` - an ordered list of domain properties values (the order must correspond to the `getDomain()` method)

Returns:
a range property value

cz.cuni.versatile.api.relops
Interface One2ManyMapping

All Superinterfaces:
PropertyMapping

---

public interface **One2ManyMapping**
extends PropertyMapping

An information extraction mapping `Px -> (P1, P2, P3, ..., Pn)`. There are quite a few instances of meta-data properties which contain *composite literal* values or combine multiple semantical entities into a single named property and in turn require further parsing to extract the individual sub-properties. For example:

- HTTP `User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.0.4`
- UAProf `SoftwarePlaform/JavaPlatform` can contain both `Configuration/CLDC-1.0` versus `Profile/MIDP-2.0` thus combining *Profile* and *Configuration* under one UAProf attribute.

The motivation for introducing this type of mapping is to be able to parse the source property value and extract all the output data values one pass to improve performance when using this within a `DeliveryContext`. Semantically, this type of mapping can be replaced by a set of unary mappings.

Method Summary

Property	getDomain() Returns a domain of this mapping.
Property[]	getRange() Returns an ordered list of properties which form a range of this mapping.
Object[]	mapValue(Object dom) Executes the actual transformation - a mapping.

Methods inherited from interface cz.cuni.versatile.api.relops.PropertyMapping

getDomainSet, getRangeSet, getReverse, providesReverse

Methods

getDomain

`public Property getDomain()`

Returns a domain of this mapping.

Returns:
a domain property

---

getRange

`public Property[] getRange()`

(continued from last page)

Returns an ordered list of properties which form a range of this mapping.

Returns:
an ordered list of range properties

---

mapValue

```
public Object[] mapValue(Object dom)
```

Executes the actual transformation - a mapping.

Parameters:
`dom` - a domain property value

Returns:
an ordered list of range properties values (the order must corresponds to the `getRange()` method)

cz.cuni.versatile.api.relops
Interface One2OneMapping

All Superinterfaces:
PropertyMapping

All Known Implementing Classes:
IdentityMapping

---

public interface **One2OneMapping**
extends PropertyMapping

An unary mapping `(P1) -> (P2)` to transform values of one property to values of another property. Can be used for canonicalization or/and to generate taxonomies (hierarchical classifications) out of the raw unchecked meta data values.

Method Summary

`Property`	`getDomain()` Returns a domain of this mapping.
`Property`	`getRange()` Returns a range of this mapping.
`Object`	`mapValue(Object dom)` Executes the actual transformation - a mapping.

Methods inherited from interface `cz.cuni.versatile.api.relops.PropertyMapping`

`getDomainSet`, `getRangeSet`, `getReverse`, `providesReverse`

Methods

getDomain

`public Property getDomain()`

Returns a domain of this mapping.

Returns:
a domain property

---

getRange

`public Property getRange()`

Returns a range of this mapping.

Returns:
a range property

(continued from last page)

mapValue

```
public Object mapValue(Object dom)
```

Executes the actual transformation - a mapping.

Parameters:

`dom` - a domain property value

Returns:

a range property value

cz.cuni.versatile.api.relops
Interface PropertyMapping

All Subinterfaces:
Many2ManyMapping, Many2OneMapping, One2ManyMapping, One2OneMapping

---

public interface **PropertyMapping**
extends

`PropertyMapping` interface represents a generic a transformational map from a set of properties to another set of another properties. Mathematically speaking, it is an n-ary function `(P1, P2, P3, ..., Pn) -> (P1', P2', P3', ...Pm)`, `n` and `m` being positive integers, so that given input values `(x1:P1, ..., xn:Pn)` it calculates a tuple of output values `(y1:P1', ..., ym:Pm)`.

This generic interface is a super-type for a set of specialized mapping interfaces with different arities. The purpose of this interface to serve as a marker interface and for meta-modeling capabilities like property dependency tracking. One of its sub-types `Many2ManyMapping` corresponds to the most generic `n:m` mapping, other are simpler `1:n`, `n:1` and `1:1` arity maps.

The reason behind the chosen type hierarchy organization is to make implementation of the most common mappings easier, by most common we assume `1:1` - unary function.

Method Summary

`Set`	`getDomainSet()` Domain: a set of all source properties required to calculate a value of this mapping.
`Set`	`getRangeSet()` Range: Returns a set of all target properties calculated by this mapping.
`PropertyMapping`	`getReverse()` Returns an implementation of the corresponding reverse mapping, if implemented.
`boolean`	`providesReverse()` Check whether this property mapping is aware of its corresponding reverse transformation map.

Methods

providesReverse

`public boolean **providesReverse**()`

Check whether this property mapping is aware of its corresponding reverse transformation map.

Returns:
`true` if an invocation to `getReverse()` will return a reverse mapping.

---

getDomainSet

`public Set **getDomainSet**()`

Domain: a set of all source properties required to calculate a value of this mapping. The size of the set is the arity of the mapping.

(continued from last page)

Returns:
a set of all source properties, must contain at least one item

getRangeSet

```
public Set getRangeSet()
```

Range: Returns a set of all target properties calculated by this mapping.

Returns:
a set of all target properties, must contain at least one item

getReverse

```
public PropertyMapping getReverse()
```

Returns an implementation of the corresponding reverse mapping, if implemented. A reverse mapping has their domain and range sets swapped and in addition to that it should semantically complement its counterpart. If this methods does return `null`, it does not mean a reverse mapping does not exist: there can still be one or more independent implementations.

