# Sequence Manager

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

## Table of contents

# 1. The OJB Sequence Manager

All sequence manager implementations shipped with OJB you can 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 *field-descriptor* 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 columns. This mechanism helps you to keep your business logic free from code that computes unique ID's 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* attribute 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 a *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="seq.start" attribute-value="10000"/>
        <attribute attribute-name="grabSize" attribute-value="20"/>
    </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 custom 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

The sequence manager implementation class `ojb.broker.util.sequence.SequenceManagerHighLowImpl` and is able to generate ID's unique to a given object and all *extent* objects declared in the objects class descriptor.
If you ask for an ID using an interface with several implementor classes, or a baseclass with several subclasses the returned ID have to be unique accross all tables representing objects of the interface or base class (more see here).
It's also possible to use this implementation in a *global mode*, generate global unique id's.

This implementation **needs an internal database table and object mapping declaration** to persist the used sequences. The table structure can be found in in platform guide the object metadata mapping can be found in OJB internal mapping file (called *repository_internal.xml*).

To declare this *sequence manager* implementation specify a `sequence-manager` element within the jdbc-connection-descriptor:

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

    <attribute attribute-name="seq.start" attribute-value="5000"/>
    <attribute attribute-name="grabSize" attribute-value="20"/>
    <attribute attribute-name="autoNaming" attribute-value="true"/>
    <attribute attribute-name="globalSequenceId" attribute-value="false"/>
    <!-- deprecated settings -->
    <attribute attribute-name="sequenceStart" attribute-value="5000"/>
</sequence-manager>
```

The property `seq.start` (or deprecated `sequenceStart`) define the start value of the id generation (default was '1'). It's recommended to use start values greater than '0' to avoid problems with primitive primary key fields when used as foreign key in references.

With property `grabSize` you set the size of the assigned ID's kept in memory for each *autoincrement* field. If the assigned ID's are exhausted a database call is made to lookup the next bunch of ID's (default *grabSize* is 20).
If OJB was shutdown/redeployed all unused assigned ID's are lost.

If property `globalSequenceId` was set `true` you will get global unique ID's over all persistent objects. Default was `false`.
NOTE: If the database is already populated or the global sequence name in *OJB_HL_SEQ* database table was removed (by accident), the *seq.start* value must be greater than the biggest PK value in database.

This sequence manager implementation supports *user defined sequence-names* as well as *automatic generated sequence-names* to manage the sequences - more about *sequence-names* here.
The attribute `autoNaming` can be used to enable *auto-generation* of *sequence-names*, default value is *true*.
More info about attribute *autoNaming* here.

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 ID's across all *extents*, using the same mechanism as the High/Low sequence manager implementation.

To declare this *sequence manager* implementation specify a `sequence-manager` element within the jdbc-connection-descriptor:

```
<sequence-manager
className="org.apache.ojb.broker.util.sequence.SequenceManagerInMemoryImpl">
    <attribute attribute-name="seq.start" attribute-value="0"/>
    <attribute attribute-name="autoNaming" attribute-value="true"/>
```

```
</sequence-manager>
```

The property `seq.start` (or deprecated `sequenceStart`) define the start value of the id generation (default was '1'). It's recommended to use start values greater than '0' to avoid problems with primitive primary key fields when used as foreign key in references.

This sequence manager implementation supports *user defined sequence-names* as well as *automatic generated sequence-names* to manage the sequences - more about sequence-names.
The attribute `autoNaming` can be used to enable *auto-generation* of *sequence-names*, default value is *true*.
More info about autoNaming.

The specified sequences will only be used *in memory*. First time a sequence was used OJB does a `select max`-query to find the latest/greatest value for the autoincrement field and use this as starting point for further *in-memory* key generation.

This is the fastest standard sequence manager implementation and should work with all databases without any specific preparation, but has some Limitations.

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
- only declare "number" fields as *autoincrement* fields (because e.g. "select max ... does not work with CHAR columns in the used manner)

### 1.4.3. Database sequences based implementation

If your database support sequence key generation (e.g. Oracle, SAP DB, PostgreSQL, ...) you can use the `SequenceManagerNextValImpl` implementation to force generation of the sequence keys by your database.

Database based sequences (sequence objects, sequence generators) are special (single-row) tables in the database created with an specific statement, e.g. `CREATE SEQUENCE sequenceName`.
This implementation use database based sequences to assign ID's in *autoincrement fields*.
The sequences can be managed by hand, by a database tool or by OJB. If the *autoNaming* attribute is enabled OJB creates sequences if needed. Also it's possible to declare *sequence names* in the field-descriptor

