



# Reference Documentation for db4o Java

**www.db4o.com**  
**version 7.3**

# Reference Documentation

Thanks for visiting the new db4o Reference Documentation! This reference section provides thorough and handy information that will let you master db4o.

To see the latest snapshot of this documentation, please, check the [online version](#). Other valuable documentation and educational resources are available in the [Resources](#) section of the db4o developer website.



## [Navigation Guide](#)

Use the Navigation Guide to have a quick look at the reference documentations contents.



## [Index](#)

Check the alphabetically sorted list of topics in this section.



## [Search](#)

Use the search box on the top right or click on "[More Search Options](#)" to do an advanced search.



## [Reference by Product](#)

[db4o](#) / [db4o Client-Server](#) / [db4o Replication Services](#) (dRS) / [Object Manager](#) (OM)



## [Documentation List](#)

Comprehensive list of db4o documents grouped by category.



## [Formula One Tutorial](#)

On-line db4o tutorial based on a Formula One domain (Cars, Pilots, etc).

# Usage Guidelines



## **Navigation**

For navigation, use the links located within the topic body, or below the body on most topics. In our online version you can also use the navigation located in the "bread-crumbs" above this topic. (just below the navigation bar). In the offline version a "navigation frame" is provided on the left (TOC).



## **Using Sample Code**

The "[Using Sample Code](#)" guide explains how to work with the code examples provided in this reference.



## **Environments**

The reference is offered for *Java* and *.NET* environments, providing examples in *Java*, *Visual Basic* and *C#*. The online version features "*filters*" on the right pane, which can help you to concentrate on a specific programming language.



## Topics

The information is presented in topics sets. Each of them is atomic and can be used in arbitrary order according to your current needs. The same applies to the code examples which can be downloaded and [tested in your environment](#).

---

More Reading:

- [Navigation Guide](#)
- [Using Sample Code](#)
- [Getting Started](#)
- [Product Philosophy](#)
- [Basic Concepts](#)
- [Object Lifecycle](#)
- [Implementation Strategies](#)
- [Platform Specific Issues](#)
- [Tuning](#)
- [Usage Pitfalls](#)
- [Client-Server](#)
- [db4o Replication System \(dRS\)](#)
- [Db4o Testing Framework](#)
- [Object Manager For db4o](#)
- [Working With Source Code](#)
- [License](#)
- [Contacts](#)

This revision (122) was last Modified 2008-01-21T03:57:26 by richard\_liang.

# Navigation Guide

The following list is a short guide to the document contents:

- For the product background, goals and positioning see [Product Philosophy](#).
- For an overview of the db4o basic concepts as an object database see [Basic Concepts](#). This collection of topics includes [object and relational database comparison](#), db4o implementation of [ACID Model](#), [concurrency control](#) and [Transaction](#). You will also find an [introduction to Native Queries](#) here.
- For installation instructions see [Getting Started](#).
- For information on how to open a database; store, retrieve, update and delete an objects see [Object Container](#) and [Simple Persistence](#).
- Persistence can be really effortless with [Transparent Persistence](#).
- See [Querying](#) to learn about different ways to query a database see. See [Query Modes](#) to control query speed and results delivery. See [Sorting Query Results](#) to add sorting functionality to your queries.
- See [Transaction](#) and [Concurrency Control](#) for information about transaction isolation level used and how it can be controlled.
- See [Working With Structured Objects](#), [Activation](#), [Update Depth](#) and [Delete Behavior](#) for information on structured objects (having other objects as fields) and collections.
- [Enhancement Tools](#) explains what enhancements are currently available and how they can be applied.
- See [Transparent Activation Framework](#) for information about Transparent Activation implementation and usage.
- [Object Construction](#), [Classloader issues](#) tells about the specifics of the objects construction from the database records. [Translators](#) and [Db4o Reflection API](#) explain how a user can take control of the way the objects are stored and restored from the database. [Server Without Persistent Classes Deployed](#) shows how db4o can be used when the stored classes definitions are not available. And finally [TypeHandlers](#) expose the internal kitchen of object persistence: take a control and write it yourself!
- Identity questions are discussed in [Object Identity](#) and [IDs and UUIDs](#).
- For information about how a db4o file is composed internally see [Freespace Management System](#).
- For the specifics of db4o implementations on different Java/.NET versions see respectively [db4o on Java Platforms](#), [db4o on .NET Platforms](#)
- For db4o usage in specific environments see:
  - [ASP.NET](#);
  - [Servlets](#);
  - [Symbian OS](#);
  - [Data Binding](#) (Windows Forms);
  - [Database For OSGi](#);
  - [Android](#)
  - [LINQ](#)

- Using different reporting systems with db4o is discussed in [Reporting](#).
- For db4o-XML converting issues see [Xml Import-Export In .NET](#)/ [Xml Import-Export In Java](#).
- For the issues dealing with specific types see: [Collections](#), [Blobs](#), [Static Fields And Enums](#), [Delegates And Events](#), [Final Fields](#). For the internal type representations see [Db4o meta-information](#).
- For information on how to update your class structure see [Refactoring and Schema Evolution](#).
- [Aliases](#) describe how to access persisted classes with aliased names or from different environment (Java classes from .NET and vice versa).
- [Tuning](#) discusses different tuning settings including configuration, debugging, diagnostics and optimization. One of the most important search optimization strategies is explained in [Indexing](#) and [Enable Field Indexes](#).
- Further to performance tuning learn [what influences CRUD operations](#) and use our [IO Benchmark Tools](#) to test your real application+device performance.
- [Maintenance](#) section explains [Defragment](#), [Backup](#) and [database version update](#).
- Java annotations and .NET attributes usage with db4o is explained in [Using Annotations](#) and [Using Attributes](#) accordingly.
- [Callbacks](#) and [Object Callbacks](#) sections explain the use of db4o callbacks (new and old version respectively). Topics include committed callbacks, referential integrity and autoincrement examples
- [IO Adapter](#) explains how to work with db4o pluggable IO adapter.
- [Embedded](#) and [Networked](#) topics explain the corresponding client/server setups.
- For working with several databases in CS mode see [Switching Database Files in CS Mode](#).
- [Remote Code Execution](#) will show you how to create code, which will be executed on the server only.
- [Messaging](#) explains the communication flow between a client and a server.
- For short description of the system allowing replication of data between db4o databases and db4o database and a relational database see [db4o Replication System \(dRS\)](#).
- To learn how to work with db4o tests and create your own see [Db4o Testing Framework](#).
- For an overview of graphical tool for browsing db4o files see [Object Manager For db4o](#).
- For db4o licensing information see [License](#).

This revision (40) was last Modified 2008-03-11T14:20:55 by Tetyana.

# Using Sample Code

## Contents

- [Variables](#)
- [Embedded Code](#)

Db4o reference documentation accompanies theoretical material with example code, which can be used for the material evaluation and better understanding.

The examples are provided in java, c# and VB.NET. In the online version, you can use filters on the right pane to display the relevant content. Offline version provides the content depending on the downloaded version of db4o.

Most of the examples are console applications except for those clearly mentioning the environment (web-applications, windows forms etc.).

Java examples are JDK5 compatible.

## Variables

---

The in-text code snippets are mainly used to show the syntax of a particular method. The following variables are used:

`objectContainer` - references and open ObjectContainer/IObjectContainer or ObjectClient/IObjectClient. For more information see [Object Container](#) and [Embedded](#).

`configuration` variable is used to reflect Configuration object. See [Configuration](#).

## Embedded Code

---

The examples usually show only the method code demonstrating the functionality discussed. In this case, additional code (opening and closing database methods, listing results, global variable definitions) and persistent class definitions are not visible. In order to see them you can download the whole example as a \*.zip file using the button in the top-right corner of the example code block. Note: usually the \*.zip file contains all the examples for the current topic.

In the majority of the examples, `objectContainer` is opened and closed in each method. This is done to keep examples atomic. In the real-world application, the best technique would be to open an `objectContainer` on the first request and keep it open for the application lifetime.

The downloaded code can be easily used to run the examples.

In Eclipse (using JDK5):

- unzip the contents in /Example/com/db4odoc/[example\_name] folder (check the correct package name in the code files);
- select File/New/Project/Java Project;
- choose the project name as "ExampleProject";
- select "Create project from existing source" and browse to the /Example folder;
- press "Next";
- in the "Libraries" tab add the required db4o libraries (db4o-x.x-java5.dll, db4o-x.x-nqopt.jar, db4o-x.x-db4ounit.jar, bloat.jar);
- press "Finish";
- run the example.