Returns:
reverse mapping if exists and known to this mapping implementation

cz.cuni.versatile.api.relops
Interface ReflexiveClosure

All Superinterfaces:
RelationalOperator

public interface **ReflexiveClosure**
extends RelationalOperator

`ReflexiveClosure` operator turns an input binary relation represented by a `RelationalProperty` into a reflexive binary relation. In case the source property is already reflexive, the operator should return the source property.

Note: A generic implementation of this interface is feasible which constructs reflexive closures on the fly.
See Also:
`RelationalProperty.isReflexive()`

Method Summary

`RelationalProperty`	`reflexiveClosure(RelationalProperty source)` Returns a `RelationalProperty` which corresponds to a reflexive closure of the source binary relation.

Methods

reflexiveClosure

`public RelationalProperty reflexiveClosure(RelationalProperty source)`

Returns a `RelationalProperty` which corresponds to a reflexive closure of the source binary relation.

Parameters:
`source` - a source relational property

cz.cuni.versatile.api.relops
Interface RelationalOperator

All Subinterfaces:
InverseOf, ReflexiveClosure, SymmetricClosure, TotalOrder, TransitiveClosure

public interface **RelationalOperator**
extends

cz.cuni.versatile.api.relops
Interface RelationalOperatorsRegistry

public interface **RelationalOperatorsRegistry**
extends

`RelationalOperatorsRegistry` is a generic class factory allowing to register custom implementations of relational operators for individual *relational* properties.
See Also:
`RelationalProperty`

Method Summary

`InverseOf`	`getInverseOf(String uniqueName)` Returns a `InverseOf` implementation for a given relational property identified by its `uniqueName`.
`ReflexiveClosure`	`getReflexiveClosure(String uniqueName)` Returns a `ReflexiveClosure` implementation for a given relational property identified by its `uniqueName`.
`SymmetricClosure`	`getSymmetricClosure(String uniqueName)` Returns a `SymmetricClosure` implementation for a given relational property identified by its `uniqueName`.
`TotalOrder`	`getTotalOrder(String uniqueName)` Returns a `TotalOrder` implementation for a given relational property identified by its `uniqueName`.
`TransitiveClosure`	`getTransitiveClosure(String uniqueName)` Returns a `TransitiveClosure` implementation for a given relational property identified by its `uniqueName`.
`void`	`registerImplementation(String uniqueName, RelationalOperator implementation)` Register an implementation for one or more relational operators for a given property.

Methods

registerImplementation

```
public void registerImplementation(String uniqueName,
        RelationalOperator implementation)
```

Register an implementation for one or more relational operators for a given property. This method uses `uniqueName` instead of an instance of `RelationalProperty` in order to avoid the need to physically instantiate all properties for which we want to configure relational operators with the registry. In the typical situation, there are generic (property independent) implementations for certain relational operators (`InversOf`, `ReflexiveClosure`, `SymmetricClosure` and also `TotalOrder`), these can be overridden by custom implementations for individual properties if needed.

Parameters:

`uniqueName` - a fully qualified name of a relational property

`implementation` - an implementation of one or more relational operators, Java reflection is used to determine the set of relational operators the given implementation supports.

getReflexiveClosure

```
public ReflexiveClosure getReflexiveClosure(String uniqueName)
  throws UnsupportedPropertyException
```

Returns a `ReflexiveClosure` implementation for a given relational property identified by its `uniqueName`.

Parameters:
`uniqueName` - a fully qualified name of a relational property

Returns:
an implementation of a relational operator if one exists for a given property

Throws:
`UnsupportedPropertyException` - in case there is no implementation of the operator for a given property

getSymmetricClosure

```
public SymmetricClosure getSymmetricClosure(String uniqueName)
  throws UnsupportedPropertyException
```

Returns a `SymmetricClosure` implementation for a given relational property identified by its `uniqueName`.

Parameters:
`uniqueName` - a fully qualified name of a relational property

Returns:
an implementation of a relational operator if one exists for a given property

Throws:
`UnsupportedPropertyException` - in case there is no implementation of the operator for a given property

getTransitiveClosure

```
public TransitiveClosure getTransitiveClosure(String uniqueName)
  throws UnsupportedPropertyException
```

Returns a `TransitiveClosure` implementation for a given relational property identified by its `uniqueName`.

Parameters:
`uniqueName` - a fully qualified name of a relational property

Returns:
an implementation of a relational operator if one exists for a given property

Throws:
`UnsupportedPropertyException` - in case there is no implementation of the operator for a given property

getInverseOf

```
public InverseOf getInverseOf(String uniqueName)
  throws UnsupportedPropertyException
```

Returns a `InverseOf` implementation for a given relational property identified by its `uniqueName`.

Parameters:
`uniqueName` - a fully qualified name of a relational property

(continued from last page)

Returns:
an implementation of a relational operator if one exists for a given property

Throws:
`UnsupportedPropertyException` - in case there is no implementation of the operator for a given property

---

getTotalOrder

```
public TotalOrder getTotalOrder(String uniqueName)
  throws UnsupportedPropertyException
```

Returns a `TotalOrder` implementation for a given relational property identified by its `uniqueName`.