```
<class-descriptor
    class="org.greatest.software.Person"
    table="GS_PERSON"
>
    <field-descriptor
        name="seqId"
        column="SEQ_ID"
        jdbc-type="INTEGER"
        primarykey="true"
        autoincrement="true"
        sequence-name="PERSON_SEQUENCE"
    />

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

To declare this *sequence manager* implementation specify a `sequence-manager` element within the jdbc-connection-descriptor:

```
<sequence-manager
className="org.apache.ojb.broker.util.sequence.SequenceManagerNextValImpl">
    <attribute attribute-name="seq.start" attribute-value="200000"/>
```

```
    <attribute attribute-name="autoNaming" attribute-value="true"/>

    <!-- optional attributes supported by Oracle, PostgreSQL, MaxDB/SapDB, DB2
-->
    <!-- attribute attribute-name="seq.incrementBy" attribute-value="1"/ -->
    <!-- attribute attribute-name="seq.maxValue"
attribute-value="999999999999999999999999999"/ -->
    <!-- attribute attribute-name="seq.minValue" attribute-value="1"/ -->
    <!-- attribute attribute-name="seq.cycle" attribute-value="false"/ -->
    <!-- attribute attribute-name="seq.cache" attribute-value="20"/ -->
    <!-- attribute attribute-name="seq.order" attribute-value="false"/ -->
</sequence-manager>
```

Attribute `autoNaming`, default setting is *true*. If set *true* OJB will try to auto-generate a sequence name if none was found in field-descriptor's *sequence-name* attribute and create a database sequence if needed - more details see autoNaming section.
The auto-generated name will be set as *sequence-name* in the *field-descriptor*.
If set *false* OJB throws an exception if none *sequence-name* was found in *field-descriptor*, also OJB does NOT try to create a database sequence when for a given sequence name (specified in *field-descriptor*) no database sequence can be found.

The table below show additional *sequence properties*. To specifiy the properties use custom-attributes within the sequence-manager element.
The database sequence specific properties are generally speaking, see database user guide for detailed description.

| Name | Value | Description | Supported By |
|---|---|---|---|
| seq.as | database specific, e.g. *INTEGER* | Database sequence specific property. Specifies the datatype of the sequence, the allowed datatypes depend on the used database implementation. | DB2 |
| seq.start | 1 ... max INTEGER | Database sequence specific property. Specifies the first sequence number to be generated. Allowed: *1* or greater. | Oracle, PostgreSQL, MaxDB/SapDB, DB2 |
| seq.incrementBy | >=1 | Database sequence specific property. Specifies the interval between sequence numbers. This value can be any positive or negative integer, but it cannot be 0. *Decrement sequences are currently not supported* | Oracle, PostgreSQL, MaxDB/SapDB, DB2 |
| seq.maxValue | 1 ... max INTEGER | Database sequence specific property. Set max value for sequence numbers. | Oracle, PostgreSQL, MaxDB/SapDB, DB2 |
| seq.minValue | min INTEGER | Database sequence specific property. Set min value for | Oracle, PostgreSQL, MaxDB/SapDB, DB2 |