This revision (4) was last Modified 2007-11-28T15:27:54 by Tetyana.

# Getting Started

## Contents

- [Download Contents](#)
  - [Java Platform](#)
- [The db4o Engine](#)
- [Installation](#)
  - [Java Installation](#)

This topic will give you some start information about db4o and will help you to get your environment ready to work with db4o.

## Download Contents

---

### Java Platform

---

The db4o Java distribution comes as one zip file, db4o-X.X-java.zip. You will have to extract the contents to any folder before starting to use db4o.

#### **db4o-X.X/doc/reference**

contains reference documentation, which you are reading.

In addition to it you will find the following docs in your distribution:

#### **db4o-X.X/doc/tutorial/index.html**

This is the interactive "formula-1" HTML tutorial. Examples can be run "live" against a db4o database from within the browser. In order to use the interactive functionality a Java JRE 1.3 or above needs to be installed and integrated into the browser. Java security settings have to allow applets to be run.

It is recommended to take a first quick "drive" with "formula-1" before studying other db4o documents.

#### **db4o-X.X/doc/tutorial/db4o-X.X-tutorial.pdf**

The PDF version of the tutorial allows best fulltext search capabilities.

#### **db4o-X.X/doc/api/index.html**

The API documentation for db4o is supplied as JavaDocs HTML files. While you read through this documentation it may be helpful to look into the API documentation occasionally.

Additional online resources are available here: <http://developer.db4o.com/Resources>

## The db4o Engine

---

The java version of db4o object database engine consists of one single jar file. This is all that you need to program against. The versions supplied with the distribution can be found in db4o-X.X/lib/.

### db4o-X.X-java1.1.jar

will run with most Java JDKs that supply JDK 1.1.x functionality such as reflection and Exception handling. That includes many IBM J9 configurations, Symbian and Savaje.

### db4o-X.X-java1.2.jar

is built for all Java JDKs between 1.2 and 1.4.

### db4o-X.X-java5.jar

is built for Java JDK 5.

[Security Requirements On Java Platform](#) reviews db4o jar security permissions requirements.

## Installation

---

### Java Installation

---

If you add one of the above db4o-\*jar files to your CLASSPATH, db4o is installed. In case you work with an integrated development environment like [Eclipse](#) you would copy the db4o-\*jar to a /lib/ folder under your project and add db4o to your project as a library. (You only need to copy the one jar file for the distribution you are targeting.)

Here is how to add the db4o library to an Eclipse project

- create a folder named "lib" under your project directory, if it doesn't exist yet
- copy db4o-\*jar to this folder
- Right-click on your project in the Package Explorer and choose "refresh"
- Right-click on your project in the Package Explorer again and choose "properties"
- select "Java Build Path" in the treeview on the left
- select the "Libraries" tabpage.

## Getting Started

- click "Add Jar"
- the "lib" folder should appear below your project
- choose db4o-\*.jar in this folder
- hit OK twice

Please, note, that db4o can't be installed in JDK or JRE lib folder, the reasons are explained [further in the documentation](#).

This revision (33) was last Modified 2007-10-14T16:06:10 by Tetyana.

# Product Philosophy

db4o database is sponsored and supported by db4objects, a privately-held company based in San Mateo, California. db4objects was incorporated in 2004 under CEO [Christof Wittig](#), with the financial backing of top-tier Silicon Valley investors including [Mark Leslie](#), founding CEO of Veritas; [Vinod Khosla](#), founding CEO of Sun Microsystems; and [Jerry Fiddler](#), founding CEO of Wind River.

It is db4objects' mission to give developers a choice when it comes to object persistence and thus make their life a lot easier. db4o is designed to be a universal, affordable product platform, that is easy to learn and easy to implement. db4object's open source dual-license business model combines the power of the open source development community with servicing commercial customers' needs for product roadmap predictability, indemnification, single point of contact, and full tech support with fast response times.

db4objects has users and customers coming from 170 different countries, from Albania to Zimbabwe, and ranging from world class leaders like [Boeing](#), [Bosch](#), [Intel](#), [Ricoh](#), and [Seagate](#) to a wide range of highly innovative start-up companies.

More Reading:

- [Data Persistence](#)
- [OODBMS](#)
- [Db4o Position](#)
- [Why Choose Db4o](#)

This revision (2) was last Modified 2007-05-07T17:52:24 by Tetyana.

# Data Persistence

Software programs using different data persistence technologies are an integral part of contemporary informational space. More than often such systems are implemented with the help of object-oriented programming language (Java, c#, etc.) and a relational database management system (Oracle, MySQL, etc.). This implementation originally contains a mismatch between relational and object worlds, which is often called "object/relational impedance mismatch" (OR mismatch shortly). The essence of the problem is in the way the systems are designed. Object systems consist of objects and are characterized by identity, state, behavior, encapsulation. The relational model consists of tables, columns, rows and foreign keys and is described by relation, attribute, tuple, relation value and relation variable.

The object-relational mismatch has become enormously significant with the total adoption of OO technology. This resulted in the rapid development of so-called object-relational mappers (ORM), such as Hibernate or Toplink. This solution "cures" the symptoms of the OR mismatch by adding a layer into the software stack that automates the tedious task of linking objects to tables. However, this approach creates a huge drain on system performance, drives up software complexity, and increases the burden on software maintenance, thus resulting in higher cost of ownership. While the mapper solution may be feasible in large, administered datacenter environments, it is prohibitive in distributed and zero-administration architectures such as those required for embedded databases in client software, mobile devices, middleware or real-time systems.

This revision (1) was last Modified 2007-05-03T10:50:02 by Tetyana.

# OODBMS

The emergence of distributed data architectures - in networks, on clients and embedded in "smart" products such as cars or medical devices - is forcing companies in an array of industries to look beyond traditional RDBMS technology and ORM for an improved way to deal with object persistence. They are searching for a solution that can handle an enormous number of often complex objects, offers powerful replication and querying capabilities, reduces development and maintenance costs and requires minimum (zero) administration.

These requirements can be fulfilled by using an Object Oriented Database Management System (OODBMS). OODBMS provides an ideal match with object oriented environments like Java and .NET reducing the cost of development, support and versioning.

Using OODBMS in software projects also better supports modern Agile software engineering practices like:

- continuous refactoring;
- agile modeling;
- continuous regression testing;
- configuration management;
- developer "sandboxes".

Object-oriented database technology was first introduced in the early 1990s with great fanfare. It was wrongly positioned, at the time, as a replacement for RDBMS technology, and consequently failed to fulfill that expectation. However, it is now clear that these two database technologies are complimentary, each with its own strengths and weaknesses, and thus with its own applicability domain. While, RDBMS technologies are very successful in large Enterprise environments, where data has static and relatively flat structure, and where reporting and high volume data analysis are required, OODBMS are recognized as superior for embedded (invisible to the end user) applications, where the data has a complex and frequently changing structure and/or where zero administration is often a requirement.

Additionally, most OODBMS implementations suffered technical deficiencies that obscured the true strengths inherent in the technology, such as Native Queries support (i.e. using the syntax and semantics of popular OO languages for querying), and hence couldn't realize the true benefits of a uniform object-oriented software development environment for both the application and for object persistence. The early OODBMS products also suffered from a lack of an appropriate market focus and business model.

Consequently, nearly all OODBMS market entrants have since gone through market consolidation and have become focused on "vertical" target niches, such as defense and healthcare. These proprietary object-oriented databases tend to be expensive and usually require heavy vendor support for their complicated, proprietary interfaces. On average, 70% of these companies' revenues are derived from professional services rather than licensing. Among these are Versant (VSNT, consolidated with POET); ObjectStore, now the Real-Time division of Progress (PRGS); and Objectivity. As a result, most IT decision makers have ceased to consider OODBMS technology as a mainstream option for their

deployment projects.

(For more information about OODBMS technology refer to the [ODBMS.ORG website](http://www.odbms.org).)

This revision (1) was last Modified 2007-05-03T10:51:13 by Tetyana.

# Db4o Position

## Contents

- [Open Source](#)
- [Success Drivers](#)

db4o came to the market in 2004 with a goal to become the mainstream persistence architecture for embedded applications (in which the database is invisible to the end user) in general, and for mobile and embedded devices running on Java or .NET, in particular. db4o vision as a company (db4objects) is to become the affordable, dominant Persistence solution of choice in a market now overrun with hundreds of small vertical niche vendors offering predominantly unsuitable, proprietary pre-relational or relational technology at exorbitant prices.

db4o has achieved in a very short time, mainstream adoption in a fast growing user community currently counting over 20,000 members, due to its efficient innovative technology, its Native Queries deployment in Java and .NET environments and its open source dual licensing business model. .

