Repository File

by Thomas Mahler, Daren Drummond, Brian McCallister, Armin Waibel, Thomas Dudziak

1. Introduction - repository syntax

The syntax of the OJB repository xml files is defined by the *repository.dtd*.
The repository.dtd can be found here.

The actual repository metadta declaration is split up into several separate files, here is an excerpt of the most important files:

1. the repository.xml. Main file for metadata declaration. This file is split into several sub files using xml-Entity references.
2. the repository_database.xml. This file contains the mapping information for database/connection handling.
3. the repository_internal.xml. This file contains the mapping information for the OJB internal tables. These tables are used for implementing SequenceManagers and persistent collections.
4. the repository_user.xml. This file contains mappings for the tutorial applications and may be used to hold further user defined class mappings.
5. the repository_junit.xml. This file contains mapping information for common OJB JUnit regression test suite. In production environments these tables are not needed.
6. other repository_junit_XYZ.xml
   More specific junit test mapping. In production environments these tables are not needed.
7. There are some more files, for more information see comment in appropriate xml-file.

2. descriptor-repository

The *descriptor-repository* is the root element of a repository.xml file. It consists of one *jdbc-connection-descriptor* and at least one *class-descriptor* element.

2.1. Elements

```
<!ELEMENT descriptor-repository (documentation?, attribute*,
        jdbc-connection-descriptor*, class-descriptor*)>
```

The *documentation* element can be used to store arbitrary information.

The *attribute* element allows to add custom attributes, e.g. for passing arbitrary properties.

The *jdbc-connection-descriptor* element specifies a jdbc connection for the repository.

The *class-descriptor* element specify o/r mapping information for persistent class.

```
<!ELEMENT descriptor-repository (
    documentation?,
    attribute*,
    jdbc-connection-descriptor*,
    class-descriptor* )
>
```

2.2. Attributes

```
<!ATTLIST descriptor-repository
```

```
        version (1.0) #REQUIRED
        isolation-level (read-uncommitted | read-committed | repeatable-read |
                         serializable | optimistic) "read-uncommitted"
        proxy-prefetching-limit CDATA "50"
>
```

2.2.1. version

The *version* attribute is used to bind a repository.xml file to a given version of this dtd. A given OJB release will work properly only with the repository version shipped with that relase. This strictness maybe inconvenient but it does help to avoid the most common version conflicts.

2.2.2. isolation

The *isolation* attribute defines the default isolation level for *class-descriptor* that do not define a specific isolation level. This isolation level is used within the ODMG-api and does not touch the isolation-level off the database.

2.2.3. proxy-prefetching-limit

The *proxy-prefetching-limit* attribute specifies a default value to be applied to all proxy instances. If none is specified a default value of 50 is used. Proxy prefetching specifies how many instances of a proxied class should be loaded in a single query when the proxy is first accessed.

```
<!ATTLIST descriptor-repository
    version                 ( 1.0 ) #REQUIRED
    isolation-level         ( read-uncommitted |
                              read-committed   |
                              repeatable-read  |
                              serializable     |
                              optimistic ) "read-uncommitted"
    proxy-prefetching-limit CDATA "50"
>
```

3. jdbc-connection-descriptor

The *jdbc-connection-descriptor* element specifies a jdbc connection for the repository. It is allowed to define more than one *jdbc-connection-descriptor*. All *class-descriptor* elements are independent from the *jdbc-connection-descriptor*s. More info about connection handling here.

3.1. Elements

```
<!ELEMENT jdbc-connection-descriptor (documentation?, attribute*,
                object-cache?, connection-pool?, sequence-manager?)>
```

The *object-cache* element specifies the object-cache implementation class associated with this class.

A *connection-pool* element may be used to define connection pool properties for the specified JDBC connection.

Further a *sequence-manager* element may be used to define which sequence manager implementation should be used within the defined connection.

