



JDO Annotations Reference (v5.2)

Table of Contents

JDO Class-Level Annotations	2
@PersistenceCapable	3
@PersistenceAware	3
@Cacheable	4
@EmbeddedOnly	4
@Inheritance	4
@Discriminator	5
@DatastoreIdentity	5
@Version	6
@PrimaryKey	7
@FetchPlan	7
@FetchGroup	8
@Sequence	8
@Query	9
@Index	10
@Unique	10
@ForeignKey	11
@Join	12
JDO Field-Level Annotations	13
@Persistent	13
@Serialized	16
@NotPersistent	16
@Transactional	16
@Cacheable	17
@PrimaryKey	17
@Element	17
@Order	19
@Key	19
@Value	21
@Join	22
@Embedded	23
@Column	24
@Index	25
@Unique	26
@ForeignKey	26
@Convert	27
@Extension	28
DataNucleus Class-Level Extensions	29

@ReadOnly	29
@MultiTenant	29
@SoftDelete	30
DataNucleus Field-Level Extensions	31
@SharedRelation	31
@ReadOnly	32
@CreateTimestamp	32
@UpdateTimestamp	32
Meta-Annotations	33

JDO provides the ability to use annotations to define the persistence of entities, and DataNucleus JDO supports both JDO and JPA annotations. In this section we provide a reference to the primary JDO annotations. When selecting to use annotations please bear in mind the following :-

- You must have the `datanucleus-api-jdo` jar available in your CLASSPATH.
- You must have the `javax.jdo` jar in your CLASSPATH since this provides the annotations
- Annotations should really only be used for attributes of persistence that you won't be changing at deployment. Things such as table and column names shouldn't really be specified using annotations although it is permitted. Instead it would be better to put such information in an ORM MetaData file.
- Annotations can be added in two places - for the class as a whole, or for a field in particular.
- You can annotate fields or getters with field-level information. If you annotate fields then the fields are processed for persistence. If you annotate the methods (getters) then the methods (properties) are processed for persistence.
- Annotations are prefixed by the `@` symbol and can take attributes (in brackets after the name, comma-separated)
- DataNucleus provides its own annotations for some extension features.
- You have to import `javax.jdo.annotations.XXX` where XXX is the annotation name of a JDO annotation
- You have to import `org.datanucleus.api.jdo.annotations.XXX` where XXX is the annotation name of a DataNucleus value-added annotation

Annotations supported by DataNucleus are shown below.

JDO Class-Level Annotations

The following annotations are specified at class-level and are JDO standard. Using these provide portability for your application.

Annotation	Class/Field	Description
@PersistenceCapable	Class	Specifies that the class/interface is persistent. In the case of an interface this would utilise JDO's "persistent-interface" capabilities
@PersistenceAware	Class	Specifies that the class is not persistent but needs to be able to access fields of persistent classes
@Cacheable	Class	Specifies whether this class can be cached in a Level 2 cache or not.
@EmbeddedOnly	Class	Specifies that the class is persistent and can only be persisted embedded in another persistent class
@DatastoreIdentity	Class	Specifies the details for generating datastore-identity for this class
@Version	Class	Specifies any versioning process for objects of this class
@FetchPlan	Class	Defines a fetch plan
@FetchGroup	Class	Defines a fetch group for this class
@Sequence	Class	Defines a sequence for use by this class
@Query	Class	Defines a named query for this class
@Inheritance	Class	Specifies the inheritance model for persisting this class
@Discriminator	Class	Specifies any discriminator for this class to be used for determining object types
@PrimaryKey	Class	ORM : Defines the primary key constraint for this class
@Index	Class	ORM : Defines an index for the class as a whole (typically a composite index)
@Unique	Class	ORM : Defines a unique constraint for the class as a whole (typically a composite)
@ForeignKey	Class	ORM : Defines a foreign-key for the class as a whole (typically for non-mapped columns/tables)
@Join	Class	ORM : Defines a join to a secondary table from this table
@Column	Class	ORM : Defines a column that doesn't have associated fields ("unmapped columns")
@Extension	Class/Field/Method	Defines a JDO extension

@PersistenceCapable

This annotation is used when you want to mark a class as persistent. It equates to the <class> XML element (though with only some of its attributes). Specified on the **class**.

Attribute	Type	Description	Default
requiresExtent	String	Whether an extent is required for this class	true
embeddedOnly	String	Whether objects of this class can only be stored embedded in other objects	false
detachable	String	Whether objects of this class can be detached	false
identityType	IdentityType	Type of identity (APPLICATION, DATASTORE, NONDURABLE)	
objectIdClass	Class	Object-id class	
cacheable	String	Whether the class can be L2 cached.	true , false
serializeRead	String	Whether to default reads of this object type to lock the object	false
extensions	Extension	Vendor extensions	
table	String	ORM : Name of the table where this class is persisted	
catalog	String	ORM : Name of the catalog where this table is persisted	
schema	String	ORM : Name of the schema where this table is persisted	

```
@PersistenceCapable(identityType=IdentityType.APPLICATION)
public class MyClass
{
    ...
}
```

@PersistenceAware

This annotation is used when you want to mark a class as being used in persistence but not being persistable. That is "persistence-aware" in JDO terminology. It has no attributes. Specified on the **class**.

```
@PersistenceAware
public class MyClass
{
    ...
}
```

See the documentation for [Class Mapping](#)

@Cacheable

This annotation is a shortcut for `@PersistenceCapable(cacheable={value})` specifying whether the class can be cached in a Level 2 cache. Specified on the **class**.