The target environments for db4o are persistence architectures where there is no database administrator present and no RDBMS legacy, i.e. primarily on [equipment](#), [mobile](#) and [desktop](#) clients, and in the middleware. Typical industries of db4o customers include [transportation](#), communication, [automation](#), [medical sciences](#), [industrial](#), [consumer](#) and financial applications, among many others. Existing customers range from world-class leaders like [Boeing](#), [Bosch](#), [Intel](#), [Ricoh](#), and [Seagate](#) to a broad range of highly innovative start-up companies - in the Americas, EMEA, and Asia-Pacific.

As a client-side, embeddable database, db4o is particularly suited to be deployed in devices with embedded software.

## Open Source

---

db4objects uses the now-established, open source dual license business model as pioneered by MySQL, one of the world's most popular relational databases. In this model, db4o is available as open source under the [GPL](#) and the [dOCL](#), and as a commercial product under the commercial license. Any developer wishing to use the software in an open source product that falls under the GPL or other open-source licenses (Apache, LGPL, BSD, EPL as specified by the [dOCL](#)) can use the free open source version. Those developers wishing to embed db4o into a for-profit product can choose the affordable commercial runtime license. Other uses and licenses including those for evaluation, development, and academic application remain free under the GPL, creating a large and lively community around the product at a very low cost to the vendor.

## Success Drivers

---

Open Source platform usage is one of the key factors of db4o success. db4o's openness attracted a vast (20,000 and counting) community of users and contributors. Through the community support db4o gets broad and immediate testing, receives constructive suggestions (from the users actually looking into the code) and invaluable peer exchange of experiences - positive and negative.

Another factor to db4o success is the technology used. As a new-generation object database, native to both Java and .NET, db4o eliminates the traditional trade-off between performance and object-orientation. Recent PolePosition benchmark results show that db4o outperforms object-relational mappers by orders of magnitude, up to 44x in use cases with complex object models.

db4o uniquely offers object persistence with zero-administration, cross-platform applicability to Java and .NET, object-oriented querying, replication and browsing capabilities, and a small footprint. Its single library (JAR/DLL) is easily deployed and runs in the same memory process as the application, making it a fully integrated and tunable portion of the developers application.

Customers, analysts, and experts agree that the db4o object database is one of the world's best and most popular choices, because it stores and retrieves objects natively and not only eliminates the overhead and resource consumption of an ORM, but also greatly reduces the product development and maintenance costs, resulting in a lean, fast and easily integratable into an OO development environment persistence solution, far superior in many cases to that of any RDBMS.

This revision (4) was last Modified 2007-05-03T10:55:41 by Tetyana.

# Why Choose Db4o

There are many advantages of using "native" object technology over RDBMS or RDBMS paired with an OR mapper, and these technological advantages significantly impact an organization's competitiveness and bottom line.

First, object databases not only simplify development by eliminating the resource-consuming OR-mismatch entirely, but they also foster more sophisticated and differentiated product development through gains in flexibility and productivity brought on by "true" object-orientation.

db4o's ground-breaking object-oriented replication technology solves problems arising from distributed data architectures. Partially connected devices need to efficiently replicate data with peers or servers. The challenge lies in the creation of "smart" conflict resolution algorithms, when redundant data sets are simultaneously modified and need to be merged. With db4o's OO approach, developers can build smarter and easier synchronization conflict resolution and embed the necessary business logic into the data layer, rather than into the middle-tier or application layer. This creates "smart" objects that can be stored in a distributed fashion, but easily consolidated, as the object itself knows how to resolve synchronization conflicts. It also enables db4o solution on a client to synchronize data with an RDBMS backend server.

As a result, developers can now more consistently persist data on distributed, partially connected clients than ever before, while decreasing bandwidth requirements and increasing the responsiveness and reach of their mobile solutions or smart devices to make products more competitive in the marketplace.

Second, with an object database, the object scheme is the same as the data model. Developers can easier update their models to meet changing requirements, or for purposes of debugging or refactoring. db4o lets developers work with object structures almost as if they were "in-memory" structures. Little additional coding is required to manage object persistence. As a result, companies can add new features to their products faster to stay ahead of the competition.

Third, developers can now use object-oriented and entirely native approaches when it comes to querying, since db4o was the first in the industry to provide Native Queries (NQ) with its Version 5.0 launched in November 2005. Db4o Native Queries provide an API that uses the programming languages Java or .NET to access the database. No time is spent on learning a separate data manipulation or query language. Unlike incumbent, string-based, non-native APIs (such as SQL, OQL, JDOQL and others) Native Queries are 100% type-safe, 100% refactorable and 100% object-oriented, boosting developer productivity by up to 30%. In addition, the sophisticated modern programming development environments can be used to simplify the development and maintenance work even further.

Fourth, db4o also allows for more complex object models than its relational or non-native counterparts do. As the persistence requirements become more complex, db4o's unique design easily handles (or absorbs) the added complexity, so developers can continue to work as though new complexity were never introduced. Complexity means not only taller object trees and extensive use of inheritance, but also dynamically evolving object models, most extremely if development is taking place under runtime conditions (which makes db4o a leading choice for biotech simulation software, for instance). db4o could

be referred to as "agnostic to complexity," because it can automatically handle changes to the data model, without requiring extra work. No type or amount of complexity will change its behavior or restrain its capabilities, as is the case with RDBMS or non-native technology. With db4o breaking through this complexity, developers are able to write more user friendly and business-appropriate software components without incurring such high costs and modify them as needed, throughout the life cycle of the product with the same low cost.

In sum, db4o's native, cross-platform OO architecture enables its users to build more competitive products with faster update cycles, more natural object models that match more realistically their use cases, and more distributed data architectures to increase the reach of products. db4o is clearly more flexible and powerful for embedded DB applications than any non-native OODBMS or RDBMS technologies available.

For more information see our ["Choose db4o" presentation](#)

This revision (2) was last Modified 2007-07-14T00:02:30 by German Viscuso.

# Basic Concepts

This topic collection discusses concepts used in the foundation of the db4o system.

db4o is an object-oriented database. It provides all the benefits of an OO environment including data abstraction, inheritance, encapsulation. The object model simplifies maintenance and refactoring and seamlessly integrates with the modern programming languages (Java, .NET). One of the valuable benefits of the OO technology is Native Queries - database queries expressed in a native programming language.

In .NET version 3.5 the preferred alternative to Native Queries is LINQ - language integrates queries. Though LINQ supports relational databases as well, in db4o case querying does not include Object-Relational mapping which is more performant and productive.

In the same time db4o provides important database features like ACID transacrions and concurrency control.

These and other technologies will be discussed in more details below (work in progress).

More Reading:

- [ACID Model](#)
- [Concurrency Control And Locking](#)
- [Object Identity](#)
- [Transaction](#)
- [Database Models](#)
- [Native Query Concepts](#)

This revision (7) was last Modified 2008-02-18T19:29:12 by Tetyana.

# ACID Model

The ACID model is one of the oldest and most important concepts of database theory. It sets out the requirements for the database reliability: atomicity, consistency, isolation and durability. Without these requirements met, a database cannot be considered reliable.

Let's have a closer look at each of its components:

More Reading:

- [Atomicity](#)
- [Consistency](#)
- [Isolation](#)
- [Durability](#)
- [ACID Properties For Db4o](#)
- [Isolation Level For Db4o](#)

This revision (1) was last Modified 2007-05-22T08:16:21 by Tetyana.

# Atomicity

Atomicity states the mode, in which either all or no modifications are written to the database. In this mode each transaction is said to be "atomic". If any part of the transaction fails, the whole transaction will fail too. For example, in a bank transfer transaction there are 2 parts: debit and credit. If the debit operation was successful, but the credit failed, the whole transaction should fail and the system should remain in the initial state.

Some of the atomicity features:

- A transaction is a unit of operation - either all the transaction's actions are completed or none are
- atomicity is maintained in the presence of deadlocks
- atomicity is maintained in the presence of database software failures
- atomicity is maintained in the presence of application software failures
- atomicity is maintained in the presence of operation system failures
- atomicity is maintained in the presence of CPU failures
- atomicity is maintained in the presence of disk failures

This revision (1) was last Modified 2007-05-22T08:18:41 by Tetyana.