Use the *custom-attribute* element to pass implementation specific properties.

```
<!ELEMENT jdbc-connection-descriptor (
    documentation?,
    attribute*,
    object-cache?,
    connection-pool?,
    sequence-manager? )
>
```

3.2. Attributes

The *jdbc-connection-descriptor* element contains a bunch of required and implied attributes:

```
<!ATTLIST jdbc-connection-descriptor
        jcd-alias CDATA #REQUIRED
    default-connection (true | false) "false"
    platform (Db2 | Hsqldb | Informix | MsAccess | MsSQLServer |
                  MySQL | Oracle | PostgreSQL | Sybase | SybaseASE |
              SybaseASA | Sapdb | Firebird | Axion | NonstopSql |
              Oracle9i | MaxDB ) "Hsqldb"
        jdbc-level (1.0 | 2.0 | 3.0) "1.0"
        eager-release (true | false) "false"
    batch-mode (true | false) "false"
    useAutoCommit (0 | 1 | 2) "1"
    ignoreAutoCommitExceptions (true | false) "false"

        jndi-datasource-name CDATA #IMPLIED

        driver CDATA #IMPLIED
        protocol CDATA #IMPLIED
        subprotocol CDATA #IMPLIED
        dbalias CDATA #IMPLIED

        username CDATA #IMPLIED
        password CDATA #IMPLIED
>
```

3.2.1. jdbcAlias

The *jcdAlias* attribute is a shortcut name for the defined connection descriptor. OJB uses the jcd alias as key for the defined connections.

3.2.2. default-connection

The *default-connection* attribute used to define if this connection should used as default connection with OJB. You could define only one connection as default connection. It is also possible to set the default connection at runtime using *PersistenceBrokerFactory#setDefaultKey(...)* method. If set *true* you can use a PB-api shortcut-method of the *PersistenceBrokerFactory* to lookup PersistenceBroker instances.

Note:

If *default-connection* was not set at runtime, it is mandatory that *username and password* is set in repository file.

3.2.3. platform

The *platform* attribute is used to define the specific RDBMS Platform. This attribute corresponds to a *org.apache.ojb.broker.platforms.PlatformXXXImpl* class. Supported databases see here. Default was *Hsqldb*.

3.2.4. jdbc-level

The *jdbc-level* attribute is used to specify the Jdbc compliance level of the used Jdbc driver. Allowed values are: *1.0, 2.0, 3.0*. Default was *1.0*.

3.2.5. eager-release

The *eager-release* attribute was adopt to solve a problem occured when using OJB within JBoss (3.0 <= version < 3.2.2, seems to be fixed in jboss 3.2.2 and higher). Only use within JBoss. *DEPRECATED* attribute.

3.2.6. batch-mode

The *batch-mode* attribute allow to enable JDBC connection batch support (if supported by used database), 'true' value allows to enable per-session batch mode, whereas 'false' prohibits it. *PB.serviceConnectionManager.setBatchMode(...)* method can be used to switch on/off batch modus, if batch-mode was enabled. On PB.close() OJB switch off batch modus, thus you have to do '...setBatchMode(true)' on each obtained PB instance again.

3.2.7. useAutoCommit

The *useAutoCommit* attribute allow to set how OJB uses the autoCommit state of the used connections. The default mode is 1. When using mode 0 or 2 with the PB-api, you must use PB transaction demarcation.

- 0 - OJB ignores the autoCommit setting of the connection and does not try to change it. This mode could be helpful if the connection won't let you set the autoCommit state (e.g. using datasources within an application server).
- 1 - Set autoCommit explicitly to *true* when a connection was created and temporary set to *false* when necessary (default mode).
- 2 - Set autoCommit explicitly to *false* when a connection was created.

3.2.8. ignoreAutoCommitExceptions

If the *ignoreAutoCommitExceptions* attribute is set to *true*, all exceptions caused by setting autocommit state, will be ignored. Default mode is *false*.

3.2.9. jndi-datasource-name

If a *jndi-datasource-name* for JNDI based lookup of Jdbc connections is specified, the four attributes *driver*, *protocol*, *subprotocol*, and *dbalias* used for Jdbc DriverManager based construction of Jdbc Connections must not be declared.

3.2.10. username

The *username* and *password* attributes are used as credentials for obtaining a jdbc connections.
If users don't want to keep user/password information in the repository.xml file, they can pass user/password using a *PBKey* to obtain a PersistenceBroker. More info see FAQ.