Parameters:
`uniqueName` - a fully qualified name of a relational property

Returns:
an implementation of a relational operator if one exists for a given property

Throws:
`UnsupportedPropertyException` - in case there is no implementation of the operator for a given property

cz.cuni.versatile.api.relops
Interface SymmetricClosure

All Superinterfaces:
RelationalOperator

---

public interface **SymmetricClosure**
extends RelationalOperator

`SymmetricClosure` operator turns a source binary relation represented by a `RelationalProperty` into a symmetric binary relation. In case the source property is already symmetric, the operator should return the source property.

Note: A generic implementation of this interface is feasible which constructs symmetric closures on the fly.
See Also:
`RelationalProperty.isSymmetric()`

Method Summary

`RelationalProperty`	`symmetricClosure(RelationalProperty source)` Returns a `RelationalProperty` which corresponds to a symmetric closure of the source binary relation.

Methods

symmetricClosure

`public RelationalProperty symmetricClosure(RelationalProperty source)`

Returns a `RelationalProperty` which corresponds to a symmetric closure of the source binary relation.

Parameters:
`source` - a source relational property

cz.cuni.versatile.api.relops
Interface TotalOrder

All Superinterfaces:
RelationalOperator

---

public interface **TotalOrder**
extends RelationalOperator

`TotalOrder` operator constructs a total order as an *extension* of a given source `OrderProperty`. By *extension* we mean that the resulting total order must be a super-set of the source (partial) order thus preserving the existing source order. In case the source order property is already a total order, the operator should return the source property.

Note: A generic implementation of this interface producing a meaningful total order for any given source property is hardly possible. We assume property-specific implementations of this interface are used in most circumstances, however, a default generic implementation can be provided to ensure a deterministic total order exists for all properties , even though there is no particular semantics associated to it.

See Also:
`OrderProperty.isTotalOrder()`

Method Summary

`OrderProperty`	`totalOrder(OrderProperty source)` Returns an `OrderProperty` which a total order extending the source `OrderProperty`.

Methods

totalOrder

`public OrderProperty totalOrder(OrderProperty source)`

Returns an `OrderProperty` which a total order extending the source `OrderProperty`.

Parameters:
`source` - a source order property

cz.cuni.versatile.api.relops
Interface TransitiveClosure

All Superinterfaces:
RelationalOperator

---

public interface **TransitiveClosure**
extends RelationalOperator

`TransitiveClosure` operator builds a transitive closure of a source `RelationalProperty` so that the resulting `RelationalProperty` represents a *transitive* binary relation. In case the source property is already transitive, the operator should return the source property.

Note: A generic implementation of this interface is practically impossible given the way the Versatile API is designed: `RelationalProperty` only has `contains(x, y)` method, so that one would first need to scan the entire domain and build a list of all ordered pairs `{x, y}` where `contains(x, y) = true` and then proceed with building the transitive closure. This represents computational complexity O(N2). It would help if we add an additional method to the `RelationalProperty`, which for a given `x` would return a set of all y such that `contains(x, y) = true`. The problem is that having such a method requires all relational properties to become *enumerable*, which is exactly what we want to avoid: the Versatile API is designed in the way which allows and encourages computational approach to implementing properties (as opposed to pure algebraic data manipulation which can be quite limiting for some applications). Given the above, we assume property-specific implementations of this interface are most likely to be used.

See Also:
`RelationalProperty.isTransitive()`

Method Summary

`RelationalProperty`	`transitiveClosure(RelationalProperty source)` Returns a `RelationalProperty` which corresponds to a transitive closure of the source binary relation.

Methods

transitiveClosure

`public RelationalProperty transitiveClosure(RelationalProperty source)`

Returns a `RelationalProperty` which corresponds to a transitive closure of the source binary relation.

Parameters:
`source` - a source relational property

cz.cuni.versatile.api.relops
Class UnsupportedPropertyException

```
java.lang.Object
   |-java.lang.Throwable
      |-java.lang.Exception
         |-cz.cuni.versatile.api.relops.UnsupportedPropertyException
```

All Implemented Interfaces:
Serializable

---

public class **UnsupportedPropertyException**
extends Exception

Thrown when unsuccessfully trying to resolve a *uniqueName* to a particular `RelationalOperator`
See Also:
`RelationalOperatorsRegistry`

Constructor Summary

public	UnsupportedPropertyException()
public	UnsupportedPropertyException(String message)
public	UnsupportedPropertyException(Throwable cause)
public	UnsupportedPropertyException(String message, Throwable cause)

Methods inherited from class `java.lang.Throwable`

`fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString`

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructors

UnsupportedPropertyException

`public` **`UnsupportedPropertyException`**`()`

UnsupportedPropertyException

`public` **`UnsupportedPropertyException`**`(String message)`

(continued from last page)