# Consistency

Consistency imposes a rule that only valid data can be written to a database. When an update is executed, the database state will change only if all the database consistency rules are obeyed, otherwise the transaction will be rolled back and the database restored to its consistent state. On the other hand, if all the updates are consistent, the database will be taken to a new consistent state.

Consistency rules can include: data type correctness, relation correctness, uniqueness of data, etc.

This revision (1) was last Modified 2007-05-22T08:20:00 by Tetyana.

# Isolation

Isolation imposes rules, which ensure that transactions do not interfere with each other even if they are executed at the same time. If 2 (or more) transactions are executed at the same time, they must be executed in a way so that transaction N won't be impacted by the intermediate data of transaction M. Note, that isolation does not dictate the order of the transactions. Another important thing to understand about isolation is serializability of transactions. If the effect on the database is the same when transactions are executed concurrently or when their execution is interleaved, these transactions are called serializable.

There are several degrees of isolation to be distinguished:

- degree 0: a transaction does not overwrite data updated by another user or process ("dirty data") of other transactions;
- degree 1: degree 0 plus a transaction does not commit any writes until it completes all its writes (until the end of transaction);
- degree 2: degree 1 plus a transaction does not read dirty data from other transactions;
- degree 3: degree 2 plus other transactions do not dirty data read by a transaction before the transaction commits.

The more common classification is by isolation levels:

## 1. Serializable

In this case, transactions are executed serially so that there is no concurrent data access. Transactions can also be executed concurrently but only when the illusion of serial transactions is maintained (i.e. no concurrent access to data occurs). If the system uses locks, a lock should be obtained over the whole range of selected data ("WHERE" clause in SQL). If the system does not use locks, no lock is acquired; however, if the system detects a concurrent transaction in progress, which would violate the serializability illusion, it must force that transaction to rollback, and the application will have to restart the transaction.

## 2. Repeatable Read

In this case, a lock is acquired over all the data retrieved from a database. Phantom reads can occur (i.e. new data from the other committed transactions included in the result)

## 3. Read Committed

In this case, read locks are acquired on the result set, but released immediately. Write locks are acquired and released only at the end of the transaction. Non-repeatable reads can occur, i.e. deletions or modifications from the other committed transactions will be visible by the current transaction. Phantom reads are also possible.

#### 4. Read Uncommitted

With this isolation level dirty reads are allowed. Uncommitted modifications from the other transactions are visible. Both phantom and nonrepeatable reads can occur.

This revision (1) was last Modified 2007-05-22T11:12:33 by Tetyana.

# Durability

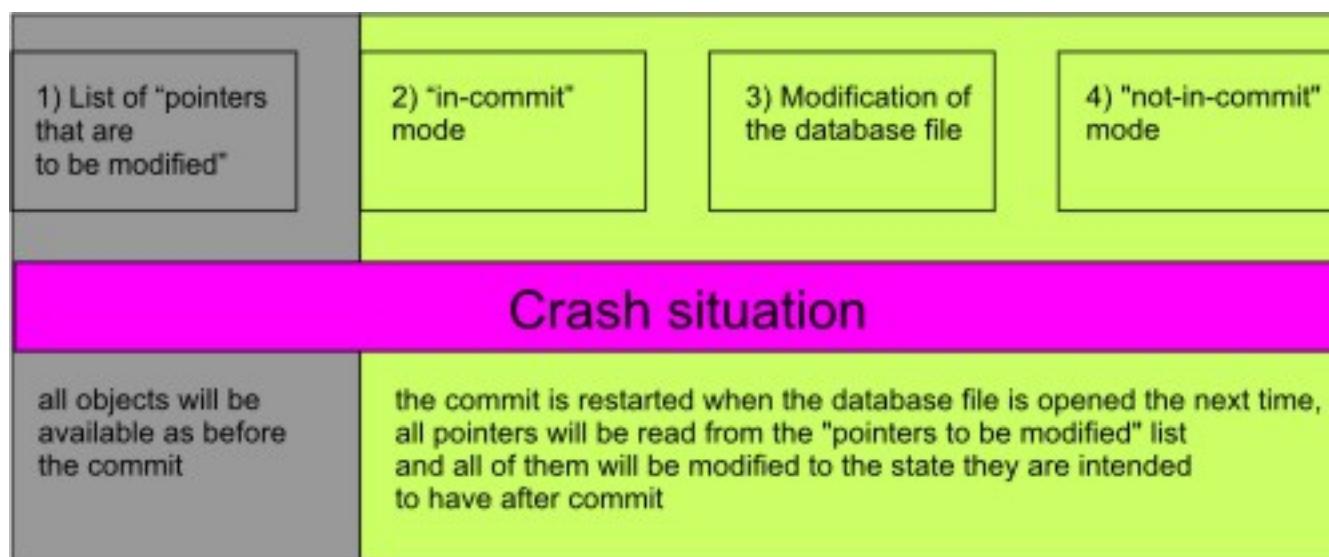
Durability property ensures that any committed transaction won't be lost. This is done by using database backups and transaction logs. Even in a case of a system or hardware failure, all the committed data should be restorable.

This revision (1) was last Modified 2007-05-22T11:14:18 by Tetyana.

# ACID Properties For Db4o

As any reliable database system db4o ensures ACID transactions. When a commit is executed, the following order of disc writes is ensured:

1. a list of "pointers that are to be modified" is written to the database file;
2. the database file is switched into "in-commit" mode;
3. the pointers are actually modified in the database file;
4. the database file is switched to "not-in-commit" mode.



As you can see from the picture above if a fatal failure occurs after the stage 1, the database will be restored to its pre-transaction consistent state. If the failure occurs after any other stage, the transaction information will be available from the transaction commit record and the commit will be restarted when the database will be open again.

The isolation in db4o database is at degree one, which means that a transaction does not overwrite "dirty data" of the other transactions and does not commit any writes until it completes all its writes (transaction commit record).

There are 2 settings that can effect ACID behavior in db4o:

1. Java:

```
configuration.flushFileBuffers(false)
```

this setting can be potentially dangerous on systems using in-memory file caching. Instead of carrying out all writes immediately, the kernel stores data temporally in the buffer cache, waiting to see if it is possible to group several writes together. Cached file changes can also be reversed, which can break the ACID model.

## 2. Java:

```
configuration.disableCommitRecovery()
```

This setting disables commit recovery when the fatal failure occurs on stage 2-3 of the commit process. This setting should only be used in emergency situations after consulting db4o support. The ACID flow of the commit can be re-enabled after restoring the original configuration.

This revision (2) was last Modified 2007-05-22T11:27:41 by Tetyana.

# Isolation Level For Db4o

The default isolation level for db4o is read-committed. In your transaction, you can see the committed changes from the other transactions. However, don't forget that db4o uses object reference cache to speed up the retrieval. This means that if the object was already retrieved in the current transaction you will need to call `extObjectContainer.refresh(object)` to obtain the most recent value of the object.

Another interesting feature to point out: when using Lazy or Snapshot queries phantom reads can occur if the other transaction commits during the current transaction query execution. For more information see [Query Modes](#).

Please, note that the isolation level is only applicable for client-server version of db4o. If you are sharing the same object container between several users, you are actually working within one transaction. If you need to have isolation, you can implement it on your application level.

This revision (3) was last Modified 2007-06-15T10:06:26 by Tetyana.

# Concurrency Control And Locking

Concurrency control and locking is a mechanism used by DBMS to ensure that database transactions are executed in a safe manner. Atomicity, consistency, and isolation are achieved through concurrency control and locking. See [ACID Model](#).

When data is accessed from more than one transaction concurrently, it is usually necessary to ensure that only one transaction at a time can change a data item. Locking is a way to do this. Because of locking, all changes to a particular data item will be made in the correct order in a transaction. See [Isolation](#).

The following types of locking are usually distinguished:

More Reading:

- [Pessimistic Locking](#)
- [Optimistic Locking](#)
- [Overly Optimistic Locking](#)
- [Concurrency Control In Db4o](#)
- [Types Of Locks](#)
- [Locks In Db4o](#)

This revision (2) was last Modified 2007-05-22T11:44:41 by Tetyana.

# Pessimistic Locking

In pessimistic locking approach an entity is locked for the entire time the entity is in application memory (often in the form of an object). A read lock indicates that the object can be read but not modified or deleted by the other transactions. A write lock indicates that the object is locked exclusively and only the current transaction can read, modify or delete the object.

**Advantage:** pessimistic locking is easy to implement and guarantees that your changes to the database are made consistently and safely.