4. connection-pool

The *connection-pool* element specifies the connection pooling parameter. More info about the connection handling can be found here.

```
<!ELEMENT connection-pool (documentation? )
>
```

Valid attributes for the *connection-pool* element are:

```
<!ATTLIST connection-pool
    maxActive                         CDATA #IMPLIED
    maxIdle                           CDATA #IMPLIED
    maxWait                           CDATA #IMPLIED
    minEvictableIdleTimeMillis        CDATA #IMPLIED
    numTestsPerEvictionRun            CDATA #IMPLIED
    testOnBorrow                  ( true | false ) #IMPLIED
    testOnReturn                  ( true | false ) #IMPLIED
    testWhileIdle                 ( true | false ) #IMPLIED
    timeBetweenEvictionRunsMillis CDATA #IMPLIED
    whenExhaustedAction           ( 0 | 1 | 2 ) #IMPLIED
    validationQuery               CDATA #IMPLIED
    logAbandoned                  ( true | false ) #IMPLIED
    removeAbandoned               ( true | false ) #IMPLIED
    removeAbandonedTimeout        CDATA #IMPLIED
>
```

maxActive is the maximum number of connections that can be borrowed from the pool at one time. When non-positive, there is no limit.

maxIdle controls the maximum number of connections that can sit idle in the pool at any time. When non-positive, there is no limit

maxWait - the maximum time block to get connection instance from pool, after that exception is thrown. When non-positive, block till last judgement

whenExhaustedAction

- 0 - fail when pool is exhausted
- 1 - block when pool is exhausted
- 2 - grow when pool is exhausted

testOnBorrow when *true* the pool will attempt to validate each object before it is returned from the pool.

testOnReturn set to *true* will force the pool to attempt to validate each object before it is returned to the pool.

testWhileIdle indicates whether or not idle objects should be validated. Objects that fail to validate will be dropped from the pool.

timeBetweenEvictionRunsMillis indicates how long the eviction thread should sleep before "runs" of examining idle objects. When non-positive, no eviction thread will be launched.

minEvictableIdleTimeMillis specifies the minimum amount of time that a connection may sit idle in the pool before it is eligable for eviction due to idle time. When non-positive, no connection will be dropped from the pool due to idle time alone (depends on *timeBetweenEvictionRunsMillis* > 0)

numTestsPerEvictionRun - the number of connections to examine during each run of the idle object evictor thread (if any)

validationQuery allows to specify a validation query used by the ConnectionFactory implementations using connection pooling, to test a requested connection (e.g. "select 1 from dual") before leave the pool (used by *ConnectionFactoryDBCPImpl* and *ConnectionFactoryPooledImpl*).
If not set, only *connection.isClosed()* will have been called before the connection was delivered.

logAbandoned is only supported when using *org.apache.ojb.broker.accesslayer.ConnectionFactoryDBCPImpl* ConnectionFactory implementation. Then it is a flag to log stack traces for application code which abandoned a Statement or Connection. Defaults to false. Logging of abandoned Statements and Connections adds overhead for every Connection open or new Statement because a stack trace has to be generated.
DEPRECATED attribute!

removeAbandoned and *removeAbandonedTimeout* When using *org.apache.ojb.broker.accesslayer.ConnectionFactoryDBCPImpl* ConnectionFactory implementation, the *removeAbandoned* flag controls the removal of abandoned connections if they exceed the *removeAbandonedTimeout*. Set to true or false, default false. If set to true a connection is considered abandoned and eligible for removal if it has been idle longer than the *removeAbandonedTimeout*. Setting this to true can recover db connections from poorly written applications which fail to close a connection.
DEPRECATED attributes!

5. sequence-manager

The *sequence-manager* element specifies the sequence manager implementation used for key generation. All sequence manager implementations shipped with OJB can be found in the *org.apache.ojb.broker.util.sequence* package. If no sequence manager is defined, OJB uses the default one. More info about sequence key generation here.

Use the *custom-attribute* element to pass implementation specific properties.

```
<!ELEMENT sequence-manager (
    documentation?,
    attribute* )
>
```

The *className* attribute represents the full qualified class name of the desired sequence manager implementation - it is mandatory when using the sequence-manager element. All sequence manager implementations you find will under *org.apache.ojb.broker.util.sequence* package named as *SequenceManagerXXXImpl*

More info about the usage of the Sequence Manager implementations can be found here.

```
<!ATTLIST sequence-manager
    className CDATA #REQUIRED
>
```

6. object-cache

The *object-cache* element can be used to specify the ObjectCache implementation used by OJB. There are three levels of declaration:

- in OJB.properties file, to declare the standard (default) ObjectCache implementation
- on jdbc-connection-descriptor level, to declare ObjectCache implementation on a per connection/user level
- on class-descriptor level, to declare ObjectCache implementation on a per class level

Note:

The priority of the declared object-cache elements are:
per class > per jdbc descriptor > standard

E.g. if you declare ObjectCache implementation 'my.cacheDef' as standard, set ObjectCache implementation 'my.cacheA' in class-descriptor for class A and class B does not declare an object-cache element. Then OJB use 'my.cacheA' as ObjectCache for class A and 'my.cacheDef' for class B.

```
<!ELEMENT object-cache (documentation?, attribute*)>
```

Use the *custom-attribute* element to pass implementation specific properties.

```
<!ATTLIST object-cache
    class  CDATA  #REQUIRED
>
```

Attribute 'class' specifies the full qualified class name of the used ObjectCache implementation.

