

Sequence Manager

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

Table of contents

1 The OJB Sequence Manager.....	3
1.1 Automatical assignment of unique values.....	3
1.2 Force computation of unique values.....	3
1.3 How to change the sequence manager?.....	4
1.4 SequenceManager implementations.....	5
1.4.1 High/Low sequence manager.....	5
1.4.2 In-Memory sequence manager.....	6
1.4.3 Database sequences based implementation.....	7
1.4.4 Database sequences based high/low implementation.....	8
1.4.5 Oracle-style sequencing.....	9
1.4.6 Microsoft SQL Server 'uniqueidentifier' type (GUID) sequencing.....	10
1.4.7 Identity based sequence manager.....	11
1.4.8 The sequence-name attribute.....	12
1.4.9 The autoNaming property.....	13
1.5 How to write my own sequence manager?.....	13
1.6 Questions.....	14
1.6.1 When using sequence-name attribute in field-descriptor?.....	14
1.6.2 What to hell does extent aware mean?	14
1.6.3 How could I prevent auto-build of the sequence-name?.....	15
1.6.4 Sequence manager handling using multiple databases.....	15
1.6.5 One sequence manager with multiple databases?.....	15
1.6.6 Can I get direct access to the sequence manager?.....	15

1.6.7 Any known pitfalls?.....	16
--------------------------------	----

1. The OJB Sequence Manager

All [sequence manager](http://api/org/apache/ojb/broker/util/sequence/SequenceManager.html) ([../api/org/apache/ojb/broker/util/sequence/SequenceManager.html](http://api/org/apache/ojb/broker/util/sequence/SequenceManager.html)) 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](http://docu/tutorials/mapping-tutorial.html) ([../docu/tutorials/mapping-tutorial.html](http://docu/tutorials/mapping-tutorial.html)) 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](http://docu/guides/repository.html#field-descriptor) ([../docu/guides/repository.html#field-descriptor](http://docu/guides/repository.html#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](http://api/org/apache/ojb/broker/Identity.html) ([../api/org/apache/ojb/broker/Identity.html](http://api/org/apache/ojb/broker/Identity.html)) 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](#) ([../docu/tutorials/pb-tutorial.html#find-by-pk](http://docu/tutorials/pb-tutorial.html#find-by-pk)) .

1.3. How to change the sequence manager?

To enable a specific [SequenceManager](#) ([../api/org/apache/ojb/broker/util/sequence/SequenceManager.html](http://api/org/apache/ojb/broker/util/sequence/SequenceManager.html)) implementation declare an *sequence-manager* attribute within the [jdbc-connection-descriptor](#) ([../docu/guides/repository.html#jdbc-connection-descriptor](http://docu/guides/repository.html#jdbc-connection-descriptor)) element in the [repository file](#) ([../docu/guides/repository.html](http://docu/guides/repository.html)) .

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](#) ([../repository.dtd.txt](http://repository.dtd.txt)) 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"
```

Sequence Manager

```
        validationQuery="select count(*) from OJB_HL_SEQ"
    />

    <sequence-manager
className="org.apache.obj.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 [custom attribute](#) ([../docu/guides/repository.html#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](#)

([../api/org/apache/obj/broker/util/sequence/SequenceManager.html](#)) implementations can be found in `org.apache.obj.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](#) ([../api/org/apache/obj/broker/util/sequence/SequenceManager.html](#)) implementation class

`obj.broker.util.sequence.SequenceManagerHighLowImpl` and is able to generate ID's unique to a given object and all [extent objects](#) ([../docu/guides/advanced-technique.html#extents](#)) 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 (see [OJB internal mapping](#) ([../docu/guides/repository.html#introduction](#)) for more info).

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

```
<attribute attribute-name="grabSize" attribute-value="20"/>
<attribute attribute-name="sequenceStart" attribute-value="0"/>
<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 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.

The attribute `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.

If property `globalSequenceId` was set `true` you will get global unique ID's over all persistent objects. Default was `false`.

The attribute `globalSequenceStart` define the start value of the global id generation (default was 10000). *This property is deprecated, please use property 'sequenceStart' instead.*

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

Sequence Manager

max value of the target column - all following request were performed in memory. This implementation ditto generate unique ID's across all [extents](#) ([../docu/guides/advanced-technique.html#extents](#)) , using the same mechanism as the High/Low sequence manager implementation.

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

The attribute `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.

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](#).

This is the fastest standard sequence manager implementation and should work with all databases without any specific preparation, 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 can use the `SequenceManagerNextValImpl` implementation to force generation of the sequence keys by your database.

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

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](#) (../docu/guides/repository.html#field-descriptor)

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](#) (../docu/guides/repository.html#field-descriptor) *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.

```
<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>
```

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.

Attribute [autoNaming](#) is the same as for [SequenceManagerNextValImpl](#).

Sequence Manager

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. 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 `objb_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.obj.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 when [autoNaming](#) is enabled.

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.obj.broker.util.sequence.SequenceManagerMSSQLGuidImpl"
```

Sequence Manager

```
/>
```

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](#) ([../api/org/apache/ojb/broker/Identity.html](#)). 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](#) ([../docu/guides/repository.html#class-descriptor](#)), add the following attributes to the primary key/identity key [field-descriptor](#) ([../docu/guides/repository.html#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](#).

1.4.8. The sequence-name attribute

Several [SequenceManager](#)

([../api/org/apache/obj/broker/util/sequence/SequenceManager.html](#)) 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](#) ([../docu/guides/repository.html#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 to be [extent-aware](#).

Sequence Manager

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.

1.4.9. The autoNaming property

All shipped [SequenceManager](#)

([../api/org/apache/ojb/broker/util/sequence/SequenceManager.html](#)) implementations using *sequences* for ID generation support a property called *autoNaming* which can be declared as a [custom attribute](#) ([../docu/guides/repository.html#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](#) ([../docu/guides/repository.html#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*.

1.5. 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.obj.broker.util.sequence.AbstractSequenceManager` it's easy to pass configuration properties to your implementation using [custom attributes](#) ([../docu/guides/repository.html#custom-attribute](#)) .

```
<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 responsible to use the same name across extents, we call it [extent-aware](#) (see more info about [extents and polymorphism](#) ([../docu/guides/advanced-technique.html#polymorphism](#))). Per default the sequence manager build its own *extent aware* sequence name with an simple algorithm (see `org.apache.obj.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 pitfalls](#).

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 and [declare the inheritance](#) ([../docu/guides/advanced-technique.html#polymorphism](#)) in OJB. Now it is possible to do a query like *give me all animals*. To make this working in OJB the

Sequence Manager

ID's of Dog and Cat objects must be unique across the tables of both classes or else you may not get a valid query result.

The reason for this behaviour is the org.apache.ojb.broker.Identity (../api/org/apache/ojb/broker/Identity.html) class implementation (more details see javadoc/source of this class).

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="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](#).

1.6.4. 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 a 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.getServiceSequenceManager();  
...  
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]

1.6.7. 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* a 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](#) ([../docu/guides/repository.html#class-descriptor](#)) table name was used as sequence name. The algorithm can be found in

`org.apache.objb.broker.util.sequence.SequenceManagerHelper#buildSequenceName`

When using base classes/interfaces with *extent classes* (declared in the [class-descriptor](#) ([../docu/guides/repository.html#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.