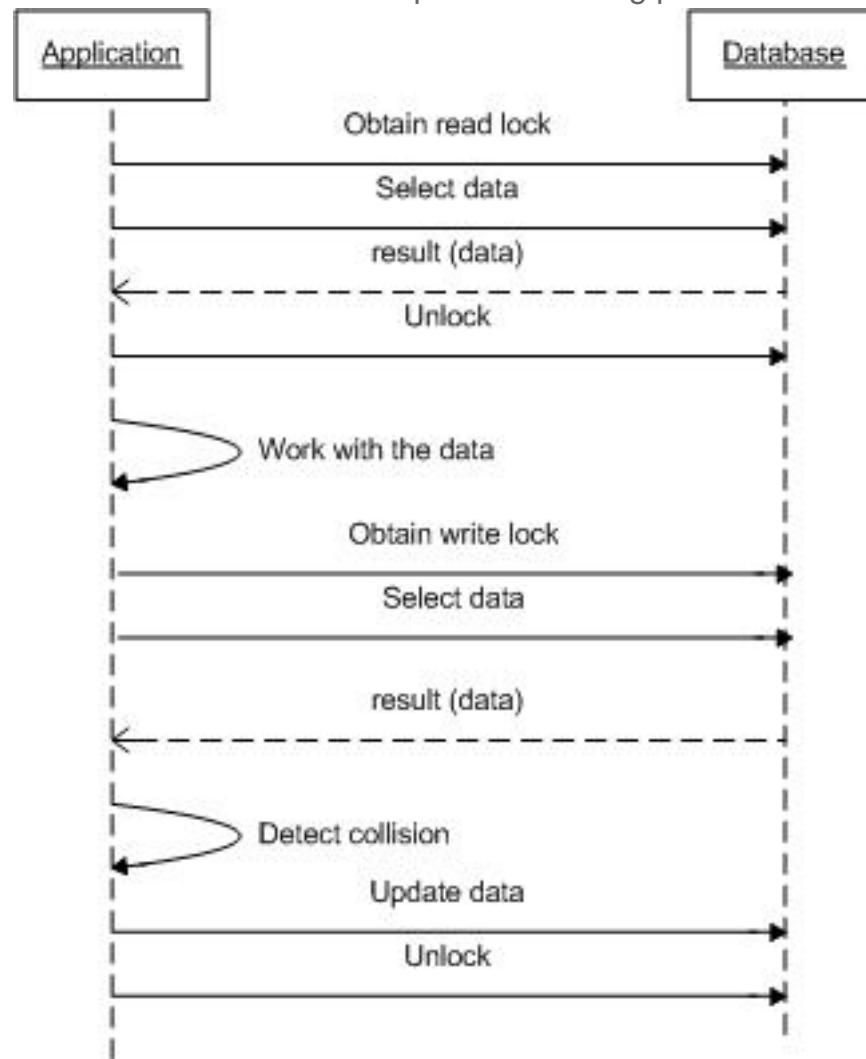
**Disadvantage:** pessimistic locking is not scalable; in systems with many users or long-running transactions, the waiting time for a lock to be released can be too long.

This revision (1) was last Modified 2007-05-22T11:46:16 by Tetyana.

# Optimistic Locking

In the real world there are many systems where collisions are not very frequent. For example if 2 users are processing bank transfers they may work with account objects but the accounts are different, so that they do not collide.

For the situation described above, optimistic locking will be a reasonable solution. When optimistic locking is used, it is accepted that collisions may occur, but instead of trying to prevent them, the system tries to detect and resolve them. Optimistic locking process flow is shown on the figure below.



1. Data is retrieved with a read lock. The lock is released immediately after the retrieval.
2. Data modifications are done on unlocked data.
3. Before committing the modifications, the system tries to detect a collision. If a collision is detected, it should be handled according to the system rules (for example: the collision information can be logged or displayed to the user, transaction modifications can be merged, rejected or accepted).
4. The data is committed to the database or rolled back. The lock is released.

There are different ways to detect a collision. For example, you can mark the retrieved objects with a unique identifier, timestamp or username. Alternatively, you can store a copy of the originally retrieved

object and compare it to the object from the database before trying to update.

This revision (1) was last Modified 2007-05-22T11:48:53 by Tetyana.

# Overly Optimistic Locking

Overly Optimistic Locking is a strategy, which assumes that the collision will never occur, therefore the system does not try to prevent a collision or to detect it. It is important to understand that this system can be only used in a single-user mode. Multi-user mode will require additional concurrency control strategy to be applied.

This revision (1) was last Modified 2007-05-22T11:50:50 by Tetyana.

# Concurrency Control In Db4o

Db4o uses overly optimistic concurrency control: an object is locked for read and write, but no collision detection is used. In the same time, db4o provides you with all means to implement a suitable for you concurrency control strategy in your application. For more information see [Concurrency Control](#).

This revision (1) was last Modified 2007-05-22T11:53:19 by Tetyana.

# Types Of Locks

Locking can also be classified by the entities that are being locked. Usually the following types are distinguished:

- Page locking: all the data on a specific memory page is locked.
- Cluster locking: all the objects in a cluster are locked (applies only to cluster-enabled object databases).
- Class or table locking: all objects of a class (OODBMS) or all rows in a table (RDBMS) will be locked.
- Object or instance locking: a single object (OODBMS) or a single relational tuple (RDBMS) will be locked.

This revision (2) was last Modified 2007-06-13T18:36:27 by Tetyana.

# Locks In Db4o

In db4o locks can be implemented with the help of [Semaphores](#). Though you cannot implement page or cluster locking, you can still vary the range of locking by using different semaphore names. For example:

1. The following semaphore will lock all the objects of a class:

Java:

```
extObjectContainer.setSemaphore(Pilot.class.getName, 3000)
```

2. This semaphore will lock a single object

Java:

```
extObjectContainer.setSemaphore("LOCK_"+objectContainer.ext().getID(pilot), 3000)
```

This revision (1) was last Modified 2007-05-22T12:02:20 by Tetyana.

# Object Identity

Db4o keeps references to all persistent objects that are currently held in RAM, whether they were retrieved, created or activated in this session. The main role of the reference system is to provide access to the required data with the best speed and lowest memory consumption. Performance and usability of the reference system depend much on how the system manages objects identities.

More Reading:

- [Unique identity concept](#)
- [Identity Vs Equals](#)
- [Binding objects](#)
- [Weak References](#)

This revision (5) was last Modified 2006-11-14T18:48:08 by Eric Falsken.

# Unique identity concept

Db4o uses the concept of uniqueness of each object in reference cache. If an object is accessed by multiple queries or through multiple navigation access paths, db4o will always return the one single object, helping you to put your object graph together exactly the same way as it was when it was stored, without having to use IDs. You can simply use '==' to check the identity of two database objects.

IdentityExample.java: checkUniqueness

```

01 ①②③private static void checkUniqueness() {
02 |     setObjects();
03 |     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
04 ④⑤⑥try {
05 |     ObjectSet cars = container.query(Car.class);
06 |     Car car = (Car)cars.get(0);
07 |     String pilotName = car.getPilot().getName();
08 |     ObjectSet pilots = container.get(new Pilot(pilotName));
09 |     Pilot pilot = (Pilot)pilots.get(0);
10 |     System.out.println("Retrieved objects are identical: " + (pilot == car.getPilot()));
11 ⑩⑪⑫} finally {
12 |     container.close();
13 | }
14 L }
```

How does db4o realize such behavior? Each object is loaded into reference cache only once in the session: db4o will return a new object only if it is not present in the cache yet, otherwise it will give you a reference to the object already in cache. This helps db4o to distinguish between objects that are to be updated and those ones that are to be created. All "known" objects are the subjects of update whereas "unknown" should be created. (Note that the reference system will only be in place as long as an ObjectContainer is open. Closing and reopening an ObjectContainer will clean the references system of the ObjectContainer and all objects in RAM will be treated as "new" afterwards.).

IdentityExample.java: checkReferenceCache

```

01 ①②③private static void checkReferenceCache() {
02 |     setObjects();
03 |     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
04 ④⑤⑥try {
05 |     ObjectSet pilots = container.query(Pilot.class);
06 |     Pilot pilot = (Pilot)pilots.get(0);
07 |     String pilotName = pilot.getName();
08 |     pilot.setName("new name");
09 |     System.out.println("Retrieving pilot by name: " + pilotName);
10 |     ObjectSet pilots1 = container.get(new Pilot(pilotName));
```

```

11 |     listResult(pilots1);
12 | } finally {
13 |     container.close();
14 | }
15 | }
```

In the example *pilot* object is retrieved from the database (placed into cache) and changed, but not saved. The following retrieval uses pilot's name to retrieve the object from the database, but that object was already instantiated, so its cached (and modified) instance is actually returned.