7. custom attribute

An *attribute* element allows arbitrary name/value pairs to be represented in the repository. See the repository.dtd for details on which elements support it.

```
<!ELEMENT attribute EMPTY>
```

The *attribute-name* identifies the name of the attribute.

The *attribute-value* identifies the value of the attribute.

```
<!ATTLIST attribute
    attribute-name CDATA #REQUIRED
    attribute-value CDATA #REQUIRED
>
```

8. class-descriptor

For interfaces or abstract classes a *class-descriptor* holds a sequence of *extent-class* elements which specify the types extending this class.
Concrete base classes may specify a sequence of extent-class elements, naming the derived classes.

For concrete classes it must have *field-descriptor*s that describe primitive typed instance variables. References to other persistent entity classes are specified by *reference-descriptor* elements. Collections or arrays attributes that contain other persistent entity classes are specified by *collection-descriptor* elements
A class-descriptor may contain user defined custom attribute elements.

Use the *custom-attribute* element to pass implementation specific properties.

```
<!ELEMENT class-descriptor (
    (
        documentation?,
        extent-class+,
        attribute* ) |
    (
        documentation?,
        object-cache?,
        extent-class*,
        field-descriptor+,
        reference-descriptor*,
        collection-descriptor*,
        index-descriptor*,
        attribute*,
        insert-procedure?,
        update-procedure?,
        delete-procedure? )
    )
>
```

The *class* attribute contains the full qualified name of the specified class. As this attribute is of the XML type ID there can only be one *class-descriptor* per class.

The *isolation-level* attribute specifies the transactional isolation to be used for this class on ODMG-level.

Note:
The *isolation-level* does not touch the jdbc-connection isolation level. It's completely independend from the database connection setting.

If the *proxy* attribute is set, proxies are used for all loading operations of instances of this class. If set to *dynamic*, dynamic proxies are used. If set to another value this value is interpreted as the full-qualified name of the proxy class to use. More info about using of proxies here.

The *proxy-prefetching-limit* attribute specifies a limit to the number of elements loaded on a proxied reference. When the first proxied element is loaded, a number up to the proxy-prefetch-limit will be loaded in addition.

The *schema* attribute may contain the database schema owning the table mapped to this class.

The *table* attribute speciefies the table name this class is mapped to.

The *row-reader* attribute may contain a full qualified class name. This class will be used as the RowReader implementation used to materialize instances of the persistent class.

The *extends* attribute ************TODO: description*************

The *accept-locks* attribute specifies whether implicit locking should propagate to this class. Currently relevant for the ODMG layer only.

The optional *initialization-method* specifies a no-argument instance method that is invoked after reading an instance from a database row. It can be used to do initialization and validations.

The optional *factory-class* specifies a factory class that that is to be used instead of a no argument constructor when new objects are created. If the factory class is specified, then the *factory-method* also must be defined. It refers to a static no-argument method of the factory class that returns a new instance.

The *refresh* attribute can be set to *true* to force OJB to refresh instances when loaded from cache. Means all field values (except references) will be replaced by values retrieved from the database. It's set to *false* by default.

```
<!ATTLIST class-descriptor
    class ID #REQUIRED
    isolation-level (read-uncommitted | read-committed |
        repeatable-read | serializable | optimistic) "read-uncommitted"
    proxy CDATA #IMPLIED
    proxy-prefetching-limit CDATA #IMPLIED
    schema CDATA #IMPLIED
    table CDATA #IMPLIED
    row-reader CDATA #IMPLIED
    extends IDREF #IMPLIED
    accept-locks (true | false) "true"
    initialization-method CDATA #IMPLIED
    factory-class CDATA #IMPLIED
    factory-method CDATA #IMPLIED
    refresh (true | false) "false"
>
```

9. extent-class

An extent-class element is used to specify an implementing class or a derived class that belongs to the extent of all instances of the interface or base class.

```
<!ELEMENT extent-class EMPTY>
```

The *class-ref* attribute must contain a fully qualified classname and the repository file must contain a class-descriptor for this class.

```
<!ATTLIST extent-class
    class-ref IDREF #REQUIRED
>
```

10. field-descriptor

A field descriptor contains mapping info for a primitive typed attribute of a persistent class.
A field descriptor may contain custom attribute elements.

