Sequence Manager

by Armin Waibel, Thomas Mahler, Ryan Vanderwerf et al., Andrew Clute

1. The OJB Sequence Manager

All sequence manager implementations you will find under the `org.apache.ojb.broker.util.sequence` package using the following naming convention `SequenceManagerXXXImpl`.

1.1. Automatical assignment of unique values

As mentioned in mapping tutorial OJB provides a mechanism to automatic assign unique values for primary key attributes. You just have to enable the `autoincrement` attribute in the respective *FieldDescriptor* of the XML repository file as follows:

```
<class-descriptor
  class="my.Article"
  table="ARTICLE"
>
    <field-descriptor
     name="articleId"
     column="ARTICLE_ID"
     jdbc-type="INTEGER"
     primarykey="true"
     autoincrement="true"
    />
    ....
</class-descriptor>
```

This definitions contains the following information:
The attribute `articleId` is mapped on the table's column `ARTICLE_ID`. The JDBC Type of this column is `INTEGER`. This is a primary key column and OJB shall automatically assign unique values to this attribute.

This mechanism works for all whole-numbered column types like BIGINT, INTEGER, SMALLINT,... and for CHAR, VARCHAR coliumns. This mechanism helps you to keep your business logic free from code that computes unique Ids for primary key attributes.

1.2. Force computation of unique values

By default OJB triggers the computation of unique ids during calls to PersistenceBroker.store(...). Sometimes it will be necessary to have the ids computed in advance, before a new persistent object was written to database. This can be done by simply obtaining the Identity of the respective object as follows:

```
Identity oid = broker.serviceIdentity().buildIdentity(Object newPersistentObject);
```

This creates an Identity object for the new persistent object and set all primary key values of the new persistent object - But it only works if `autoincrement` is enabled for the primary key fields.

Warning:

Force computation of unique values is not allowed when using *database based Identity columns* for primary key generation (e.g via Identity column supporting sequence manager), because the *real* PK value is at the earliest available after database insert operation. If you nevertheless force PK computing, OJB will use an temporary dummy PK value in the Identity object and this may lead to unexpeted behavior.

Info about lookup persistent objects by primary key fields see here.

1.3. How to change the sequence manager?

To enable a specific `SequenceManager` implementation declare an `sequence-manager` within the `jdbc-connection-descriptor` element in the repository file. If no `sequence-manager` was specified in the `jdbc-connection-descriptor`, OJB use a default sequence manager implementation (default was *SequenceManagerHighLowImpl*).

Further information you could find in the repository.dtd section sequence-manager element.

Example `jdbc-connection-descriptor` using `sequence-manager` tag:

```
<jdbc-connection-descriptor
        jcd-alias="farAway"
        platform="Hsqldb"
        jdbc-level="2.0"
        driver="org.hsqldb.jdbcDriver"
        protocol="jdbc"
        subprotocol="hsqldb"
        dbalias="../OJB_FarAway"
        username="sa"
        password=""
        batch-mode="false"
    >
        <connection-pool
            maxActive="5"
            whenExhaustedAction="0"
            validationQuery="select count(*) from OJB_HL_SEQ"
        />

        <sequence-manager className="org.apache.ojb.broker.util.
                                sequence.SequenceManagerHighLowImpl">
            <attribute attribute-name="grabSize" attribute-value="5"/>
            <attribute attribute-name="globalSequenceId"
                    attribute-value="false"/>
            <attribute attribute-name="globalSequenceStart"
                    attribute-value="10000"/>
        </sequence-manager>
</jdbc-connection-descriptor>
```

The mandatory `className` attribute needs the full-qualified class name of the desired sequence-manager implementation. If a implementation needs configuration properties you pass them using `attribute` tags with `attribute-name` represents the property name and `attribute-value` the property value. Each sequence manager implementation shows all properties on the according javadoc page.