Attribute	Type	Description	Default
value	String	Whether the class is cacheable	true , false

```
@Cacheable("false")
public class MyClass
{
    ...
}
```

See the documentation for [L2 Caching](#)

@EmbeddedOnly

This annotation is a shortcut for `@PersistenceCapable(embeddedOnly="true")` meaning that the class can only be persisted embedded into another class. It has no attributes. Specified on the **class**.

```
@EmbeddedOnly
public class MyClass
{
    ...
}
```

@Inheritance

Annotation used to define the inheritance for a class. Specified on the **class**.

Attribute	Type	Description	Default
strategy	InheritanceStrategy	The inheritance strategy (NEW_TABLE, SUBCLASS_TABLE, SUPERCLASS_TABLE)	
customStrategy	String	Name of a custom inheritance strategy (DataNucleus supports "complete-table")	

```

@PersistenceCapable
@Inheritance(strategy=InheritanceStrategy.NEW_TABLE)
public class MyClass
{
    ...
}

```

See the documentation for [Inheritance](#)

@Discriminator

Annotation used to define a discriminator to be stored with instances of this class and is used to determine the types of the objects being stored. Specified on the **class**.

Attribute	Type	Description	Default
strategy	DiscriminatorStrategy	The discriminator strategy (VALUE_MAP, CLASS_NAME, NONE)	
value	String	Value to use for instances of this type when using strategy of VALUE_MAP	
column	String	ORM : Name of the column to use to store the discriminator	
indexed	String	ORM : Whether the discriminator column is to be indexed	
columns	Column	ORM : Column definitions used for storing the discriminator	

```

@PersistenceCapable
@Inheritance(strategy=InheritanceStrategy.NEW_TABLE)
@Discriminator(strategy=DiscriminatorStrategy.CLASS_NAME)
public class MyClass
{
    ...
}

```

@DatastoreIdentity

Annotation used to define the identity when using datastore-identity for the class. Specified on the **class**.

Attribute	Type	Description	Default
strategy	IdGeneratorStrategy	The inheritance strategy (NATIVE, SEQUENCE, IDENTITY, INCREMENT, UUIDSTRING, UUIDHEX)	

Attribute	Type	Description	Default
customStrategy	String	Name of a custom id generation strategy (e.g "max", "auid"). This overrides the value of "strategy"	
sequence	String	Name of the sequence to use (when using SEQUENCE strategy) - refer to @Sequence	
extensions	Extension	Vendor extensions	
column	String	ORM : Name of the column for the datastore identity	
columns	Column	ORM : Column definition for the column(s) for the datastore identity	

```
@PersistenceCapable
@DatastoreIdentity(strategy=IdGeneratorStrategy.INCREMENT)
public class MyClass
{
    ...
}
```

See the documentation for [Datastore Identity](#)

@Version

Annotation used to define the versioning details for use with optimistic transactions. Specified on the **class**.

Attribute	Type	Description	Default
strategy	VersionStrategy	The version strategy (NONE, STATE_IMAGE, DATE_TIME, VERSION_NUMBER)	
indexed	String	Whether the version column(s) is indexed	
extensions	Extension	Vendor extensions	
column	String	ORM : Name of the column for the version	
columns	Column	ORM : Column definition for the column(s) for the version	

```
@PersistenceCapable
@Version(strategy=VersionStrategy.VERSION_NUMBER)
public class MyClass
{
    ...
}
```

See the documentation for [Optimistic Transactions](#)

@PrimaryKey

Annotation used to define the primary key constraint for a class. Maps across to the <primary-key> XML element. Specified on the **class**.

Attribute	Type	Description	Default
name	String	ORM : Name of the primary key constraint	
column	String	ORM : Name of the column for this key	
columns	Column	ORM : Column definition for the column(s) of this key	

```
@PersistenceCapable
@PrimaryKey(name="MYCLASS_PK")
public class MyClass
{
    ...
}
```

@FetchPlan

Annotation used to define a fetch plan. Is equivalent to the <fetch-plan> XML element. Specified on the **class**. Used by named queries

Attribute	Type	Description	Default
name	String	Name of the FetchPlan	
maxFetchDepth	int	Maximum fetch depth	1
fetchSize	int	Size hint for fetching query result sets	0
fetchGroups	String[]	Names of the fetch groups included in this FetchPlan.	

See the documentation for [FetchGroups](#)



There is a [@FetchPlans](#) annotation but in JDO 3.2 you can simply use multiple [@FetchPlan](#) to achieve the same cleaner

```

@PersistenceCapable
@FetchPlan(name="plan_3", maxFetchDepth=3, fetchGroups={"group1", "group4"})
public class MyClass
{
    ...
}

```

See the documentation for [FetchGroups](#)

@FetchGroup

Annotation used to define a fetch group. Is equivalent to the <fetch-group> XML element. Specified on the **class**.

