

# PB-api Guide

by Armin Waibel

## Table of contents

1 Introduction.....	2
2 How to access the PB-api?.....	2
3 Notes on Using the PersistenceBroker API.....	2
3.1 Exception Handling.....	2
3.2 Management of PersistenceBroker instances.....	2
3.3 Transactions.....	3
4 Questions.....	3
4.1 How to use multiple Databases.....	3
4.2 Hook into OJB - PB-Listener and Instance Callbacks.....	3

## 1. Introduction

The *PersistenceBroker API (PB-api)* provides the lowest level access to OJB's persistence engine. While it is a low-level API compared to the standardised ODMG or JDO API's it is still very straightforward to use.

The core class in the PersistenceBroker API is the [org.apache.ojb.broker.PersistenceBroker](http://org.apache.ojb.broker.PersistenceBroker) class. This class provides the point of access for all persistence operations in this API.

This document is not a [PB tutorial](#) (newbies please read the tutorial first) rather than a guide showing the specific usage and possible pitfalls in handling the PB-api.

If you don't find an answer for a specific question, please have a look at the [FAQ](#) and the other [reference guides](#).

## 2. How to access the PB-api?

The `org.apache.ojb.broker.PersistenceBrokerFactory` make several methods available:

```
public PersistenceBroker createPersistenceBroker(PBKey key) throws
PBFactoryException;

public PersistenceBroker createPersistenceBroker(String jcdAlias, String user,
String password)
    throws PBFactoryException;

public PersistenceBroker defaultPersistenceBroker() throws PBFactoryException;
```

Method `defaultPersistenceBroker()` can be used if the attribute [default-connection](#) is set *true* in *jdbc-connection-descriptor*. It's a convenience method, useful when only one database is used.

The standard way to lookup a broker instance is via `org.apache.ojb.broker.PBKey` by specify *jcdAlias* (defined in the *jdbc-connection-descriptor* of the [repository file or sub file](#)), *user* and *passwd*. If the user and password is already set in *jdbc-connection-descriptor* it is possible to lookup the broker instance only by specify the *jcdAlias* in `PBKey`:

```
PBKey pbKey = new PBKey("myJcdAliasName", "user", "password");
// alternative if user/passwd set in configuration file
PBKey pbKey = new PBKey("myJcdAliasName");
PersistenceBroker broker =
PersistenceBrokerFactory.createPersistenceBroker(pbKey);
```

See further in FAQ ["Needed to put user/password of database connection in repository file?"](#).

## 3. Notes on Using the PersistenceBroker API

### 3.1. Exception Handling

The exception handling is described in the PB-tutorial [exception handling section](#).

### 3.2. Management of PersistenceBroker instances

There is no need to cache or pool the used [PersistenceBroker](#) instances, because OJB itself use a PB-pool. The configuration of the PB-pool is adjustable in the [OJB.properties](#) file.

Using the `PersistenceBroker.close()` method releases the broker back to the pool under the default implementation. For this reason the examples in the [PB tutorial](#) all retrieve, use, and close a new broker for each logical transaction.

Apart from the pooling management `PersistenceBroker.close()` force the internal cleanup of the used broker instance - e.g. removing of temporary `PersistenceBrokerListener` instances, release of used connection if needed, internal used object registration lists, ... Therefore it's not recommended always refer to the same PB instance without closing it.

### 3.3. Transactions

Transactions in the PersistenceBroker API are *database level transactions*. This differs from *object level transactions* used by e.g. the [odmg-api](#). The broker does not maintain a collection of modified, created, or deleted objects until a commit is called -- it operates on the database using the databases transaction mechanism. If object level transactions are required, one of the higher level API's (ODMG, JDO, or OTM) should be used.

## 4. Questions

### 4.1. How to use multiple Databases

For each database define a [jdbc-connection-descriptor](#) same way as described in the [FAQ](#).

Now each database will be accessible via the `PersistenceBrokerFactory` using a `PBKey` matching the defined `jcdAlias` name as shown in section [How to access the PB-api?](#).

### 4.2. Hook into OJB - PB-Listener and Instance Callbacks

All *Listener* and instance callback interfaces supported by the *PB-api* can be used in the top-level API (like [ODMG-api](#)) as well.

The OJB Kernel supports three types of "hook" into OJB:

- [PersistenceBrokerAware](#)

A callback interface used by persistence capable objects (the object class is declared in OJB [metadata mapping](#)) to be aware of `PersistenceBroker` operations on itself. More detailed information can be found in the [Advanced-Technique Guide](#).

- [PBStateListener](#)

The listener interface for receiving `PersistenceBroker` state changes.

- [PBLifeCycleListener](#)

The listener interface for receiving *persistent object* life cycle information. This interface is intended for non persistent objects which want to track persistent object life cycle. Persistence capable objects can implement [PersistenceBrokerAware](#) - see above.

To add a *PBListener* use one of the following `PersistenceBroker` methods:

```
public void addListener(PBListener listener) throws PersistenceBrokerException;
public void addListener(PBListener listener, boolean permanent) throws PersistenceBrokerException;
```

The first method adds a **temporary** `org.apache.ojb.broker.PBListener` to the current `PersistenceBroker` instance - on `PersistenceBroker.close()` call the listener was removed.

The second one allows to add **permanent** `org.apache.obj.broker.PBListener` when the method argument is set *true*. If set *false* the listener only be temporary added.

**Note:**

Be carefully when adding **permanent** listener, keep in mind you don't know which instance was returned next time from the pool, with a permanent listener or without!

To guarantee that any pooled broker instance use the permanent listener, best way is to extend the used `org.apache.obj.broker.core.PersistenceBrokerFactoryIF` implementation and add the listener at creation of the [PersistenceBroker](#) instances.

Or add each time you lookup a `PersistenceBroker` instance the listener as a temporary listener.