1.4. SequenceManager implementations

Source code of all `SequenceManager` implementations can be found in `org.apache.ojb.broker.util.sequence` package.
If you still think something is missing you can just write your own sequence manager implementation.

1.4.1. High/Low sequence manager

Per default OJB internally uses a High/Low algorithm based sequence manager for the generation of unique ids, as described in Mapping Objects To Relational Databases.
This implementation is called `ojb.broker.util.sequence.SequenceManagerHighLowImpl` and is able to generate IDs unique to a given object and all extent objects declared in the objects class descriptor.
If you ask for an uid using an interface with several implementor classes, or a baseclass with several subclasses the returned uid have to be unique accross all tables representing objects of the extent in question (more see here).

It's also possible to use this implementation in a *global mode*, generate global unique id's.

```
<sequence-manager className=
    "org.apache.ojb.broker.util.sequence.SequenceManagerHighLowImpl">

    <attribute attribute-name="grabSize" attribute-value="20"/>
    <attribute attribute-name="globalSequenceId"
                                    attribute-value="false"/>
    <attribute attribute-name="globalSequenceStart"
                                    attribute-value="10000"/>
    <attribute attribute-name="autoNaming"
                                    attribute-value="true"/>
</sequence-manager>
```

With property `grabSize` you set the size of the assigned ids (default was 20).

If property `globalSequenceId` was set `true` you will get global unique ids over all persistent objects. Default was `false`.
The attribute `globalSequenceStart` define the start value of the global id generation (default was 10000).

This sequence manager implementation supports user defined *sequence-names* to manage the sequences. The attribute `autoNaming` define if sequence names should be build automatic if none found in `field-descriptor`.
If set 'true' OJB try to build a sequence name automatic if none found in field-descriptor and set this name as `sequence-name` in field-descriptor (see more). If set 'false' OJB throws an exception if none sequence name was found in field-descriptor (default was 'true').

Limitations:
- do **not** use in **managed environments** when connections were enlisted in running transactions, e.g. when using DataSources of an application server
- if set connection-pool attribute 'whenExhaustedAction' to 'block' (wait for connection if connection-pool is exhausted), under heavy load this sequence manager implementation can block application.
- superfluously to mention, do not use if other non-OJB applications insert objects too

1.4.2. In-Memory sequence manager

Another sequence manager implementation is a *In-Memory* version called `ojb.broker.util.sequence.SequenceManagerInMemoryImpl`.
Only the first time an uid was requested for a object, the manager query the database for the max value of the target column - all following request were performed in memory. This implementation ditto generate unique ids across all extents, using the same mechanism as the High/Low implementation.

```
<sequence-manager className="org.apache.ojb.broker.util.
                             sequence.SequenceManagerInMemoryImpl">
    <attribute attribute-name="autoNaming"
                                    attribute-value="true"/>
</sequence-manager>
```

For attribute `autoNaming` see

This sequence manager implementation supports user defined *sequence-names* to manage the sequences (see more) or if not set in `field-descriptor` it is done automatic.

This is the fastest standard sequence manager implementation, but has some Limitations:
- do not use in clustered environments
- superfluously to mention, do not use (or handle with care) if other non-OJB applications insert objects too

1.4.3. Database sequences based implementation

If your database support sequence key generation (e.g. Oracle, SAP DB, PostgreSQL) you could use the

`SequenceManagerNextValImpl` implementation let your database generate the requested ids.

```
<sequence-manager className="org.apache.ojb.broker.util.
                             sequence.SequenceManagerNextValImpl">
    <attribute attribute-name="autoNaming"
                                attribute-value="true"/>
</sequence-manager>
```

Attribute `autoNaming` default was 'true'. If set 'true' OJB try to build a sequence name automatic if none found in field-descriptor and set this generated name as `sequence-name` in field-descriptor.
If set 'false' OJB throws an exception if none sequence name was found in field-descriptor, ditto OJB does NOT try to create a database sequence entry when for given sequence name no database sequence could be found.