Attribute	Type	Description	Default
name	String	Name of the fetch group	
postLoad	String	Whether to call jdoPostLoad after loading this fetch group	
members	Persistent	Definitions of the fields/properties to include in this fetch group	

```

@PersistenceCapable
@FetchGroup(name="one_two", members={@Persistent(name="field1"), @Persistent(name="field2")})
public class MyClass
{
    @Persistent
    String field1;

    @Persistent
    String field2;
    ...
}

```

See the documentation for [FetchGroups](#)



There is a [@FetchGroups](#) annotation but in JDO 3.2 you can simply use multiple [@FetchGroup](#) to achieve the same cleaner

@Sequence

Annotation used to define a sequence generator. Is equivalent to the <sequence> XML element. Specified on the **class**.

Attribute	Type	Description	Default
name	String	Name of the sequence	
strategy	SequenceStrategy	Strategy for the sequence (NONTRANSACTIONAL, CONTIGUOUS, NONCONTIGUOUS)	
datastoreSequence	String	Name of a datastore sequence that this maps to	
factoryClass	Class	Factory class to use to generate the sequence	
initialValue	int	Initial value of the sequence	1
allocationSize	int	Allocation size of the sequence	50
extensions	Extension	Vendor extensions	

See the documentation for [Sequences](#)

@Query

Annotation used to define a named query. Is equivalent to the <query> XML element. Specified on the **class**.

Attribute	Type	Description	Default
name	String	Name of the query	
value	String	The query string itself	
language	String	Language of the query (JDOQL, SQL, ...)	JDOQL
unmodifiable	String	Whether the query is not modifiable at runtime	
unique	String	Whether the query returns unique results (for SQL queries only)	
resultClass	Class	Result class to use (for SQL queries only)	
fetchPlan	String	Name of a named FetchPlan to use with this query	
extensions	Extension	Vendor extensions	

```

@PersistenceCapable
@Query(name="PeopleCalledSmith", language="JDOQL",
      value="SELECT FROM mydomain.samples.Person WHERE surname == \"Smith\"")
public class Person
{
    @Persistent
    String surname;

    ...
}

```

See the documentation for [Named Queries](#)



There is a `@Queries` annotation but in JDO 3.2 you can simply use multiple `@Query` to achieve the same cleaner

@Index

Annotation used to define an index for the class as a whole typically being a composite index across multiple columns or fields/properties. Is equivalent to the `<index>` XML element when specified under class. Specified on the **class**.

Attribute	Type	Description	Default
name	String	ORM : Name of the index	
table	String	ORM : Name of the table for the index	
unique	String	ORM : Whether the index is unique	
members	String[]	ORM : Names of the fields/properties that make up this index	
columns	Column	ORM : Columns that make up this index	
extensions	Extension	Vendor extensions	

```
@PersistenceCapable
@Index(name="MY_COMPOSITE_IDX", members={"field1", "field2"})
public class MyClass
{
    @Persistent
    String field1;

    @Persistent
    String field2;

    ...
}
```

See the documentation for [Schema Constraints](#)



There is a `@Indices` annotation but in JDO 3.2 you can simply use multiple `@Index` to achieve the same cleaner

@Unique

Annotation used to define a unique constraints for the class as a whole typically being a composite constraint across multiple columns or fields/properties. Is equivalent to the `<unique>` XML element when specified under class. Specified on the **class**.

Attribute	Type	Description	Default
name	String	ORM : Name of the constraint	
table	String	ORM : Name of the table for the constraint	
deferred	String	ORM : Whether the constraint is deferred	
members	String[]	ORM : Names of the fields/properties that make up this constraint	
columns	Column	ORM : Columns that make up this constraint	
extensions	Extension	Vendor extensions	

```

@PersistenceCapable
@Unique(name="MY_COMPOSITE_IDX", members={"field1", "field2"})
public class MyClass
{
    @Persistent
    String field1;

    @Persistent
    String field2;

    ...
}

```

See the documentation for [Schema Constraints](#)



There is a `@Uniques` annotation but in JDO 3.2 you can simply use multiple `@Unique` to achieve the same cleaner

@ForeignKey

Annotation used to define a foreign-key constraint for the class. Specified on the **class**.

Attribute	Type	Description	Default
name	String	ORM : Name of the constraint	
table	String	ORM : Name of the table that the FK is to	
deferred	String	ORM : Whether the constraint is deferred	
unique	String	ORM : Whether the constraint is unique	
deleteAction	ForeignKeyAction	ORM : Action to apply to the FK to be used on deleting	ForeignKeyAction.RESTRICT
updateAction	ForeignKeyAction	ORM : Action to apply to the FK to be used on updating	ForeignKeyAction.RESTRICT

Attribute	Type	Description	Default
members	String[]	ORM : Names of the fields/properties that compose this FK.	
columns	Column	ORM : Columns that compose this FK.	

See the documentation for [Schema Constraints](#)



There is a `@ForeignKey` annotation but in JDO 3.2 you can simply use multiple `@ForeignKey` to achieve the same cleaner

@Join

Annotation used to specify a join for a secondary table. Specified on the **class**.