| | | | |
|---|---|---|---|
| | | sequence numbers. *Negative sequences are not tested as yet.* | |
| seq.cycle | true/false | Database sequence specific property. If *true*, specifies that the sequence continues to generate values after reaching either its maximum or minimum value. If *false*, specifies that the sequence cannot generate more values after reaching its maximum or minimum value. | Oracle, PostgreSQL, MaxDB/SapDB, DB2 |
| seq.cache | >= 2 | Database sequence specific property. Specifies how many values of the sequence Oracle preallocates and keeps in memory for faster access. Allowed values: *2* or greater. If set *0*, an explicite *nocache* expression will be set. | Oracle, PostgreSQL, MaxDB/SapDB, DB2 |
| seq.order | true/false | Database sequence specific property. If set *true*, guarantees that sequence numbers are generated in order of request. If *false*, a *no order* expression will be set. | Oracle, MaxDB/SapDB, DB2 |

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 property `grabSize` you set the size of the assigned ID's kept in memory for each *autoincrement* field. If the assigned ID's are exhausted a database call is made to lookup the next bunch of ID's using the next database sequence (default *grabSize* is 20).
If OJB was shutdown/redeployed all unused assigned ID's are lost.

**Note:**
Keep in mind that the database sequence value does not correspond with the used value in the *autoincrement*-field (table column value).

Attribute *autoNaming* is the same as for SequenceManagerNextValImpl.

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 when autoNaming is enabled.

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

### 1.4.5. Database Identity-column 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 for best compatibility try to avoid using *Identity columns* in your database model. If this is not possible, use this sequence manager implementation to work with database *Identity columns*.

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 persistent class mapping class-descriptor, add the following attributes to the primary key/identity key field-descriptor:
`primarykey="true"`, `autoincrement="true"` and **`access="readonly"`**
The first and second attributes are the same as all sequence manager implementations use to support autoincrement PK fields, the third one is mandatory for database *Identity columns* only.

```
<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 greater than '0'** and should never be negative.
- Use of Identity columns is **not extent aware** (This may change in further versions). More info here.

### 1.4.6. Stored Procedures based (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, ...).

To declare this *sequence manager* implementation specify a `sequence-manager` element within the jdbc-connection-descriptor:

```
<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 or if not set in `field-descriptor` it is done automatic when autoNaming is enabled.

- Add a new table `OJB_NEXTVAL_SEQ` to your database.
- You will also need a stored procedure called `ojb_nextval_proc` that will take care of giving you a guaranteed unique sequence number.

Below you can find the stored procedures you need to use sequencing for MSSQL server and Informix.
You have to adapt the scripts for other databases (We are interested in scripts for other databases).

Here you can find the currently supported databases and the statements to create the sql functions:

| Database | Table Statement | Stored Procedure |
|---|---|---|
| MSSQL | `DROP TABLE OJB_NEXTVAL_SEQ; CREATE TABLE OJB_NEXTVAL_SEQ ( SEQ_NAME VARCHAR(150) NOT NULL, MAX_KEY INTEGER, CONSTRAINT SYS_PK_OJB_NEXTVAL PRIMARY KEY(SEQ_NAME) );` | `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` |
| Informix | `drop table ojb_nextval_seq; create table ojb_nextval_seq ( seq_name varchar(250,0) not` | `create function ojb_nextval_proc (out arg1 int8, arg2 varchar(250,250)) returns int8; let arg1 = 0; update ojb_nextval_seq` |

| | | |
|---|---|---|
| | null,<br>max_key int8,<br>primary key(seq_name)<br>);<br> | set max_key = max_key + 1<br>where seq_name = arg2;<br>select max_key into arg1<br>from ojb_nextval_seq<br>where seq_name = arg2;<br>return arg1;<br>end function;<br> |
| Oracle | <br>TODO | <br>TODO |

Limitations:
- currently none known

### 1.4.7. 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.

## 1.5. The sequence-name attribute

Several SequenceManager implementations using *sequences* (synonyms: *sequence objects*, *sequence generators*) to manage the ID generation. Sequences are *entities* which generate unique ID's using e.g. database table per sequence, database row per sequence or an in-memory java-object.