Such behavior can be sometimes undesirable - you may expect to get object as it saved in the database instead of its modified instance in cache. One of the ways to do that is to use [ExtObjectContainer#peekPersisted\(object\)](#) method, which will give you a disconnected copy of a database object.

Another way is to purge objects from the cache before re-retrieving them.

You can use the following methods:

- ExtObjectContainer#isCached(object) shows if the object is present in reference cache
- ExtObjectContainer#purge(object) removes the object from the cache.

Let's look at our previous example extended with these methods:

#### IdentityExample.java: checkReferenceCacheWithPurge

```

01 | private static void checkReferenceCacheWithPurge() {
02 |     setObjects();
03 |     ObjectContainer container = Db4o.openFile(DB40_FILE_NAME);
04 |     try {
05 |         ObjectSet pilots = container.query(Pilot.class);
06 |         Pilot pilot = (Pilot)pilots.get(0);
07 |         String pilotName = pilot.getName();
08 |         pilot.setName("new name");
09 |         System.out.println("Retrieving pilot by name: " + pilotName);
10 |         long pilotID = container.ext().getID(pilot);
11 |         if (container.ext().isCached(pilotID)){
12 |             container.ext().purge(pilot);
13 |         }
14 |         ObjectSet pilots1 = container.get(new Pilot(pilotName));
15 |         listResult(pilots1);
16 |     } finally {
17 |         container.close();
18 |     }
19 | }
```

Now the second retrieval re-instantiates Pilot object from the database.

An object removed with ExtObjectContainer#purge(object) becomes "unknown" to the ObjectContainer, so this method may also be used to create multiple copies of objects:

#### IdentityExample.java: testCopyingWithPurge

```

01 ①②③private static void testCopyingWithPurge() {
02 |     setObjects();
03 |     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
04 ④⑤    try {
05 |         ObjectSet pilots = container.query(Pilot.class);
06 |         Pilot pilot = (Pilot)pilots.get(0);
07 |         container.ext().purge(pilot);
08 |         container.set(pilot);
09 |         pilots = container.query(Pilot.class);
10 |         listResult(pilots);
11 ⑥⑦    } finally {
12 |        container.close();
13 |    }
14 }
```

Each reference in db4o works directly with the object. As only one instance of the object exists in cache there is no problem with object locks.

You can see an example of another concept used in JDO system <http://access1.sun.com/jdo/>

Actually db4o reference is a pointer to the object in the database file. It means that the size of the database does not affect query time: the object is retrieved from the known position without any necessity to traverse values.

This revision (13) was last Modified 2008-03-02T16:39:20 by Tetyana.

# Identity Vs Equals

One of the most common questions of db4o users is: why does not db4o allow to use equals() and hashCode to identify objects in the database. From the first glance it seems like a very attractive contract - let the developer decide what should be the base for comparing objects and making them unique in the database. For example if the database identity is based on the object's field values it will prevent duplicate objects from being stored to the database, as they will automatically be considered one object.

Yes, it looks attractive, but there is a huge pitfall: when we deal with objects, we deal with their references to each other comprising a unique object graph, which can be very complex. Preserving these references becomes a task of storing many-to-many relationships. This task can only be solved by providing unique identification to each object **in memory** and not only in the database, which means that it can't depend on the information stored in the object (like an aggregate of field values).

To see it clearly, let's look at an example. Suppose we have Pilot{string name} and Car{Pilot pilot} classes, and their equals method is based on comparing field values:

1. Store a pilot1(name="name1") and car1(pilot=pilot1) to the database
2. Retrieve pilot1
3. change pilot1(name = "name1") to pilot1(name="name2"). Note that though it is the same object from the runtime point of view, these are 2 different objects for the database based on equals comparison.
4. Now let's try to retrieve the car object, which has pilot = pilot1. We will get no results as the initial pilot stored with the database is not equal to the pilot1(name="name2"), and there is no car for the updated pilot anymore!

Now, this was a simple example, and can be solved by updating the car object together with the pilot. But what happens if there are thousands of objects referencing this pilot instance? They will all have to be retrieved and updated. Further, those objects can be also referenced somewhere and potentially a single update in a pilot object can trigger the re-write of the whole database.

Objects without identity also make Transparent Persistence and Activation impossible, as there will be no way to decide which instance is the right one for update or activation.

So unique identification of database objects in memory is unavoidable and identity based on an object reference is the most straightforward way to get this identification.

This revision (2) was last Modified 2008-03-02T17:47:16 by Tetyana.

# Binding objects

Db4o adds additional flexibility to its reference system allowing the user to re-associate an object with its stored instance or to replace an object in database:

Java:

```
ExtObjectContainer#bind(object, id)
```

Typical usecases could be:

- [enums and static fields](#)
- working on objects disconnected from the database
- refactoring

The following requirements should be met:

- The ID needs to be a valid internal object ID, previously retrieved with ExtObjectContainer#getID(object)
- The object parameter needs to be of the same class as the stored object.

Calling ExtObjectContainer#bind(object,id) does not have any impact on persisted objects. It only attaches the new object to the database identity. ObjectContainer#set(object) should be used to persist the change.

Let's look how it works in practice.

**IdentityExample.java: testBind**

```
01 | private static void testBind() {
02 |     setObjects();
03 |     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
04 |     try {
05 |         Query q = container.query();
06 |         q.constrain(Car.class);
07 |         q.descend("model").constrain("Ferrari");
08 |         ObjectSet result = q.execute();
09 |         Car car1 = (Car)result.get(0);
10 |         long IdCar1 = container.ext().getID(car1);
```

```

11 |     Car car2 = new Car("BMW", new Pilot("Rubens Barrichello"));
12 |     container.ext().bind(car2, IdCar1);
13 |     container.set(car2);
14 |
15 |     result = container.query(Car.class);
16 |     listResult(result);
17 } finally {
18 |     container.close();
19 |
20 }

```

So this method gives you control over internal object storage. But its usage is potentially dangerous and normally should be avoided. Let's look at an example how bind can damage your object consistency:

Imagine three objects referencing eachother:

a1 => b1 => c1

Now if you call #bind() to replace b1 with b2 in memory you will get the following:

a1 => b1 => c1  
b2 => c1

b2 will be the new in-memory copy of the persistent object formerly known as b1.

a1 will still point to b1 which is now a transient object.

If you now store a1, you will get a duplicate copy of b1 stored.

Please, remember this scenario and use ExtObjectContainer#bind(object,id) only for short-lived objects and in controlled situations where no other references exist.

For the scenarios, which merging disconnected transient object, please refer to [Merge Module](#) project suggested design.

This revision (13) was last Modified 2008-03-02T17:59:04 by Tetyana.

# Weak References

Each retrieved or created object is automatically placed into reference system. Of course you have control over it and can purge or deactivate retrieved objects to prevent ever-growing memory consumption. However this requires a lot of attention and coding effort. Luckily, this is not necessary as db4o offers much easier way to manage the memory - WeakReferences.

Any object is kept in the memory while application has references to it otherwise it becomes eligible for garbage collection.

In the default configuration db4o uses weak references and a dedicated thread to clean them up after objects have been garbage collected by the VM. Weak references need extra resources and the cleanup thread will have a considerable impact on performance since it has to be synchronized with the normal operations within the ObjectContainer.

This revision (3) was last Modified 2006-11-14T09:00:22 by Tetyana.

# Transaction

## Contents

- [Commit And Rollback](#)
- [Refresh Live Objects](#)

All work within db4o ObjectContainer is transactional. A transaction is implicitly started when you open a container, and the current transaction is implicitly committed when you close it again. db4o transaction is tied to an open object container and only one transaction is allowed per object container instance.

## Commit And Rollback

You may choose to make a commit explicit or you may leave it for the `#close()` call:

### TransactionExample.java: storeCarCommit

```

1 private static void storeCarCommit(ObjectContainer container) {
2     Pilot pilot=new Pilot("Rubens Barrichello", 99);
3     Car car=new Car("BMW");
4     car.setPilot(pilot);
5     container.set(car);
6     container.commit();
7 }
```

### TransactionExample.java: listAllCars

```

1 private static void listAllCars(ObjectContainer container) {
2     ObjectSet result=container.get(Car.class);
3     listResult(result);
4 }
```

Before transaction is committed all the modifications to a database are written to a temporary memory storage. Commit (explicit or implicit) writes the modifications to the disk.