Attribute	Type	Description	Default
table	String	ORM : Table name used when joining the PK of a FCO class table to a secondary table.	
column	String	ORM : Name of the column used to join to the PK of the primary table (when only one column used)	
outer	String	ORM : Whether to use an outer join when retrieving fields/properties stored in the secondary table	
columns	Column	ORM : Name of the columns used to join to the PK of the primary table (when multiple columns used)	
extensions	Extension	Vendor extensions	

```
@PersistenceCapable(name="MYTABLE")
@Join(table="MY_OTHER_TABLE", column="MY_PK_COL")
public class MyClass
{
    @Persistent(name="MY_OTHER_TABLE")
    String myField;
    ...
}
```



There is a `@Joins` annotation but in JDO 3.2 you can simply use multiple `@Join` to achieve the same cleaner

JDO Field-Level Annotations

The following annotations are specified at field/method-level and are JDO standard. Using these provide portability for your application.

Annotation	Class/Field	Description
@Persistent	Field/Method	Defines the persistence for a field/property of the class
@Serialized	Field/Method	Defines this field as being stored serialised
@NotPersistent	Field/Method	Defines this field as being not persisted
@Transactional	Field/Method	Defines this field as being transactional (not persisted, but managed)
@Cacheable	Field/Method	Specifies whether this field/property can be cached in a Level 2 cache or not.
@PrimaryKey	Field/Method	Defines this field as being (part of) the primary key
@Element	Field/Method	Defines the details of elements of an array/collection stored in this field
@Key	Field/Method	Defines the details of keys of a map stored in this field
@Value	Field/Method	Defines the details of values of a map stored in this field
@Convert	Field/Method	Specify an AttributeConverter for this field/method
@Extension	Class/Field/Method	Defines a JDO extension
@Order	Field/Method	ORM : Defines the details of ordering of an array/collection stored in this field
@Join	Field/Method	ORM : Defines the join to a join table for a collection/array/map
@Embedded	Field/Method	ORM : Defines that this field is embedded and how it is embedded
@Column	Field/Method	ORM : Defines a column where a field is persisted
@Index	Field/Method	ORM : Defines an index for the field
@Unique	Field/Method	ORM : Defines a unique constraint for the field
@ForeignKey	Field/Method	ORM : Defines a foreign key for the field

@Persistent

Annotation used to define the fields/properties to be persisted. Is equivalent to the <field> and <property> XML elements. Specified on the **field/method**.

Attribute	Type	Description	Default
persistenceModifier	PersistenceModifier	Whether the field is persistent (PERSISTENT, TRANSACTIONAL, NONE)	[depends on field type]

Attribute	Type	Description	Default
defaultFetchGroup	String	Whether the field is part of the DFG	
nullValue	NullValue	Required behaviour when inserting a null value for this field (NONE, EXCEPTION, DEFAULT).	NONE
embedded	String	Whether this field as a whole is embedded. Use @Embedded to specify details.	
embeddedElement	String	Whether the element stored in this collection/array field/property is embedded	
embeddedKey	String	Whether the key stored in this map field/property is embedded	
embeddedValue	String	Whether the value stored in this map field/property is embedded	
serialized	String	Whether this field/property as a whole is serialised	
serializedElement	String	Whether the element stored in this collection/array field/property is serialised	
serializedKey	String	Whether the key stored in this map field/property is serialised	
serializedValue	String	Whether the value stored in this map field/property is serialised	
dependent	String	Whether this field is dependent, deleting the related object when deleting this object	
dependentElement	String	Whether the element stored in this field/property is dependent	
dependentKey	String	Whether the key stored in this field/property is dependent	
dependentValue	String	Whether the value stored in this field/property is dependent	
primaryKey	String	Whether this field is (part of) the primary key	false
valueStrategy	IdGeneratorStrategy	Strategy to use when generating values for the field (NATIVE, SEQUENCE, IDENTITY, INCREMENT, UUIDSTRING, UUIDHEX)	
customValueStrategy	String	Name of a custom id generation strategy (e.g "max", "auid"). This overrides the value of "valueStrategy"	
sequence	String	Name of the sequence when using valueStrategy of SEQUENCE - refer to @Sequence	

Attribute	Type	Description	Default
types	Class[]	Type(s) of field (when using interfaces/reference types). DataNucleus currently only supports the first value although in the future it is hoped to support multiple.	
mappedBy	String	Field in other class when the relation is bidirectional to signify the owner of the relation	
name	String	Name of the field when defining an embedded field.	
cacheable	String	Whether the field/property can be L2 cached.	true , false
recursionDepth	int	Recursion depth for this field when fetching. Only applicable when specified within @FetchGroup	1
loadFetchGroup	String	Name of a fetch group to activate when a load of this field is initiated (due to it being currently unloaded). Not used for getObjectById, queries, extents etc. Better to use @FetchGroup and define your groups	
converter	Class	Converter class that implements javax.jdo.AttributeConverter	
useDefaultConversion	boolean	Whether we should disable any default conversion for this field	false
extensions	Extension	Vendor extensions	
table	String	ORM : Name of the table where this field is persisted. If this field is a collection/map/array then the table refers to a join table, otherwise this refers to a secondary table.	
columns	Column	ORM : Column definition(s) for the columns into which this field is persisted. This is only typically used when specifying columns of a field of an embedded class.	

```

@PersistenceCapable
public class MyClass
{
    @Persistent(primaryKey="true")
    String myField;
    ...
}