Use the *custom-attribute* element to pass implementation specific properties.

```
<!ELEMENT field-descriptor (documentation?, attribute*)>
```

The *id* attribute is optional. If not specified, OJB internally sorts field-descriptors according to their order of appearance in the repository file.
If a different sort order is intended the id attribute may be used to hold a unique number identifying the decriptors position in the sequence of field-descriptors.

Note:
The order of the numbers for the field-descriptors must correspond to the order of columns in the mapped table.

The *name* attribute holds the name of the persistent classes attribute. More info about persistent field handling.

The *table* attribute may specify a table different from the mapped table for the persistent class. (**currently not implemented**).

The *column* attribute specifies the column the persistent classes field is mapped to.

The *jdbc-type* attribute specifies the JDBC type of the column. If not specified OJB tries to identify the JDBC type by inspecting the Java attribute by reflection - OJB use the java/jdbc mapping desribed here.

The *primarykey* specifies if the column is a primary key column, default value is *false*.

The *nullable* attribute specifies if the column may contain null values.

The *indexed* attribute specifies if there is an index on this column

The *autoincrement* attribute specifies if the values for the persistent attribute should be automatically generated by OJB. More info about sequence key generation here.

The *sequence-name* attribute can be used to state explicitly a sequence name used by the sequence manager implementations. Check the javadocs of the used sequence manager implementation to get information if this is a mandatory attribute. OJB standard sequence manager implementations build a sequence name by its own, if the attribute was not set. More info about sequence key generation here.

The *locking* attribute is set to *true* if the persistent attribute is used for *optimistic locking*. More about optimistic locking. The default value is *false*.

The *updatelock* attribute is set to *false* if the persistent attribute is used for optimistic locking AND the dbms should update the lock column itself. The default is *true* which means that when locking is true then OJB will update the locking fields. Can only be set for TIMESTAMP and INTEGER columns.

The *default-fetch* attribute specifies whether the persistent attribute belongs to the JDO default fetch group.

The *conversion* attribute contains a fully qualified class name. This class must implement the interface `org.apache.ojb.accesslayer.conversions.FieldConversion`. A FieldConversion can be used to implement conversions between Java- attributes and database columns. More about field conversion.

The *length* attribute can be used to specify a length setting if required by the jdbc-type of the underlying database column.

The *precision* attribute can be used to specify a precision setting, if required by the jdbc-type of the underlying database column.

The *scale* attribute can be used to specify a sclae setting, if required by the jdbc-type of the underlying database column.

The *access* attribute specifies the accessibility of the field. Fields marked as *readonly* are not to modified. *readwrite* marks fields that may be read and written to. *anonymous* marks anonymous fields.
An anonymous field has a database representation (column) but no corresponding Java attribute. Hence the name of such a field does not refer to a Java attribute of the class, but is used as a unique identifier only. More info about anonymous keys here.

```
<!ATTLIST field-descriptor
    id CDATA #IMPLIED
    name CDATA #REQUIRED
    table CDATA #IMPLIED
    column CDATA #REQUIRED
    jdbc-type (BIT | TINYINT | SMALLINT | INTEGER | BIGINT | DOUBLE |
               FLOAT | REAL | NUMERIC | DECIMAL | CHAR | VARCHAR |
               LONGVARCHAR | DATE | TIME | TIMESTAMP | BINARY |
               VARBINARY | LONGVARBINARY | CLOB | BLOB) #REQUIRED
    primarykey (true | false) "false"
    nullable (true | false) "true"
    indexed (true | false) "false"
    autoincrement (true | false) "false"
    sequence-name CDATA #IMPLIED
    locking (true | false) "false"
    update-lock (true | false) "true"
    default-fetch (true | false) "false"
    conversion CDATA #IMPLIED
    length CDATA #IMPLIED
    precision CDATA #IMPLIED
    scale CDATA #IMPLIED
    access (readonly | readwrite | anonymous) "readwrite"
>
```

11. reference-descriptor

A reference-descriptor contains mapping info for an attribute of a persistent class that is not primitive but references another persistent entity Object. More about 1:1 references here.

A *foreignkey* element contains information on foreign key columns that implement the association on the database level.

```
<!ELEMENT reference-descriptor ( foreignkey+)>
```

The *name* attribute holds the name of the persistent classes attribute. Depending on the used PersistendField implementation, there must be e.g. an attribute in the persistent class with this name or a JavaBeans compliant property of this name.