Please, remember to always commit or close your ObjectContainer when the work is done, to make sure that the data is saved to the permanent storage. [Commit Strategies](#) contains some important information on when

and how commit should be used to achieve the best performance.

If you do not want to save changes to the database, you can call rollback, resetting the state of our database to the last commit point.

#### TransactionExample.java: storeCarRollback

```

1 private static void storeCarRollback(ObjectContainer container) {
2     Pilot pilot=new Pilot("Michael Schumacher", 100);
3     Car car=new Car("Ferrari");
4     car.setPilot(pilot);
5     container.set(car);
6     container.rollback();
7 }
```

#### TransactionExample.java: listAllCars

```

1 private static void listAllCars(ObjectContainer container) {
2     ObjectSet result=container.get(Car.class);
3     listResult(result);
4 }
```

## Refresh Live Objects

There is one thing that you should remember when rolling back: the #rollback() method will cancel the modifications, but it won't change back the state of the objects in your reference cache.

#### TransactionExample.java: carSnapshotRollback

```

1 private static void carSnapshotRollback(ObjectContainer container) {
2     ObjectSet result=container.get(new Car("BMW"));
3     Car car=(Car)result.next();
4     car.snapshot();
5     container.set(car);
6     container.rollback();
7     System.out.println(car);
```

8 L }

You have to explicitly refresh your live objects when their state might become different from the state in the database:

TransactionExample.java: carSnapshotRollbackRefresh

```
1 [+]private static void carSnapshotRollbackRefresh(ObjectContainer container) {  
2 |     ObjectSet result=container.get(new Car("BMW"));  
3 |     Car car=(Car)result.next();  
4 |     car.snapshot();  
5 |     container.set(car);  
6 |     container.rollback();  
7 |     container.ext().refresh(car, Integer.MAX_VALUE);  
8 |     System.out.println(car);  
9 L }
```

The `#refresh()` method might be also helpful when the changes to the database are done from different threads. See [Client-Server](#) for more information.

This revision (13) was last Modified 2007-08-20T12:18:37 by Tetyana.

# Database Models

This section covers object and relational database models and discusses them in comparison.

More Reading:

- [Object Database Model](#)
- [Relational Database Model](#)
- [Object And Relational Model Comparison](#)
- [Object-Relational How To](#)

This revision (1) was last Modified 2007-06-02T19:24:00 by Tetyana.

# Object Database Model

Object model is characterized by:

[Data Abstraction](#)

[Encapsulation](#)

[Inheritance](#)

This revision (1) was last Modified 2007-06-02T19:24:52 by Tetyana.

# Data Abstraction

Data Abstraction enables isolation of the object's implementation details from the way it is used. Data Abstraction groups the pieces of data that describe some entity, so that programmers can manipulate that data as a unit. It helps a programmer to cope with the complexity of data by hiding the details.

In the object database model, each single entity is called an object instance. Each object has a unique unchanging identity. Object identity is characterized by the following features:

- object identity is independent from the data contained in the object - internal data values are not used to generate an identity;
- object identities are generated by the object system and are not controlled by a programmer or a database system;
- object identity lives as long as the object itself, it does not change with the modifications of the objects data content.

Objects are described by classes. A class defines a structure and attributes (fields) of an object. Classes are also used to define hierarchical properties: child and parent relationships.

Object model directly supports references. Note that references establish a connection between objects using their identities. For example:



In this example Pilot object is referenced from a Car object.

This revision (2) was last Modified 2007-06-02T19:28:21 by Tetyana.

# Encapsulation

Encapsulation is an object model concept, which allows to hide specific behavior or processing abilities within object instances defined by a class. Method definitions within a class are an integral part of encapsulation, which allows to store data and code together.

For example:

Car
+getPilot() : Pilot

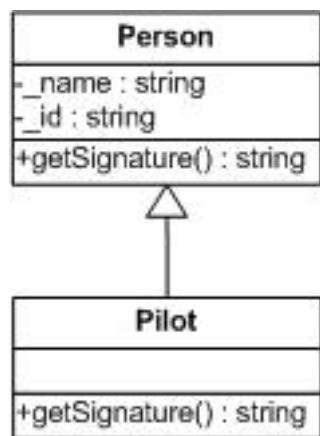
getPilot method can be "attached to" a Car object or encapsulated. Alternatively, getPilot method can be stored in the application or a separate library and distributed with the object database. However, encapsulation approach has an important advantage: method code cannot be lost or outdated as in the case of an application/library storage. Object system recognizes which methods belong to which data. The process of the correct method linking to the object is called ***dispatching***.

This revision (1) was last Modified 2007-06-02T19:29:45 by Tetyana.

# Inheritance

Inheritance is a way to form new classes based on the classes that have already been defined.

For example:



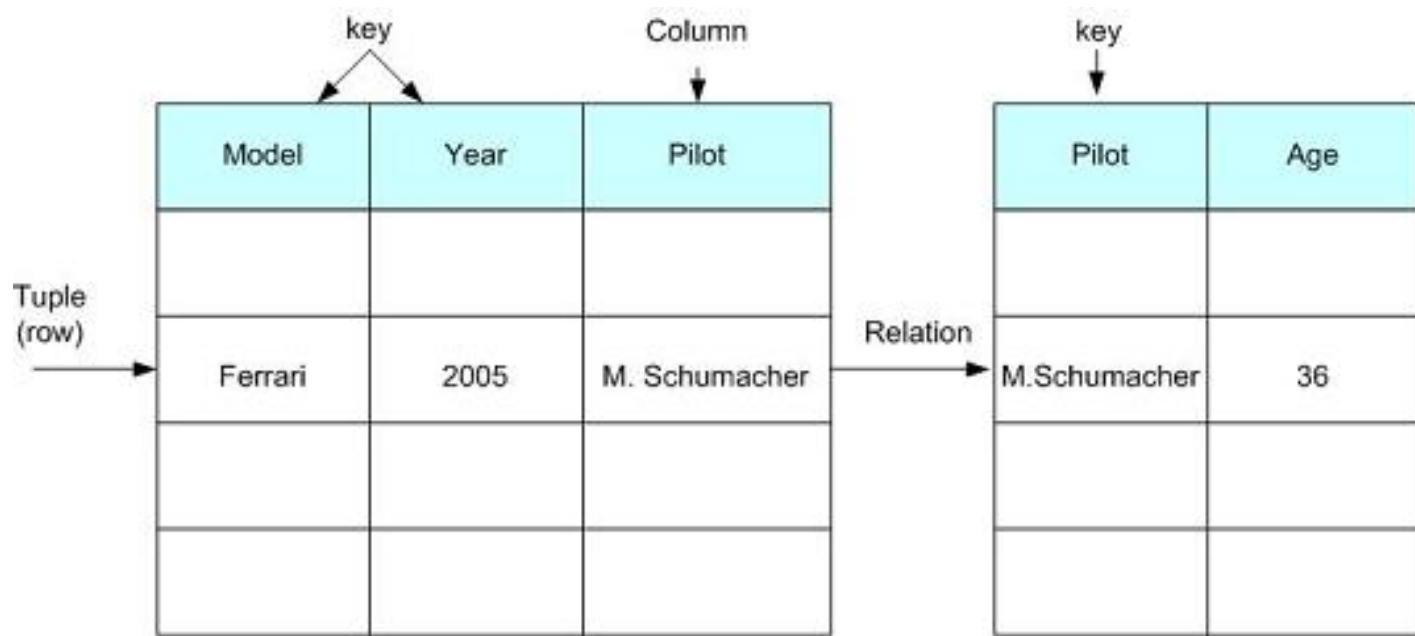
The derived class inherits all the fields and methods of the base class. Note that in the object model, there is no distinction between using system and user-defined types, so you can define sub-types of the system types. This feature is known as extensibility.

In the example above `getSignature` method is overridden in the derived class - this is known as **polymorphism**. The ability of the object model to execute the correct method based on the type of a class is called **dispatching**.

This revision (1) was last Modified 2007-06-02T19:30:55 by Tetyana.

# Relational Database Model

Relational data model is based on the concept of a relation or a table. An entity is presented by a tuple or row. Table fields are used to describe entity attributes (Model, Year, Pilot, Age in the example below). Each entity should be identified by a primary key, which is constructed from the data in the row. In the example below the key is created by combining the values of "Model" and "Year" fields.