```

See the documentation for [Fields/Properties](#)

@Serialized

This annotation is a shortcut for `@Persistent(serialized="true")` meaning that the field is stored serialized. It has no attributes. Specified on the **field/method**.

```
@PersistenceCapable
public class MyClass
{
    @Serialized
    Object myField;
    ...
}
```

See the documentation for [Serialising](#)

@NotPersistent

This annotation is a shortcut for `@Persistent(persistenceModifier=PersistenceModifier.NONE)` meaning that the field/property is not persisted. It has no attributes. Specified on the **field/method**.

```
@PersistenceCapable
public class MyClass
{
    @NotPersistent
    String myOtherField;
    ...
}
```

See the documentation for [Fields/Properties](#)

@Transactional

This annotation is a shortcut for `@Persistent(persistenceModifier=PersistenceModifier.TRANSACTIONAL)` meaning that the field/property is not persisted yet managed. It has no attributes. Specified on the **field/method**.

```
@PersistenceCapable
public class MyClass
{
    @Transactional
    String myOtherField;
    ...
}
```

See the documentation for [Fields/Properties](#)

@Cacheable

This annotation is a shortcut for `@Persistent(cacheable={value})` specifying whether the field/property can be cached in a Level 2 cache. Specified on the **field/property**. The default

Attribute	Type	Description	Default
value	String	Whether the field/property is cacheable	true, false

```
public class MyClass
{
    @Cacheable("false")
    Collection elements;
    ...
}
```

See the documentation for [L2 Caching](#)

@PrimaryKey

This annotation is a shortcut for `@Persistent(primaryKey="true")` meaning that the field/property is part of the primary key for the class. No attributes are needed when specified like this. Specified on the **field/method**.

```
@PersistenceCapable
public class MyClass
{
    @PrimaryKey
    String myOtherField;
    ...
}
```

See the documentation for [Schema Constraints](#)



If the primary key is composite, then the order of the columns in the corresponding index (in the database) will be determined by the value of the `@Column#position` property.

Alternatively, consider using the `@PrimaryKey` at the class level, specifying the `@PrimaryKey#columns` property.

@Element

Annotation used to define the element for any collection/array to be persisted. Maps across to the `<collection>`, `<array>` and `<element>` XML elements. Specified on the Collection/array **field/method**.

Attribute	Type	Description	Default
types	Class[]	Type(s) of element. While the attribute allows multiple values DataNucleus currently only supports the first type value	When using an array is not needed. When using a collection will be taken from the collection definition if using generics, otherwise must be specified.
embedded	String	Whether the element is embedded into a join table	
serialized	String	Whether the element is serialised into the join table	
dependent	String	Whether the element objects are dependent when deleting the owner collection/array	
mappedBy	String	Field in the element class that represents this object (when the relation is bidirectional)	
embeddedMapping	Embedded	Definition of any embedding of the (persistable) element. Only 1 "Embedded" should be provided	
converter	Class	Converter class that implements <code>javax.jdo.AttributeConverter</code>	
useDefaultConversion	boolean	Whether we should disable any default conversion for this element	false
extensions	Extension	Vendor extensions	
table	String	ORM : Name of the table for this element	
column	String	ORM : Name of the column for this element	
foreignKey	String	ORM : Name of any foreign-key constraint to add	
generateForeignKey	String	ORM : Whether to generate a FK constraint for the element (when not specifying the name)	
deleteAction	ForeignKeyAction	ORM : Action to be applied to the foreign key for this element for action upon deletion	
updateAction	ForeignKeyAction	ORM : Action to be applied to the foreign key for this element for action upon update	
index	String	ORM : Name of any index constraint to add	
indexed	String	ORM : Whether this element column is indexed	
unique	String	ORM : Whether this element column is unique	
uniqueKey	String	ORM : Name of any unique key constraint to add	

Attribute	Type	Description	Default
columns	Column	ORM : Column definition for the column(s) of this element	

```
@PersistenceCapable
public class MyClass
{
    @Element(types=mydomain.samples.MyElementClass.class, dependent="true")
    Collection myField;
    ...
}
```

@Order

Annotation used to define the ordering of an order-based Collection/array to be persisted. Maps across to the <order> XML element. Specified on the **field/method**.

Attribute	Type	Description	Default
extensions	Extension	Vendor extensions	
mappedBy	String	ORM : Field in the element class that represents the ordering of the collection/array	
column	String	ORM : Name of the column for this order	
columns	Column	ORM : Column definition for the column(s) of this order	

```
@PersistenceCapable
public class MyClass
{
    @Element(types=mydomain.samples.MyElementClass.class, dependent="true")
    @Order(column="ORDER_IDX")
    Collection myField;
    ...
}
```

@Key

Annotation used to define the key for any map to be persisted. Maps across to the <map> and <key> XML elements. Specified on the **field/method**.