When using this sequence manager it is possible to define a **sequence-name** `field-descriptor` attribute in the repository file for each autoincrement/pk field. If you don't specify a sequence name, the sequence manager try to build a extent-aware sequence name on its own - except you set attribute `autoNaming` to 'false', then an exception will be thrown.
Keep in mind that in this case you are responsible to be aware of extents. Thus you have to use the same `sequence-name` attribute value for all extents, even if the extents were mapped to different database tables.
See usage of the `sequence-name` attribute:

```
<class-descriptor
      class="org.apache.ojb.broker.sequence.SMDatabaseSequence"
      table="SM_TAB_DATABASE_SEQUENCE"
    >
        <field-descriptor
         name="seqId"
         column="SEQ_ID"
         jdbc-type="INTEGER"
         primarykey="true"
         autoincrement="true"
         sequence-name="TEST_SEQUENCE"
        />

        ....
    </class-descriptor>
```

Limitations:
- none known

1.4.4. Database sequences based high/low implementation

Based on the sequence manager implementation described above, but use a high/low algorithm to avoid database access.

```
<sequence-manager className="org.apache.ojb.broker.util.
                         sequence.SequenceManagerSeqHiLoImpl">
<attribute attribute-name="grabSize" attribute-value="20"/>
<attribute attribute-name="autoNaming"
                                attribute-value="true"/>
</sequence-manager>
```

With the property `grabSize` you set the size of the assigned ids. For attribute `autoNaming` see.

This sequence manager implementation supports user defined *sequence-names* to manage the sequences (see more) or if not set in `field-descriptor` it is done automatic.

Limitations:
- superfluously to mention, do not use (or handle with care) if other non-OJB applications insert objects too

1.4.5. Oracle-style sequencing

(By Ryan Vanderwerf et al.) This solution will give those seeking an oracle-style sequence generator a final answer (Identity

columns really suck). If you are using multiple application servers in your environment, and your database does not support read locking like Microsoft SQL Server, this is the only safe way to guarantee unique keys (HighLowSequenceManager WILL give out duplicate keys, and corrupt your data).
The `SequenceManagerStoredProcedureImpl` implementation enabled database sequence key generation in a *Oracle-style* for all databases (e.g. MSSQL, MySQL, DB2, ...).
First add a new table `OJB_NEXTVAL_SEQ` to your database.

```
CREATE TABLE OJB_NEXTVAL_SEQ
(
    SEQ_NAME    VARCHAR(150) NOT NULL,
    MAX_KEY     INTEGER,
    CONSTRAINT SYS_PK_OJB_NEXTVAL PRIMARY KEY(SEQ_NAME)
)
```

You will also need a stored procedure called `ojb_nextval_proc` that will take care of giving you a guaranteed unique sequence number.
Here is an example for the stored procedure you need to use sequencing for MSSQL server:

```
CREATE PROCEDURE OJB_NEXTVAL_PROC
@SEQ_NAME varchar(150)
AS
declare @MAX_KEY BIGINT
-- return an error if sequence does not exist
-- so we will know if someone truncates the table
set @MAX_KEY = 0

UPDATE OJB_NEXTVAL_SEQ
SET    @MAX_KEY = MAX_KEY = MAX_KEY + 1
WHERE  SEQ_NAME = @SEQ_NAME

if @MAX_KEY = 0
select 1/0
else
select @MAX_KEY
RETURN @MAX_KEY
```

You have to adapt this script if MSSQL was not used (We are interested in scripts for other databases). Last, enable this sequence manager implementation:

```
<sequence-manager className="org.apache.ojb.broker.util.
                        sequence.SequenceManagerStoredProcedureImpl">
    <attribute attribute-name="autoNaming"
                                    attribute-value="true"/>
</sequence-manager>
```

For attribute *autoNaming* see .