The *class-ref* attribute contains a fully qualified class name. This class is the Object type of the persistent reference attribute. As this is an IDREF there must be a class-descriptor for this class in the repository too.

The *proxy* attribute can be set to *true* to specify that proxy based lazy loading should be used for this attribute.

The *proxy-prefetch-limit* attribute specifies a limit to the number of elements loaded on a proxied reference. When the first proxied element is loaded, a number up to the proxy-prefetch-limit will be loaded in addition.

The *refresh* attribute can be set to *true* to force OJB to refresh object references on instance loading.

Note:

This does not mean that all referenced objects will be read from database. It only means that the reference will be refreshed, the objects itself may provided by the cache. To refresh the objects set the *refresh* attribute of *class-descriptor*.

The *auto-retrieve* attribute specifies whether OJB automatically retrieves this reference attribute on loading the persistent object. If set to *false* the reference attribute is set to null. In this case the user is responsible to fill the reference attribute.
More info about auto-retrieve here.

The *auto-update* attribute specifies whether OJB automatically stores this reference attribute on storing the persistent object.
More info about the auto-XXX settings here.

Note:

This attribute must be set to false if using the OTM, ODMG or JDO layer.

The *auto-delete* attribute specifies whether OJB automatically deletes this reference attribute on deleting the persistent object.
More info about the auto-XXX settings here.

Note:

This attribute must be set to false if using the OTM, ODMG or JDO layer.

The *otm-dependent* attribute specifies whether the OTM layer automatically creates the referred object or deletes it if the reference field is set to null. Also otm-dependent references behave as if auto-update and auto-delete were set to true, but the auto-update and auto-delete attributes themself must be always set to false for use with OTM layer.

```
<!ATTLIST reference-descriptor
    name CDATA #REQUIRED
    class-ref IDREF #REQUIRED

    proxy (true | false) "false"
    proxy-prefetching-limit CDATA #IMPLIED
    refresh (true | false) "false"

    auto-retrieve (true | false) "true"
    auto-update (true | false) "false"
    auto-delete (true | false) "false"
    otm-dependent (true | false) "false"
>
```

12. foreignkey

A *foreignkey* element contains information on a foreign-key persistent attribute that implement the association on the database level.

```
<!ELEMENT foreignkey EMPTY>
```

The *field-ref* and *field-id-ref* attributes contain the name and the id attributes of the field-descriptor used as a foreign key.

Note:

Exactly one of these attributes must be specified.

```
<!ATTLIST foreignkey
    field-id-ref CDATA #IMPLIED
    field-ref CDATA #IMPLIED
>
```

13. collection-descriptor

A collection-descriptor contains mapping info for a liCollection- or Array-attribute of a persistent class that contains persistent entity Objects. See more about 1:n and m:n references.

The *inverse-foreignkey* elements contains information on foreign-key attributes that implement the association on the database level.

The *fk-pointing-to-this-class* and *fk-pointing-to-element-class* elements are only needed if the Collection or array implements a m:n association. In this case they contain information on the foreign-key columns of the intermediary table.

Use the *custom-attribute* element to pass implementation specific properties.

```
<!ELEMENT collection-descriptor (
    documentation?,
    orderby*,
    inverse-foreignkey*,
    fk-pointing-to-this-class*,
    fk-pointing-to-element-class*,
    attribute*)>
```

The *name* attribute holds the name of the persistent classes attribute. More info about persistent field handling.

The *collection-class* may hold a fully qualified class name. This class must be the Java type of the Collection attribute. This attribute must only specified if the attribute type is not a `java.util.Collection` (or subclass) or Array type. It is also possible to use non Collection or Array type user defined "collection" classes. More info see section manageable collection.

The *element-class-ref* attribute contains a fully qualified class name. This class is the Object type of the elements of persistent collection or Array attribute. As this is an IDREF there must be a class-descriptor for this class in the repository too.

The *orderby* attribute may specify a field of the element class. The Collection or Array will be sorted according to the specified attribute. The sort attribute may be used to specify ascending or descending order for this operation.

The *indirection-table* must specify the name of an intermediary table, if the persistent collection attribute implements a m:n association.

The *proxy* attribute can be set to true to specify that proxy based lazy loading should be used for this attribute. More about using proxy here.