Attribute	Type	Description	Default
types	Class[]	Type(s) of key. While the attribute allows multiple values DataNucleus currently only supports the first type value	When using generics will be taken from the Map definition, otherwise must be specified
embedded	String	Whether the key is embedded into a join table	
serialized	String	Whether the key is serialised into the join table	
dependent	String	Whether the key objects are dependent when deleting the owner map	
mappedBy	String	Used to specify the field in the value class where the key is stored (optional).	
embeddedMapping	Embedded	Definition of any embedding of the (persistable) key. Only 1 "Embedded" should be provided	
converter	Class	Converter class that implements javax.jdo.AttributeConverter	
useDefaultConversion	boolean	Whether we should disable any default conversion for this key	false
extensions	Extension	Vendor extensions	
table	String	ORM : Name of the table for this key	
column	String	ORM : Name of the column for this key	
foreignKey	String	ORM : Name of any foreign-key constraint to add	
generateForeignKey	String	ORM : Whether to generate a FK constraint for the key (when not specifying the name)	
deleteAction	ForeignKeyAction	ORM : Action to be applied to the foreign key for this key for action upon deletion	
updateAction	ForeignKeyAction	ORM : Action to be applied to the foreign key for this key for action upon update	
index	String	ORM : Name of any index constraint to add	
indexed	String	ORM : Whether this key column is indexed	
uniqueKey	String	ORM : Name of any unique key constraint to add	
unique	String	ORM : Whether this key column is unique	
columns	Column	ORM : Column definition for the column(s) of this key	

```

@PersistenceCapable
public class MyClass
{
    @Key(types=java.lang.String.class)
    Map myField;
    ...
}

```

@Value

Annotation used to define the value for any map to be persisted. Maps across to the <map> and <value> XML elements. Specified on the **field/method**.

Attribute	Type	Description	Default
types	Class[]	Type(s) of value. While the attribute allows multiple values DataNucleus currently only supports the first type value	When using generics will be taken from the Map definition, otherwise must be specified
embedded	String	Whether the value is embedded into a join table	
serialized	String	Whether the value is serialised into the join table	
dependent	String	Whether the value objects are dependent when deleting the owner map	
mappedBy	String	Used to specify the field in the key class where the value is stored (optional).	
embeddedMapping	Embedded	Definition of any embedding of the (persistable) value. Only 1 "Embedded" should be provided	
converter	Class	Converter class that implements javax.jdo.AttributeConverter	
useDefaultConversion	boolean	Whether we should disable any default conversion for this value	false
extensions	Extension	Vendor extensions	
table	String	ORM : Name of the table for this value	
column	String	ORM : Name of the column for this value	
foreignKey	String	ORM : Name of any foreign-key constraint to add	
deleteAction	ForeignKeyAction	ORM : Action to be applied to the foreign key for this value for action upon deletion	

Attribute	Type	Description	Default
generateForeignKey	String	ORM : Whether to generate a FK constraint for the value (when not specifying the name)	
updateAction	ForeignKeyAction	ORM : Action to be applied to the foreign key for this value for action upon update	
index	String	ORM : Name of any index constraint to add	
indexed	String	ORM : Whether this value column is indexed	
uniqueKey	String	ORM : Name of any unique key constraint to add	
unique	String	ORM : Whether this value column is unique	
columns	Column	ORM : Column definition for the column(s) of this value	

```

@PersistenceCapable
public class MyClass
{
    @Key(types=java.lang.String.class)
    @Value(types=mydomain.samples.MyValueClass.class, dependent="true")
    Map myField;
    ...
}

```

@Join

Annotation used to specify a join to a join table for a collection/array/map. Specified on the **field/method**.

Attribute	Type	Description	Default
extensions	Extension	Vendor extensions	
table	String	ORM : Not used when specified on a field/property, use @Persistent(table="...") instead	
column	String	ORM : Name of the column to join our PK to in the join table (when only one column used)	
primaryKey	String	ORM : Name of any primary key constraint to add for the join table	
generatePrimaryKey	String	ORM : Whether to generate a PK constraint on the join table (when not specifying the name)	
foreignKey	String	ORM : Name of any foreign-key constraint to add	
generateForeignKey	String	ORM : Whether to generate a FK constraint on the join table (when not specifying the name)	

Attribute	Type	Description	Default
index	String	ORM : Name of any index constraint to add	
indexed	String	ORM : Whether the join column(s) is indexed	
uniqueKey	String	ORM : Name of any unique constraint to add	
unique	String	ORM : Whether the join column(s) has a unique constraint	
columns	Column	ORM : Name of the columns to join our PK to in the join table (when multiple columns used)	

```

@PersistenceCapable
public class MyClass
{
    @Persistent
    @Element(types=mydomain.samples.MyElement.class)
    @Join(table="MYCLASS_ELEMENTS", column="MYCLASS_ELEMENTS_PK")
    Collection myField;
    ...
}

```

@Embedded

Annotation used to define that the field contents is embedded into the same table as this field Maps across to the <embedded> XML element. Specified on the **field/method**.