UnsupportedPropertyException

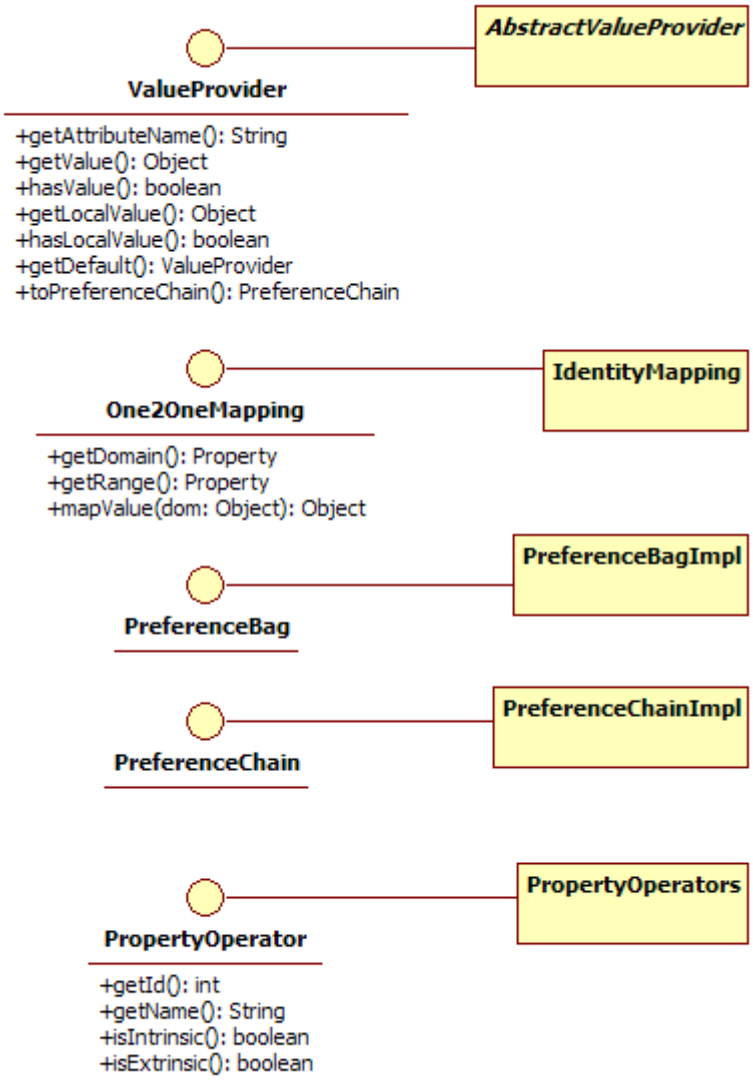
```
public UnsupportedPropertyException(Throwable cause)
```

UnsupportedPropertyException

```
public UnsupportedPropertyException(String message,
                                    Throwable cause)
```

Package
cz.cuni.versatile.core

This package contains the default implementations of some essential Versatile concepts.



cz.cuni.versatile.core
Class AbstractValueProvider

```
java.lang.Object
   |
   +-cz.cuni.versatile.core.AbstractValueProvider
```

All Implemented Interfaces:
ValueProvider

Direct Known Subclasses:
ConstantVP

---

public abstract class **AbstractValueProvider**
extends Object
implements ValueProvider

The default implementation of the `ValueProvider` interface. It implements the chaining mechanism and all the methods which are generic, independent of a particular attributes/values context. When extending `AbstractValueProvider` only the following three methods need be implemented:

1. the two-parameter constructor
2. `getLocalValue()` method
3. `hasLocalValue()` method

Field Summary

protected	attrName Attribute name.
protected	defaultProvider Next-in-the-chain value provider (default).

Constructor Summary

public	AbstractValueProvider(String attrName, ValueProvider defaultValue) Public Constructor

Method Summary

String	getAttributeName()
ValueProvider	getDefault()
Object	getValue()
boolean	hasValue()
PreferenceChain	toPreferenceChain()

Methods inherited from class `java.lang.Object`

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods inherited from interface cz.cuni.versatile.api.ValueProvider
getAttributeName, getDefault, getLocalValue, getValue, hasLocalValue, hasValue, toPreferenceChain

Fields

attrName

`protected java.lang.String attrName`

Attribute name.

See Also:
getAttributeName()

defaultProvider

`protected cz.cuni.versatile.api.ValueProvider defaultProvider`

Next-in-the-chain value provider (default).

See Also:
getDefault()

Constructors

AbstractValueProvider

```
public AbstractValueProvider(String attrName,
                             ValueProvider defaultValue)
```

Public Constructor

Parameters:
`attrName` - an attribute name in a given attributes/values context
`defaultValue` - a default value provider (fall-back), can be `null`

Methods

getAttributeName

`public final String getAttributeName()`

See Also:
ValueProvider.getAttributeName()

getValue

`public final Object getValue()`

(continued from last page)

See Also:
ValueProvider.getValue()

hasValue

```
public final boolean hasValue()
```

See Also:
ValueProvider.hasValue()

getDefault

```
public final ValueProvider getDefault()
```

See Also:
ValueProvider.getDefault()

toPreferenceChain

```
public PreferenceChain toPreferenceChain()
```

See Also:
ValueProvider.toPreferenceChain()

cz.cuni.versatile.core
Class ConstantVP

```
java.lang.Object
   |-cz.cuni.versatile.core.AbstractValueProvider
      |-cz.cuni.versatile.core.ConstantVP
```

All Implemented Interfaces:
ValueProvider

---

public final class **ConstantVP**
extends AbstractValueProvider

ConstantVP stands for *Constant Value Provider*. The last-resort value provider, usually attached to the end of a value provider chain to ensure a property value is always defined.