This sequence manager implementation supports user defined *sequence-names* to manage the sequences (see more) or if not set in `field-descriptor` it is done automatic.

Limitations:
- currently none known

1.4.6. Microsoft SQL Server 'uniqueidentifier' type (GUID) sequencing

For those users you are using SQL Server 7.0 and up, the uniqueidentifier was introduced, and allows for your rows Primary Keys to be GUID's that are guaranteed to be unique in time and space.

However, this type is different than the Identity field type, whereas there is no way to programmatically retrieve the inserted value. Most implementations when using the u.i. field type set a default value of "newid()". The SequenceManagerMSSQLGuidImpl class manages this process for you as if it was any normal generated sequence/identity

field.

Assuming that your PK on your table is set to 'uniqueidentifier', your field-description would be the same as using any other SequenceManager:

```
<field-descriptor
         name="guid"
         column="document_file_guid"
         jdbc-type="VARCHAR"
         primarykey="true"
         autoincrement="true"
      />
```

Note that the jdbc-type is a VARCHAR, and thus the attribute (in this case 'guid') on your class should be a String (SQL Server does the conversion from the String representation to the binary representation when retrieved/set).

You also need to turn on the SequenceManager in your jdbc-connection-descriptor like this:

```
<sequence-manager
    className="org.apache.ojb.broker.util.sequence.SequenceManagerMSSQLGuidImpl"
/>
```

Limitations:
-This will only work with SQL Server 7.0 and higher as the uniqueidentifier type was not introduced until then.
This works well in situations where other applications might be updated the database as well, because it guarantees (well, as much as Microsoft can guarantee) that there will be no collisions between the Guids generated.

1.4.7. Identity based sequence manager

This sequence manager implementation supports database Identity columns (supported by MySQL, MsSQL, HSQL, ...). When using identity columns we have to do a trick to make the sequence manager work.
OJB identify each persistence capable object by a unique ojb-Identity object. These ojb-Identity objects were created using the sequence manager instance to get UID's. Often these ojb-Identity objects were created before the persistence capable object was written to database.
When using Identity columns it is not possible to retrieve the next valid UID before the object was written to database. As recently as the real object was written to database, you can ask the DB for the last generated UID. Thus in SequenceManagerNativeImpl we have to do a trick and use a 'temporary' UID till the object was written to database.
So, if it's possible try to avoid using Identity columns in your database model. If not use this sequence manager implementation to as a workaround for the Identity problem.

To enable this sequence manager implementation set in your `jdbc-connection-descriptor`:

```
<sequence-manager
    className="org.apache.ojb.broker.util.sequence.SequenceManagerNativeImpl">
</sequence-manager>
```

To declare the identity column in the repository.xml file add `primarykey="true"`, `autoincrement="true"` and `access="readonly"` to the field-descriptor for your table's primary key identity column.

```
<field-descriptor
        name="identifier"
        column="NATIVE_ID"
        jdbc-type="BIGINT"
        primarykey="true"
        autoincrement="true"
        access="readonly"/>
```

Limitations:
- The Identity columns have to **start with value >= 1** and should never be negative.
- Use of Identity columns is **not extent aware** (This may change in further versions). More info here.

1.5. How to write my own sequence manager?

Very easy to do, just write a implementation class of the interface `org.apache.ojb.broker.util.sequence.SequenceManager`. OJB use a factory (`SequenceManagerFactory`) to obtain sequence manager instances.

This Factory can be configured to generate instances of your specific implementation by adding a `sequence-manager` tag in the `jdbc-connection-descriptor`.

```
<sequence-manager className="my.SequenceManagerMYImpl">
</sequence-manager>
```

That's it!

If your sequence manager implementation was derived from `org.apache.ojb.broker.util.sequence.AbstractSequenceManager` it's easy to pass configuration properties to your implementation using `attribute` tags.

```
<sequence-manager className="my.SequenceManagerMYImpl">
<attribute attribute-name="myProperty" attribute-value="test"/>
</sequence-manager>
```

With

```
public String getConfigurationProperty(String key, String defaultValue)
```

method get the properties in your implementation class.