Attribute	Type	Description	Default
ownerMember	String	ORM : The field/property in the embedded object that links back to the owning object (where it has a bidirectional relation)	
nullIndicatorColumn	String	ORM : The column in the embedded object used to judge if the embedded object is null.	
nullIndicatorValue	String	ORM : The value in the null column to interpret the object as being null.	
members	Persistent	ORM : Field/property definitions for this embedding.	

```

@PersistenceCapable
public class MyClass
{
    @Embedded(members={
        @Persistent(name="field1", columns=@Column(name="OTHER_FLD_1")),
        @Persistent(name="field2", columns=@Column(name="OTHER_FLD_2"))
    })
    MyOtherClass myField;
    ...
}

@PersistenceCapable
@EmbeddedOnly
public class MyOtherClass
{
    @Persistent
    String field1;

    @Persistent
    String field2;
}

```

@Column

Annotation used to define that the column where a field is persisted. Is equivalent to the <column> XML element when specified under field. Specified on the **field/method** (and within other annotations).

Attribute	Type	Description	Default
extensions	Extension	Vendor extensions	
name	String	ORM : Name of the column	
target	String	ORM : Column in the other class that this maps to. This is for use when you have a composite PK so acts as a way of aligning the respective columns. It is not to allow joining to some non-PK column	
targetMember	String	ORM : Field/Property in the other class that this maps to. This is for use when you have a composite PK so acts as a way of aligning the respective columns. It is not to allow joining to some non-PK column	
jdbcType	String	ORM : JDBC Type to use for persisting into this column	
sqlType	String	ORM : SQL Type to use for persisting into this column	

Attribute	Type	Description	Default
length	int	ORM : Max length of data to store in this column	
scale	int	ORM : Max number of floating points of data to store in this column	
allowsNull	String	ORM : Whether null is allowed to be persisted into this column	
defaultValue	String	ORM : Default value to persist into this column. If you want the default to be NULL, then put this as "#NULL"	
insertValue	String	ORM : Value to insert into this column when it is an "unmapped" column. If you want the inserted value to be NULL, then put this as "#NULL"	
position	int	ORM : Position of this column in the owning table (0 = first). Otherwise the column order is alphabetical. If the field is also part of a <code>@PrimaryKey</code> , then this position also is used for the relative position within the corresponding primary key index.	

```
@PersistenceCapable
public class MyClass
{
    @Persistent
    @Column(name="MYCOL", jdbcType="VARCHAR", length=40)
    String field1;

    ...
}
```



There is a `@Columns` annotation but in JDO 3.2 you can simply use multiple `@Columns` to achieve the same cleaner

@Index

Annotation used to define that this field is indexed. Is equivalent to the `<index>` XML element when specified under field. Specified on the **field/method**.

Attribute	Type	Description	Default
name	String	ORM : Name of the index	
unique	String	ORM : Whether the index is unique	

```

@PersistenceCapable
public class MyClass
{
    @Persistent
    @Index(name="MYFIELD1_IDX")
    String field1;

    @Persistent
    @Index(name="MYFIELD2_IDX", unique="true")
    String field2;

    ...
}

```

See the documentation for [Schema Constraints](#)

@Unique

Annotation used to define that this field has a unique constraint. Is equivalent to the <unique> XML element when specified under field. Specified on the **field/method**.

Attribute	Type	Description	Default
name	String	ORM : Name of the constraint	
deferred	String	ORM : Whether the constraint is deferred	

```

@PersistenceCapable
public class MyClass
{
    @Persistent
    @Unique(name="MYFIELD1_IDX")
    String field1;

    ...
}

```

See the documentation for [Schema Constraints](#)

@ForeignKey

Annotation used to define the foreign key for a relationship field. Is equivalent to the <foreign-key> XML element when specified under field. Specified on the **field/method**.

Attribute	Type	Description	Default
name	String	ORM : Name of the constraint	

Attribute	Type	Description	Default
deferred	String	ORM : Whether the constraint is deferred	
unique	String	ORM : Whether the constraint is unique	
deleteAction	ForeignKeyAction	ORM : Action to apply to the FK to be used on deleting	ForeignKeyAction.RESTRICT
updateAction	ForeignKeyAction	ORM : Action to apply to the FK to be used on updating	ForeignKeyAction.RESTRICT

```
@PersistenceCapable
public class MyClass
{
    @Persistent
    @ForeignKey(name="MYFIELD1_FK", deleteAction=ForeignKeyAction.RESTRICT)
    String field1;

    ...
}
```

See the documentation for [Schema Constraints](#)

@Convert

Annotation used to mark a field for conversion using an AttributeConverter. Specified on the **field/method**.

Attribute	Type	Description	Default
value	Class	Class for the AttributeConverter to use for this field	
enabled	boolean	Setting this to false allows us to disable (default) conversion (for this type) that was defined at PMF level	true

```
@PersistenceCapable
public class MyClass
{
    @Persistent
    @Convert(MyURLConverter.class)
    URL url;

    ...
}
```

@Extension

Annotation used to define an extension specific to a particular JDO implementation. Is equivalent to the <extension> XML element. Specified on the **class** or **field**.

Attribute	Type	Description	Default
vendorName	String	Name of the JDO vendor	
key	String	Key for the extension	
value	String	Value of the extension	

```
@PersistenceCapable
@Extension(vendorName="DataNucleus", key="RunFast", value="true")
public class Person
{
    ...
}
```



There is a `@Extensions` annotation but in JDO 3.2 you can simply use multiple `@Extension` to achieve the same cleaner

DataNucleus Class-Level Extensions

The following annotations are specified at class-level and are vendor extensions providing more functionality than the JDO spec defines. Using these will reduce the portability of your application.