To address the sequences, each *sequence* has an unique *sequence-name*.

In OJB the sequence-name of an autoincrement field is declared in a *sequence-name* attribute within the field-descriptor.

```
<class-descriptor
class="org.greatest.software.Person"
table="GS_PERSON"
>
<field-descriptor
    name="id"
    column="ID_PERSON"
    jdbc-type="INTEGER"
    primarykey="true"
    autoincrement="true"
    sequence-name="PERSON_SEQUENCE"
/>

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

The *sequence-name* attribute in the *field-descriptor* is only needed if the used sequence manager supports sequences, the field should be *autoincremented* and the auto-assign of a sequence-name is not desired.

**Note:**

Each *sequence-name* has be extent-aware.

If you don't specify a sequence name in the *field-descriptor* it is possible to auto-assign a sequence-name by OJB if autoNaming is supported by the used sequence manager implementation.

## 1.6. The autoNaming property

All shipped SequenceManager implementations using *sequences* for ID generation support a property called *autoNaming* which can be declared as a *custom attribute* within the *sequence-manager* element:

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

If set *true* OJB try to build a *sequence name* by it's own (a simple algorithm was used to auto-generate the sequence name - more details how it works in pitfalls section) and set this name as `sequence-name` in the field-descriptor of the autoincrement field if no sequence name is specified.
If set *false* the sequence manager throw an exception if a *sequence name* can't be found or was not declared in the *field-descriptor* of the autoincrement field. In this case OJB expects a valid *sequence-name* in the *field-descriptor*.
If the attribute `autoNaming` is set *false* the sequence manager never try to auto-generate a *sequence-name* (more detailed info here). If set *true* and a *sequence-name* is set in the *field-descriptor*, the *SequenceManager* will use this one and does **not** override the existing one.

The default setting is *true*.

## 1.7. 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 *custom attributes*.

```
<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.

## 1.8. Questions

### 1.8.1. 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, we call it extent-aware (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.

### 1.8.2. What to hell does extent aware mean?

Say we have an abstract base class `Animal` and two classes `Dog` and `Cat` which extend `Animal`. For each non-abstract class we create a separate database table and declare the inheritance in OJB. Now it is possible to do a query like *give me all animals* and OJB will return all `Cat` and `Dog` objects. To make this working in OJB the ID'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 (e.g. you can't query for the `Animal` with *id=23*, because in both tables such an id can exist).
The reason for this behaviour is the `org.apache.ojb.broker.Identity` class implementation (more details see javadoc/source of this class).

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

Keep in mind that user defined sequence names have to be extent-aware.

### 1.8.4. 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 a default one (currently `ojb.broker.util.sequence.SequenceManagerHighLowImpl`).

### 1.8.5. 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.

### 1.8.6. 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.
Except when using *user-defined* sequence manager implementations, in this case it could be needed.

**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]

### 1.8.7. Any known pitfalls?

- When using *sequences* based sequence manager implementations it's possible to enable auto-generation of *sequence names* - see autoNaming section. To build the *sequence name* an simple algorithm was used.
  The algorithm try to get the *top-level class* of the field's (the autoincrement *field-descriptor*) 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. The algorithm can be found in `org.apache.ojb.broker.util.sequence.SequenceManagerHelper#buildSequenceN`
  When using base classes/interfaces with *extent classes* (declared in the *class-descriptor*) based on different database tables and the *extent-class* entries in repository often change (e.g. add new top-level class, change top-level class), the algorithm could be corrupted after restart of OJB, because the first found extent class's table name could be change, hence the used sequence-name. Now the ID generation start over and could clash with existing ID's.
  To avoid this, remove the implementation specific internal sequence name entry (e.g. from table

OJB_HL_SEQ when using the Hi/Lo implementation, or remove the database sequence entry when using the 'Nextval' implementation) or use custom sequence name attributes in the field descriptor.