Fields inherited from class `cz.cuni.versatile.core.AbstractValueProvider`
`attrName`, `defaultProvider`

Constructor Summary

`public`	`ConstantVP(Object value)`

Method Summary

`Object`	`getLocalValue()` Returns the constant value provided during instantiation.
`boolean`	`hasLocalValue()` Always returns `true`.

Methods inherited from class `cz.cuni.versatile.core.AbstractValueProvider`
`getAttributeName`, `getDefault`, `getValue`, `hasValue`, `toPreferenceChain`

Methods inherited from class `java.lang.Object`
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods inherited from interface `cz.cuni.versatile.api.ValueProvider`
`getAttributeName`, `getDefault`, `getLocalValue`, `getValue`, `hasLocalValue`, `hasValue`, `toPreferenceChain`

Constructors

(continued from last page)

ConstantVP

`public` **`ConstantVP`**`(Object value)`

Methods

getLocalValue

`public Object` **`getLocalValue`**`()`

Returns the constant value provided during instantiation.

See Also:
`ValueProvider.getLocalValue()`

---

hasLocalValue

`public boolean` **`hasLocalValue`**`()`

Always returns `true`.

See Also:
`ValueProvider.hasLocalValue()`

cz.cuni.versatile.core
Class IdentityMapping

```
java.lang.Object
   |
   +-cz.cuni.versatile.core.IdentityMapping
```

All Implemented Interfaces:
One2OneMapping

---

public final class **IdentityMapping**
extends Object
implements One2OneMapping

A generic implementation of an identity mapping.

This is essentially used for property renaming to ensure compatibility between multiple property namespaces. For example when using multiple revisions of UAProf or other vocabulary, where each revision uses a different XML namespace as a prefix for their attributes, even though some of the attributes definitions remain unchanged.

Constructor Summary

public	IdentityMapping(Property domain, Property range) Constructs a mapping instance for a given pair of properties.

Method Summary

Property	getDomain()
Set	getDomainSet()
Property	getRange()
Set	getRangeSet()
PropertyMapping	getReverse()
Object	mapValue(Object dom) Implements an identity function, i.e., returns its input as the output.
boolean	providesReverse()

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods inherited from interface `cz.cuni.versatile.api.relops.One2OneMapping`

`getDomain, getRange, mapValue`

Methods inherited from interface `cz.cuni.versatile.api.relops.PropertyMapping`

getDomainSet, getRangeSet, getReverse, providesReverse

Constructors

IdentityMapping

```
public IdentityMapping(Property domain,
                       Property range)
```

Constructs a mapping instance for a given pair of properties.

Methods

getDomain

```
public Property getDomain()
```

See Also:
One2OneMapping.getDomain()

getRange

```
public Property getRange()
```

See Also:
One2OneMapping.getRange()

mapValue

```
public Object mapValue(Object dom)
```

Implements an identity function, i.e., returns its input as the output.

See Also:
One2OneMapping.mapValue(Object)

providesReverse

```
public boolean providesReverse()
```

See Also:
PropertyMapping.providesReverse()

getDomainSet

```
public Set getDomainSet()
```

(continued from last page)

See Also:
PropertyMapping.getDomainSet()

---

getRangeSet

```
public Set getRangeSet()
```

See Also:
PropertyMapping.getRangeSet()

---

getReverse

```
public PropertyMapping getReverse()
```

See Also:
PropertyMapping.getReverse()

cz.cuni.versatile.core
Class PreferenceBagImpl

```
java.lang.Object
   |
   +-java.util.AbstractCollection
      |
      +-java.util.AbstractSet
         |
         +-java.util.HashSet
            |
            +-cz.cuni.versatile.core.PreferenceBagImpl
```

All Implemented Interfaces:
PreferenceBag, Collection, Set, Serializable, Cloneable, Set

---

public class **PreferenceBagImpl**
extends HashSet
implements Set, Cloneable, Serializable, Set, Collection, PreferenceBag

Default implementation of the `PreferenceBag` interface.

Constructor Summary

public	PreferenceBagImpl()
public	PreferenceBagImpl(int initialCapacity)
public	PreferenceBagImpl(int initialCapacity, float loadFactor)
public	PreferenceBagImpl(Collection c)

Methods inherited from class `java.util.HashSet`

`add, clear, clone, contains, isEmpty, iterator, remove, size`

Methods inherited from class `java.util.AbstractSet`

`equals, hashCode, removeAll`

Methods inherited from class `java.util.AbstractCollection`

`add, addAll, clear, contains, containsAll, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray, toString`

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods inherited from interface `java.util.Collection`

`add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray`

Methods inherited from interface `java.util.Set`
add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray

Methods inherited from interface `java.util.Collection`
add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray

Methods inherited from interface `java.util.Set`
add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray

Methods inherited from interface `java.util.Collection`
add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray

Methods inherited from interface `java.util.Set`
add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray

Methods inherited from interface `java.util.Collection`
add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray

Constructors

PreferenceBagImpl

```
public PreferenceBagImpl()
```

See Also:
`java.util.HashSet()`

PreferenceBagImpl

```
public PreferenceBagImpl(int initialCapacity)
```

See Also:
`java.util.HashSet(int)`

PreferenceBagImpl

```
public PreferenceBagImpl(int initialCapacity,
                         float loadFactor)
```

(continued from last page)

See Also:
`java.util.HashSet(int, float)`

PreferenceBagImpl

```
public PreferenceBagImpl(Collection c)
```

See Also:
`java.util.HashSet(java.util.Collection)`

cz.cuni.versatile.core
Class PreferenceChainImpl

```
java.lang.Object
   ┗-java.util.AbstractCollection
      ┗-java.util.AbstractList
         ┗-java.util.ArrayList
            ┗-cz.cuni.versatile.core.PreferenceChainImpl
```

All Implemented Interfaces:
PreferenceChain, Collection, List, Serializable, Cloneable, RandomAccess, List

---

public class **PreferenceChainImpl**
extends ArrayList
implements List, RandomAccess, Cloneable, Serializable, List, Collection, PreferenceChain

Default implementation of the `PreferenceChain` interface.