Note:
Of course we interested in your solutions! If you have implemented something interesting, just contact us.

1.6. Questions

1.6.1. When using sequence-name attribute in field-descriptor?

Most `SequenceManager` implementations based on sequence names. If you want retain control of sequencing use your own `sequence-name` attribute in the `field-descriptor`. In that case you are reponsible to use the same name across extents (see more info about extents and polymorphism). Per default the sequence manager build its own *extent aware* sequence name with an simple algorithm (see `org.apache.ojb.broker.util.sequence.SequenceManagerHelper#buildSequenceName`) if necessary. In most cases this should be sufficient. If you have a very complex data model and you will do many metadata changes in the repository file in future, then it could be better to explicit use `sequence-names` in the `field-descriptor`. See more avoid pitfals.

1.6.2. What to hell does extent aware mean?

Say we have a abstract base class `Animal` and two classes `Dog` and `Cat` which extend `Animal`. For each non-abstract class we create a separate database table.
We will be able to do a query like *give me all animals*. Thus the uid's of `Dog` and `Cat` objects must be unique across the tables of both classes or else you may not get a vaild query result.
The reason for this behaviour is the `org.apache.ojb.broker.Identity` class implementation (this may change in

further versions).

1.6.3. How could I prevent auto-build of the sequence-name?

All shipped `SequenceManager` implementations which using sequence names for UID generation, support by default auto-build (autoNaming) of the sequence name if none was found in the `field-descriptor`.
To prevent this, all relevant SM implementations support a `autoNaming` property - set via `attribute` element. If set `false` OJB doesn't try to build sequence names automatic.

```
<sequence-manager className="XYZ">
...
    <attribute attribute-name="autoNaming" attribute-value="true"/>
...
</sequence-manager>
```

1.6.4. Sequence manager handling using multiple databases

If you use multiple databases you have to declare a sequence manager in each `jdbc-connection-descriptor`. If you don't specify a sequence manager OJB use the default one (currently `ojb.broker.util.sequence.SequenceManagerHighLowImpl`).

1.6.5. One sequence manager with multiple databases?

OJB was intended to use a sequence manager per database. But it shouldn't be complicated to realize a global sequence manager solution by writing your own `SequenceManager` implementation.

1.6.6. Can I get direct access to the sequence manager?

That's no problem:

```
                    PersistenceBroker broker =
                    PersistenceBrokerFactory.createPersistenceBroker(myPBKey);
                    SequenceManager sm = broker.serviceSequenceManager();
                    ...
                    broker.close();
```

If you use `autoincrement=true` in your `field-descriptor`, there is no reason to obtain UID directly from the sequence manager or to handle UID in your object model.

Note:
Don't use SequenceManagerFactory#getSequenceManager(PersistenceBroker broker), this method returns a new sequence manager instance for the given broker instance and not the current used SM instance of the given PersistenceBroker instance]

1.6.7. Any known pitfalls?

- When enable a sequence manager implementation based on *sequence-name* attributes and if the name was not set as an attribute in the `field-descriptor` (see), an simple algorithm was used to build the sequence name.
  The algorithm try to get the top-level class of the field's enclosing class, if no top-level class was found, the table name of the field's enclosing class was used. If a top-level class was found, the first found extent class table name was used as sequence name.
  When using base classes/interfaces with extent classes based on different database tables and the `extent-class` entries in repository often change, the algorithm could be corrupted, because the first found extent class's table name could be change.
  To avoid this, remove the implementation internal sequence name entry (e.g. OJB_HL_SEQ table entry when using the Hi/Lo implementation, or remove the database sequence entry when using the 'Nextval' implementation) in that case, or use custom sequence name attributes in the field descriptor.