The *proxy-prefetch-limit* attribute specifies a limit to the number of elements loaded on a proxied reference. When the first proxied element is loaded, a number up to the proxy-prefetch-limit will be loaded in addition.

The *refresh* attribute can be set to *true* to force OJB to refresh object references on instance loading.

Note:

This does not mean that all referenced objects will be read from database. It only means that the reference will be refreshed, the objects itself may provided by the cache. To refresh the objects use the *refresh* attribute in *class-descriptor*.

The *auto-retrieve* attribute specifies whether OJB automatically retrieves this reference attribute on loading the persistent object. If set to *false* the reference attribute is set to null. In this case the user is responsible to fill the reference attribute.
More info about auto-retrieve here.

The *auto-update* attribute specifies whether OJB automatically stores this reference attribute on storing the persistent object.
More info about the auto-XXX settings here.

Note:

This attribute must be set to false if using the OTM, ODMG or JDO layer.

The *auto-delete* attribute specifies whether OJB automatically deletes this reference attribute on deleting the persistent object.
More info about the auto-XXX settings here.

Note:

This attribute must be set to false if using the OTM, ODMG or JDO layer.

The *otm-dependent* attribute specifies whether the OTM layer automatically creates collection elements that were included into the collection, and deletes collection elements that were removed from the collection. Also otm-dependent references behave as if auto-update and auto-delete were set to true, but the auto-update and auto-delete attributes themself must be always set to false for use with OTM layer.

```
<!ATTLIST collection-descriptor
    name CDATA #IMPLIED
    collection-class CDATA #IMPLIED
    element-class-ref IDREF #REQUIRED
    orderby CDATA #IMPLIED
    sort (ASC | DESC) "ASC"

    indirection-table CDATA #IMPLIED

    proxy (true | false) "false"
    proxy-prefetching-limit CDATA #IMPLIED
    refresh (true | false) "false"
    auto-retrieve (true | false) "true"
    auto-update (true | false) "false"
    auto-delete (true | false) "false"
    otm-dependent (true | false) "false"
>
```

14. inverse-foreignkey

A *inverse-foreignkey* element contains information on a foreign-key persistent attribute that implement the association on the database level.

```
<!ELEMENT inverse-foreignkey EMPTY>
```

The *field-ref* and *field-id-ref* attributes contain the name and the id attributes of the field-descriptor used as a foreign key. Exactly one of these attributes must be specified.

```
<!ATTLIST inverse-foreignkey
    field-id-ref CDATA #IMPLIED
    field-ref CDATA #IMPLIED
>
```

15. fk-pointing-to-this-class

A *fk-pointing-to-this-class* element contains information on a foreign-key column of an intermediary table in a m:n scenario.

```
<!ELEMENT fk-pointing-to-this-class EMPTY>
```

The *column* attribute specifies the foreign-key column in the intermediary table that points to the class holding the collection.

```
<!ATTLIST fk-pointing-to-this-class
    column CDATA #REQUIRED
>
```

16. fk-pointing-to-element-class

A *fk-pointing-to-element-class* element contains information on a foreign-key column of an intermediary table in a m:n scenario.

```
<!ELEMENT fk-pointing-to-element-class EMPTY>
```

The *column* attribute specifies the foreign-key column in the intermediary table that points to the class of the collection elements.

```
<!ATTLIST fk-pointing-to-element-class
    column CDATA #REQUIRED
>
```

17. query-customizer

A query enhancer element to enhance the 1:n query, e.g. to modify the result objects of a query. More info about customizing collection queries.

Use the *custom-attribute* element to pass implementation specific properties.

```
<!ELEMENT query-customizer (
    documentation?,
    attribute*)>

<!ATTLIST query-customizer
    class CDATA #REQUIRED
>
```

18. index-descriptor

An *index-descriptor* describes an index by listing its columns. It may be unique or not.

```
<!ELEMENT index-descriptor (documentation?, index-column+)>

<!ATTLIST index-descriptor
    name CDATA #REQUIRED
    unique (true | false) "false">
```

19. index-column

An *index-column* is just the name of a column in an index.

```
<!ELEMENT index-column (documentation?)>

<!ATTLIST index-column
    name CDATA #REQUIRED>
```

20. Stored Procedure Support

OJB supports stored procedures for insert, update and delete operations. How to use stored procedures within OJB can be found here.

20.1. insert-procedure

Identifies the procedure/function that should be used to handle insertions for a specific class-descriptor.

The nested *argument* elements define the argument list for the procedure/function as well as the source for each argument.