Annotation	Class/Field	Description
@ReadOnly	Class	Specifies that this class is "read-only" (DataNucleus extension).
@MultiTenant	Class	Specifies multi-tenancy details for this class (DataNucleus extension).
@SoftDelete	Class	Specifies that this class will be "soft-deleted" upon deletion of objects (DataNucleus extension).

@ReadOnly

This DataNucleus-extension annotation is used to define a class as being read-only (equivalent as read-only="true"). Specified on the **class**.

```
@PersistenceCapable
@ReadOnly
public class MyClass
{
    ...
}
```

@MultiTenant

This DataNucleus-extension annotation is used specify multi-tenancy details for a class. Specified on the **class**.

Attribute	Type	Description	Default
column	String	Name of the multi-tenancy column for this class.	TENANT_ID
columnLength	int	Length of the multi-tenancy column.	
disabled	boolean	Whether the multi-tenancy for this class is disabled.	false

```
@PersistenceCapable
@MultiTenant(column="TENANT", columnLength=255)
public class MyClass
{
    ...
}
```


@SoftDelete

This DataNucleus-extension annotation is used to define a class as being soft-deleted whenever objects of this type are removed. Specified on the **class**.

Attribute	Type	Description	Default
column	String	Name of the soft-delete status column for this class.	DELETED

```
@PersistenceCapable
@SoftDelete
public class MyClass
{
    ...
}
```

DataNucleus Field-Level Extensions

The following annotations are specified at field/method-level and are vendor extensions providing more functionality than the JDO spec defines. Using these will reduce the portability of your application.

Annotation	Class/Field	Description
@SharedRelation	Field/Method	Specifies that the relation for this field/property is "shared" (DataNucleus extension).
@ReadOnly	Field/Method	Specifies that this field/property is "read-only" (DataNucleus extension).
@CreateTimestamp	Field/Method	Specifies that this field/property should store a creation timestamp when inserting (DataNucleus extension).
@UpdateTimestamp	Field/Method	Specifies that this field/property should store an update timestamp when updating (DataNucleus extension).

@SharedRelation

This DataNucleus-extension annotation is used to define a field with a (1-N/M-N) relation as being "shared" so that a distinguisher column is added. Specified on the **field/property**.

Attribute	Type	Description	Default
value	String	value to be stored in the distinguisher column for this relation field	
column	String	Name of the distinguisher column for this relation field	
primaryKey	boolean	Whether the distinguisher column should be part of the PK (when in a join table)	

```
@PersistenceCapable
public class MyClass
{
    @Persistent
    @Join
    @SharedRelation(column="ADDRESS_TYPE", value="home")
    Collection<Address> homeAddresses;

    @Persistent
    @Join
    @SharedRelation(column="ADDRESS_TYPE", value="work")
    Collection<Address> workAddresses;

    ...
}
```

@ReadOnly

This DataNucleus-extension annotation is used to define a field as being read-only (equivalent as insertable="false", updateable="false"). Specified on the **field/property**.

```
@PersistenceCapable
public class MyClass
{
    @Persistent
    @ReadOnly
    String someValue;

    ...
}
```

@CreateTimestamp

This DataNucleus-extension annotation is used to define this field as being persisted with a timestamp of the creation time of this object. Specified on the **field/property**.

```
@PersistenceCapable
public class MyClass
{
    @CreateTimestamp
    Timestamp createTime;

    ...
}
```

@UpdateTimestamp

This DataNucleus-extension annotation is used to define this field as being persisted with a timestamp of the update time of this object. Specified on the **field/property**.

```
@PersistenceCapable
public class MyClass
{
    @UpdateTimestamp
    Timestamp updateTime;

    ...
}
```

Meta-Annotations

JDO annotations are all usable as part of *meta-annotations*. A *meta-annotation* is, in simple terms, a user-defined annotation that provides one or multiple other annotations (including annotation attributes). Let's provide a couple of examples

Firstly, say we have

```
@PersistenceCapable(detachable="true")
@MultiTenant(column="TENANT")
```

and need to put this on many classes. We can introduce our own annotation

```
@Target(TYPE)
@Retention(RUNTIME)
@PersistenceCapable(detachable="true")
@MultiTenant(column="TENANT")
public @interface MultiTenantPersistable
{
}
```

so now we can simply annotate a JDO persistable class with

```
@MultiTenantPersistable
public class MyClass
{
    ...
}
```

A second example is where we are specifying several attributes on an annotation, such as

```
@PersistenceCapable(detachable="true", requiresExtent="true", cacheable="false",
identityType=IdentityType.DATASTORE)
```

so we introduce our own convenience annotation

```
@Target(TYPE)
@Retention(RUNTIME)
@PersistenceCapable(detachable="true", requiresExtent="true", cacheable="false",
identityType=IdentityType.DATASTORE)
public @interface MyPersistable
{
}
```

so now we can simply annotate a JDO persistable class with

```
@MyPersistable  
public class MyClass  
{  
    ...  
}
```



You can also make use of *meta-annotations* on fields/properties.