Fields inherited from class `java.util.AbstractList`
`modCount`

Constructor Summary

public	`PreferenceChainImpl()`
public	`PreferenceChainImpl(int initialCapacity)`
public	`PreferenceChainImpl(Collection c)`

Methods inherited from class `java.util.ArrayList`
`add, add, addAll, addAll, clear, clone, contains, ensureCapacity, get, indexOf, isEmpty, lastIndexOf, remove, removeRange, set, size, toArray, toArray, trimToSize`

Methods inherited from class `java.util.AbstractList`
`add, add, addAll, clear, equals, get, hashCode, indexOf, iterator, lastIndexOf, listIterator, listIterator, remove, removeRange, set, subList`

Methods inherited from class `java.util.AbstractCollection`
`add, addAll, clear, contains, containsAll, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray, toString`

Methods inherited from class `java.lang.Object`
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods inherited from interface `java.util.Collection`

add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray

Methods inherited from interface `java.util.List`
add, add, addAll, addAll, clear, contains, containsAll, equals, get, hashCode, indexOf, isEmpty, iterator, lastIndexOf, listIterator, listIterator, remove, remove, removeAll, retainAll, set, size, subList, toArray, toArray

Methods inherited from interface `java.util.Collection`
add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray

Methods inherited from interface `java.util.List`
add, add, addAll, addAll, clear, contains, containsAll, equals, get, hashCode, indexOf, isEmpty, iterator, lastIndexOf, listIterator, listIterator, remove, remove, removeAll, retainAll, set, size, subList, toArray, toArray

Methods inherited from interface `java.util.Collection`
add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray

Methods inherited from interface `java.util.List`
add, add, addAll, addAll, clear, contains, containsAll, equals, get, hashCode, indexOf, isEmpty, iterator, lastIndexOf, listIterator, listIterator, remove, remove, removeAll, retainAll, set, size, subList, toArray, toArray

Methods inherited from interface `java.util.Collection`
add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray

Constructors

PreferenceChainImpl

`public` **`PreferenceChainImpl`**`()`

See Also:

`java.util.ArrayList()`

PreferenceChainImpl

`public` **`PreferenceChainImpl`**`(int initialCapacity)`

See Also:

`java.util.ArrayList(int)`

PreferenceChainImpl

```
public PreferenceChainImpl(Collection c)
```

See Also:
java.util.ArrayList(java.util.Collection)

cz.cuni.versatile.core
Class PropertyOperators

```
java.lang.Object
   |
   +-cz.cuni.versatile.core.PropertyOperators
```

All Implemented Interfaces:
PropertyOperator

---

public class **PropertyOperators**
extends Object
implements PropertyOperator

A default implementation of the `PropertyOperator` interface as well as an enumeration of all built-in (pre-defined) property operators. The set of operators can be easily extended, however, it always depends on the individual implementation of the `ResourceProvider` interface which operators it recognizes. The operators listed below are **mandatory** for all compliant `ResourceProvider` implementations.

Operator	Intrinsic	Assertive	Applicable
=	yes	yes	Property
>	yes	yes	Property
<	yes	yes	Property
>=	yes	yes	Property
<=	yes	yes	Property
assert	no	yes	RelationalProperty
assertInv	no	yes	RelationalProperty
assertLevel	no	yes	Taxonomy
equivalent	no	yes	Equivalence
comparable	no	yes	OrderProperty
isParent	no	yes	Taxonomy
isChild	no	yes	Taxonomy
isAncestor	no	yes	Taxonomy
isDescendant	no	yes	Taxonomy
bestMatch	no	no	Taxonomy

See Also:
QueryTemplate

Field Summary

public static final	ASSERT *assert* extrinsic assertive operator.
public static final	ASSERT_INV *assertInv* extrinsic assertive operator.

public static final	ASSERT_LEVEL *assertLevel* extrinsic assertive operator.
protected	assertive Flag whether the operator is *assertive* or whether it allows for an approximate matching (*fall-back, constraint relaxing*).
public static final	BEST_MATCH *bestMatch* extrinsic constraint-relaxing operator.
public static final	COMPARABLE *comparable* extrinsic assertive operator.
public static final	EQ *Equals* intrinsic assertive operator.
public static final	EQUIVALENT *equivalent* extrinsic assertive operator.
public static final	GE *Greater than or equal to* (>=) intrinsic assertive operator.
public static final	GT *Greater-than* (>) intrinsic assertive operator.
protected	id The unique ID of the operator.
protected	intrinsic Flag whether the operator is *intrinsic* or *extrinsic*
public static final	IS_ANCESTOR *isAncestor* extrinsic assertive operator.
public static final	IS_CHILD *isChild* extrinsic assertive operator.
public static final	IS_DESCENDANT *isDescendant* extrinsic assertive operator.
public static final	IS_PARENT *isParent* extrinsic assertive operator.
public static final	LE *Less than or equal to* (<=) intrinsic assertive operator.
public static final	LT *Less-than* (<) intrinsic assertive operator.
protected	name A human-readable name - just for logging and debugging.