Use the *custom-attribute* element to pass implementation specific properties.

```
<!ELEMENT insert-procedure
    (documentation?, (runtime-argument | constant-argument)?, attribute*)>
```

The *name* attribute identifies the name of the procedure/function to use

The *return-field-ref* identifies the field-descriptor that will receive the value that is returned by the procedure/function. If the procedure/ function does not include a return value, then do not specify a value for this attribute.

The *include-all-fields* attribute indicates if all field-descriptors in the corresponding class-descriptor are to be passed to the procedure/ function. If include-all-fields is 'true', any nested 'argument' elements will be ignored. In this case, values for all field-descriptors will be passed to the procedure/function. The order of values that are passed to the procedure/function will match the order of field-descriptors on the corresponding class-descriptor. If include-all-fields is false, then values will be passed to the procedure/function based on the information in the nested 'argument' elements.

```
<!ATTLIST insert-procedure
    name CDATA #REQUIRED
    return-field-ref CDATA #IMPLIED
    include-all-fields (true | false) "false"
>
```

20.2. update-procedure

Identifies the procedure/function that should be used to handle updates for a specific class-descriptor.

The nested *argument* elements define the argument list for the procedure/function as well as the source for each argument.

Use the *custom-attribute* element to pass implementation specific properties.

```
<!ELEMENT update-procedure
    (documentation?, (runtime-argument | constant-argument)?, attribute*)>
```

The *name* attribute identifies the name of the procedure/function to use

The *return-field-ref* identifies the field-descriptor that will receive the value that is returned by the procedure/function. If the procedure/ function does not include a return value, then do not specify a value for this attribute.

The *include-all-fields* attribute indicates if all field-descriptors in the corresponding class-descriptor are to be passed to the procedure/ function. If include-all-fields is 'true', any nested 'argument' elements will be ignored. In this case, values for all field-descriptors will be passed to the procedure/function. The order of values that are passed to the procedure/function will match the order of field-descriptors on the corresponding class-descriptor. If include-all-fields is false, then values will be passed to the procedure/function based on the information in the nested 'argument' elements.

```
<!ATTLIST update-procedure
    name CDATA #REQUIRED
    return-field-ref CDATA #IMPLIED
    include-all-fields (true | false) "false"
>
```

20.3. delete-procedure

Identifies the procedure/function that should be used to handle deletions for a specific class-descriptor.

The nested *runtime-argument* and *constant-argument* elements define the argument list for the procedure/function as well as the source for each argument.

Use the *custom-attribute* element to pass implementation specific properties.

```
<!ELEMENT delete-procedure
    (documentation?, (runtime-argument | constant-argument)?, attribute*)>
```

The *name* attribute identifies the name of the procedure/function to use

The *return-field-ref* identifies the field-descriptor that will receive the value that is returned by the procedure/function. If the procedure/ function does not include a return value, then do not specify a value for this attribute.

The *include-pk-only* attribute indicates if all field-descriptors in the corresponding class-descriptor that are identified as being part of the primary key are to be passed to the procedure/function. If include-pk-only is 'true', any nested 'argument' elements will be ignored. In this case, values for all field-descriptors that are identified as being part of the primary key will be passed to the procedure/function. The order of values that are passed to the procedure/function will match the order of field-descriptors on the corresponding class-descriptor. If include-pk-only is false, then values will be passed to the procedure/ function based on the information in the nested 'argument' elements.

```
<!ATTLIST delete-procedure
    name CDATA #REQUIRED
    return-field-ref CDATA #IMPLIED
    include-pk-only (true | false) "false"
>
```

20.4. runtime-argument

Defines an argument that is passed to a procedure/function. Each argument will be set to a value from a field-descriptor or null.

Use the *custom-attribute* element to pass implementation specific properties.

```
<!ELEMENT runtime-argument
    (documentation?, attribute*)>
```

The *field-ref* attribute identifies the field-descriptor in the corresponding class-descriptor that provides the value for this argument. If this attribute is unspecified, then this argument will be set to null.

```
<!ATTLIST runtime-argument
    field-ref CDATA #IMPLIED
    return (true | false) "false"
>
```

20.5. constant-argument

Defines a constant value that is passed to a procedure/function.

Use the *custom-attribute* element to pass implementation specific properties.

```
<!ELEMENT constant-argument
    (documentation?, attribute*)>
```

The *value* attribute identifies the value that is passed to the procedure/ function.

```
<!ATTLIST constant-argument
    value CDATA #REQUIRED
>
```