Constructor Summary

protected	PropertyOperators(int id, String name, boolean intrinsic, boolean assertive) Protected Constructor - can be re-used by subclasses extending the set of supported operators.

Method Summary

int	getId()
String	getName()
boolean	isAssertive()
boolean	isExtrinsic()
boolean	isIntrinsic()

Methods inherited from class `java.lang.Object`

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods inherited from interface `cz.cuni.versatile.api.PropertyOperator`

`getId, getName, isAssertive, isExtrinsic, isIntrinsic`

Fields

id

`protected int` **`id`**

The unique ID of the operator.

name

`protected java.lang.String` **`name`**

A human-readable name - just for logging and debugging.

intrinsic

`protected boolean` **`intrinsic`**

Flag whether the operator is *intrinsic* or *extrinsic*

- *intrinsic* operators use methods of the actual property values for comparison, so they rely on the existing methods of Java objects
- *extrinsic* operators rely on the relations externally provided by the properties (e.g. `OrderProperty`, `Equivalence`)

assertive

`protected boolean` **`assertive`**

Flag whether the operator is *assertive* or whether it allows for an approximate matching (*fall-back*, *constraint relaxing*).

See Also:

(continued from last page)

`PropertyOperator.isAssertive()`

EQ

```
public static final cz.cuni.versatile.api.PropertyOperator EQ
```

Equals intrinsic assertive operator. When comparing two values, it invokes `Object.equals()` method.

See Also:

`Object.equals(java.lang.Object)`
`QueryTemplate.add_Equal(String)`

GT

```
public static final cz.cuni.versatile.api.PropertyOperator GT
```

Greater-than (>) intrinsic assertive operator. When comparing two values, it invokes `Comparable.compareTo()` method.

Remarks: Requires the objects to implement the `Comparable` interface.

See Also:

`Comparable.compareTo(java.lang.Object)`
`QueryTemplate.add_GT(String)`

LT

```
public static final cz.cuni.versatile.api.PropertyOperator LT
```

Less-than (<) intrinsic assertive operator. When comparing two values, it invokes `Comparable.compareTo()` method.

Remarks: Requires the objects to implement the `Comparable` interface.

See Also:

`Comparable.compareTo(java.lang.Object)`
`QueryTemplate.add_LT(String)`

GE

```
public static final cz.cuni.versatile.api.PropertyOperator GE
```

Greater than or equal to (>=) intrinsic assertive operator. When comparing two values, it invokes `Comparable.compareTo()` method.

Remarks: Requires the objects to implement the `Comparable` interface.

See Also:

`Comparable.compareTo(java.lang.Object)`
`QueryTemplate.add_GE(String)`

LE

```
public static final cz.cuni.versatile.api.PropertyOperator LE
```

Less than or equal to (<=) intrinsic assertive operator. When comparing two values, it invokes `Comparable.compareTo()` method.

Remarks: Requires the objects to implement the `Comparable` interface.

See Also:

(continued from last page)

Comparable.compareTo(java.lang.Object)
QueryTemplate.add_LE(String)

ASSERT

public static final cz.cuni.versatile.api.PropertyOperator **ASSERT**

assert extrinsic assertive operator. When matching two values, it invokes RelationalProperty.contains(x, y).

Remarks:

- Can only be applied to a RelationalProperty
- therefore it applies also to OrderProperty, Equivalence and Taxonomy

See Also:

RelationalProperty.contains(Object, Object)
QueryTemplate.addAssert(String)

ASSERT_INV

public static final cz.cuni.versatile.api.PropertyOperator **ASSERT_INV**

assertInv extrinsic assertive operator. When matching two values, it invokes RelationalProperty.contains(y, x), which is equivalent to InverseOf(p:RelationalProperty).contains(x, y).

Remarks:

- Can only be applied to a RelationalProperty
- therefore it applies also to OrderProperty, Equivalence and Taxonomy

See Also:

RelationalProperty.contains(Object, Object)
InverseOf
QueryTemplate.addAssertInv(String)

ASSERT_LEVEL

public static final cz.cuni.versatile.api.PropertyOperator **ASSERT_LEVEL**

(continued from last page)

assertLevel extrinsic assertive operator. Similar to *assert* but applies only to taxonomies. Besides the `RelationalProperty.contains(x, y)` it imposes an additional constraint on how far are `x` and `y` in the classification hierarchy (taxonomy). The additional mandatory parameter `level(int)` represents the additional constraint as follows:

- `level = 0` - equivalent to *assert*
- `level > 0` - absolute distance from the root of the taxonomy
- `level < 0` - relative distance from the context node

Remarks:

- Can only be applied to a `Taxonomy` property
- typically used together with the *bestMatch* operator to implement constrained fall-back search
- `level > 0` makes most sense for fixed-depth taxonomies (e.g. UNSPSC), for example, by combining "*bestMatch* **and** *assertLevel(2)*", we can match all entries which are equal-to or greater-than a given entry, up-to the second level of the 4-level UNSPSC hierarchy: given `43232203` (File versioning software), we generate the following search sequence to find an upper-bound best match:
  `43232203` (File versioning software)
  `43232200` (Content management software)
  `43230000` (Software)
- The parent entry `43000000` (Information Technology Broadcasting and Telecommunications) wont' match the query given the "*assertLevel(2)*" condition.
- `level < 0` on the other hand allows to express how many steps of the *best match* fall-back search we want to allow, given the example above and the query "*bestMatch* **and** *assertLevel(-1)*", only the exact match and optionally one step fall-back occurs:
  `43232203` (File versioning software)
  `43232200` (Content management software)

See Also:

`Taxonomy`
`BEST_MATCH`
`QueryTemplate.addAssertLevel(String, int)`

COMPARABLE

```
public static final cz.cuni.versatile.api.PropertyOperator COMPARABLE
```

comparable extrinsic assertive operator. Check wheter two property values are comparable given the particular order property.

Remarks:

- Can only be applied to an `OrderProperty` property

See Also:

`OrderProperty.comparable(Object, Object)`
`QueryTemplate.addComparable(String)`

EQUIVALENT

```
public static final cz.cuni.versatile.api.PropertyOperator EQUIVALENT
```

(continued from last page)

equivalent extrinsic assertive operator. Unlike the *equals* (an intrinsic operator) *equivalent* relies on `Equivalence` subtype of `RelationalProperty` to decide which values are considered "*equal*".

Remarks:

- Can only be applied to an `Equivalence` property

See Also:

`Equivalence`
`QueryTemplate.addEquivalent(String)`

IS_PARENT

`public static final cz.cuni.versatile.api.PropertyOperator` **`IS_PARENT`**

isParent extrinsic assertive operator. Please refer to `Taxonomy` API for details.

Remarks:

- Can only be applied to a `Taxonomy` property

See Also:

`Taxonomy.isParent(Object, Object)`
`QueryTemplate.addIsParent(String)`

IS_CHILD

`public static final cz.cuni.versatile.api.PropertyOperator` **`IS_CHILD`**

isChild extrinsic assertive operator. Please refer to `Taxonomy` API for details.

Remarks:

- Can only be applied to a `Taxonomy` property

See Also:

`Taxonomy.isChild(Object, Object)`
`QueryTemplate.addIsChild(String)`

IS_ANCESTOR

`public static final cz.cuni.versatile.api.PropertyOperator` **`IS_ANCESTOR`**

isAncestor extrinsic assertive operator. Please refer to `Taxonomy` API for details.

Remarks:

- Can only be applied to a `Taxonomy` property

See Also:

`Taxonomy.isAncestor(Object, Object)`
`QueryTemplate.addIsAncestor(String)`

IS_DESCENDANT

```
public static final cz.cuni.versatile.api.PropertyOperator IS_DESCENDANT
```

isDescendant extrinsic assertive operator. Please refer to `Taxonomy` API for details.

Remarks:

- Can only be applied to a `Taxonomy` property
- It is equivalent to `RelationalProperty.contains()` in case of `Taxonomy`.

See Also:

`Taxonomy.isDescendant(Object, Object)`
`QueryTemplate.addIsDescendant(String)`

BEST_MATCH

```
public static final cz.cuni.versatile.api.PropertyOperator BEST_MATCH
```

bestMatch extrinsic constraint-relaxing operator. Probably the most powerful and useful Versatile operator: it starts with a given context node = the actual property value obtained from its `ValueProvider` and tries to perform an exact match, if no resource is found, it uses `Taxonomy#getAncestorIterator()` to generate a sequence of candidates in ascending order, leveraging the classification hierarchy of the taxonomy. With properly designed taxonomies in place, one can thus easily implement quite sophisticated *fall-back* strategies using hierarchical *defaulting* (constraint relaxing).

Remarks:

- Can only be applied to a `Taxonomy` property
- An example using UNSPSC taxonomy: given `43232203` (File versioning software) we generate the following search sequence to find the closest match:
  `43232203` (File versioning software)
  `43232200` (Content management software)
  `43230000` (Software)
  `43000000` (Information Technology Broadcasting and Telecommunications)
  `/` (root = property value not set = universal concept)

See Also:

`ASSERT_LEVEL`
`Taxonomy.getAncestorIterator(Object, Comparator)`
`QueryTemplate.addBestMatch(String)`

Constructors

PropertyOperators

```
protected PropertyOperators(int id,
                            String name,
                            boolean intrinsic,
                            boolean assertive)
```

Protected Constructor - can be re-used by subclasses extending the set of supported operators.

Parameters:

`id` - unique operator ID used for processing query predicates
`name` - human readable equivalent for debugging and logging
`intrinsic` - intrinsic/extrinsic flag

Methods

getId

```
public int getId()
```

See Also:
PropertyOperator.getId()

---

getName

```
public String getName()
```

See Also:
PropertyOperator.getName()

---

isIntrinsic

```
public boolean isIntrinsic()
```

See Also:
PropertyOperator.isIntrinsic()

---

isExtrinsic

```
public boolean isExtrinsic()
```

See Also:
PropertyOperator.isExtrinsic()

---

isAssertive

```
public boolean isAssertive()
```

Index

H

I

L

M

N

P

R

S

T

U