In addition to the primary key relational data model supports foreign keys: a primary key of one table used in another table to allow joining of data. In the example above the primary key from the Pilots table ("Pilot" field) is used in the Cars table; it allows to associate a car and a pilot.

This revision (1) was last Modified 2007-06-02T19:32:31 by Tetyana.

# Object And Relational Model Comparison

Object and relational models though serving the same goal are very different in their concepts. There is no solid answer which technology is better or superior. In most cases when deciding between the 2 approaches you will have to base your decision on a close requirements analysis.

See [RDBMS And OODBMS Application](#) for a list of environments and requirements where OODBMS is preferred.

More Reading:

- [Basic Terms](#)
- [RDBMS And OODBMS Application](#)

This revision (2) was last Modified 2007-06-02T20:18:25 by Tetyana.

# Basic Terms

While relational and object databases use different sets of terms, it is possible to draw parallels between some of the terms:

OODBMS	RDBMS	Similar	Different
Class	Table	Define an entity data structure.	Class can define fields (static data) and methods, while table can only define static data.
Object instance	Tuple	Represent an instance of data defined in a class or a table.	Object can hold data of different visibility and references to other objects. Tuple consists of system type data.
Attribute	Column	Define one of the fields in the data definition.	RDBMS has a preset collection of types, which can be used to define the type of the data in the column. OODBMS can use user-defined types.
Method	Stored Procedure	Define a piece of functionality.	Method is a characteristic of a class. Stored procedure is a separate object in RDBMS.
Identity	Key	Identify a single object or tuple.	Object identity makes any object unique independently of the object content. Key imposes a uniqueness of a database object based on the uniqueness of the column values.
Reference	Relation	Link different objects within a database.	In OODBMS direct object links are used. In RDBMS objects are linked through foreign keys, i.e. identical values in different rows.

Note, that the above-mentioned terms are similar only to a certain degree; there are some fundamental differences between them.

This revision (1) was last Modified 2007-06-02T20:16:26 by Tetyana.

# RDBMS And OODBMS Application

## Contents

- Object Oriented Development
- Embedded Applications
- Complex Object Structures And Hierarchies
- Refactoring And Schema Evolution
- Agile Development

Relational data systems are proven market leaders in persistence solutions. They are mature and reliable choice for a general purpose solution. Moreover, RDBMS is supported by a huge number of applications and components and a large amount of human resources experienced in the technology.

However, OODBMS has some valuable advantages, which can't be underestimated:

- native Object Oriented language support (no object-relational impedance mismatch);
- native complex object structures and hierarchies support;
- easy refactoring and product evolution;
- zero-administration;
- continuous regression testing.

Some of the most obvious usecases and advantages of OODBMS are listed below.

## Object Oriented Development

---

The fact that OODBMS technology is the perfect match for using in OO development environment is obvious from the very name. Though RDBMS has a wide support in tools and methods of using in OO environment there will always be an overhead of the translation from the object to relational world, commonly known as object-relational impedance mismatch.

## Embedded Applications

---

Embedded device applications need a small-footprint zero-administration database, which makes a perfect match with an OODBMS. In addition, they usually need a quick response-time, which can be better achieved by a native OO technology, when no object-relational conversion is needed.

## Complex Object Structures And Hierarchies

---

In applications having to deal with deep object structures (tree or graph), OODBMS offers a much easier

native way to store them. Trees or graphs cannot be easily translated to RDBMS structure, so that the application has to deal with often complex conversion algorithms, which are difficult to maintain and refactor. On the other hand, OODBMS can store these structures transparently without any additional coding.

The same is true for difficult inter-objects relationships. In RDBMS they are realized with foreign keys. In some cases, you will need to fetch object by object until you will reach the final relation. In OODBMS all you need is to specify an [activation depth](#) or use [transparent activation](#) to reach all the related objects through the top-object fields.

From the said above you can conclude that the applications using objects with collection members (one-to-many relationships) will also benefit from OODBMS technology.

## Refactoring And Schema Evolution

---

Most applications evolve as the time goes. It means that their class and data structure (schema) can change. In applications using relational databases schema evolution is quite work-consuming process: the schema must be changed, the class structure must be changed and the query collection must be changed too. In OODBMS applications, only class structure is a subject of change and the process can be highly automated by using modern refactoring techniques available from IDEs.

## Agile Development

---

OODBMS makes it possible to implement agile development process in your team:

- continuous refactoring;
- agile modeling;
- continuous regression testing;
- configuration management;
- developer "sandboxes".

The use of RDBMS technology complicates the adoption of these techniques due to the technical impedance mismatch, the cultural impedance mismatch, and the current lack of tool support.

This revision (2) was last Modified 2007-06-04T17:28:38 by Tetyana.

# Object-Relational How To

This chapter is intended to help people moving from a relational database to db4o. The information here will try to draw parallels between object and relational concepts, explain how to achieve similar effect in both technologies and what practices are best to be used with the object database.

More Reading:

- [Identity And Primary Keys](#)
- [Foreign Keys](#)
- [Referential Integrity](#)
- [Triggers](#)
- [Unique Constraints](#)
- [Indexes](#)
- [Concurrency Control](#)

This revision (1) was last Modified 2007-06-02T20:19:29 by Tetyana.

# Identity And Primary Keys

One of the most important concepts to understand working with a database system is the identification strategy.

In RDBMS separate entities are distinguished with the help of primary keys. Effectively, objects are compared on their field contents and a constrain is used prohibiting 2 different rows in a table to have the same value in a column marked as a primary key.

In the object world, the concept is quite different - any object is unique, independently of the data it holds. For example:

```
class Car

    model
    year

car1 = new Car("BMW", "1999")

car2 = new Car("BMW", "1999")
```

`car1` and `car2` objects consist of the same data, but they are not equal.

On the other hand we can create any amount of references to the object and they will all be equal:

```
car1 = new Car("BMW", "1999")

car2 = car1

car3 = car1
```

In this example `car2` is equal to `car1` and `car3` and all these variables actually reference the same object.

Object database task is to ensure that independently of the way the object was accessed (by different queries or through different navigation access paths) the database must return a reference to the one single object. For a detailed explanation how it is achieved in db4o, please see [Unique identity concept](#).

If you are moving from a relational database to db4o it is very important to understand db4o identity concept. Instead of tying objects to the database reference key you should use the objects, obtained from the database directly without additional worries about their identity.

Unfortunately, there still are some cases, when additional identification system is necessary. This can happen when an object must be referenced out of the application memory boundaries: objects referenced between sessions, objects passed to another application, partially connected applications etc. In these cases, you are encouraged to use db4o internal IDs or Unique Universal IDs (UUID). Internal IDs are unique within one database file and do not change after defragment. UUIDs are unique between all db4o databases and do not change. For more information see [IDs and UUIDs](#).

If your application requires autoincremented ID generation you can look at an example implementation in [Autoincrement](#)

This revision (1) was last Modified 2007-06-02T20:21:13 by Tetyana.

# Foreign Keys

Foreign keys issue is closely connected to the issue of [object identity](#).

In RDBMS foreign keys are created by designating a special column (or set of columns) in 2 different tables to hold the same information. The data in a row of one table is related to the data in a row of another table if the values in those columns match. Usually primary key is used on one side of the equation (can be replaced by unique key). Foreign keys are also called [referential constraints](#).

In db4o there is no concept of a primary key, therefore no foreign key as well. Object relationships are established through direct object references, which are kept in object fields. This approach is perfectly straightforward and intuitive. It considerably simplifies the navigation: instead of issuing more and more queries to access child (referenced) objects, you can just get them by the reference in the object field. This is definitely much more productive and requires less coding.

However, object references fulfill only part of foreign key responsibilities: in addition to referencing another data entity, foreign key ensures the referential integrity. The [following chapter](#) discusses how referential integrity can be implemented in db4o.

This revision (2) was last Modified 2007-06-02T20:23:35 by Tetyana.

# Referential Integrity

Referential integrity (RI) is an important feature of RDBMS. It helps to protect data from misuse and corruption. However, in the modern world of multi-tier and distributed technologies it becomes questionable if referential integrity should be realized on a database level or is it a responsibility of a business layer. The answer to this question depends on the system design. Usually database RI makes lots of sense in a data-centric application with one main database. More complex, distributed in space and time referential constraints can be better implemented within a business object framework.

db4o database does not provide full referential integrity support, rather it gives a user possibilities to implement RI on the application level. For an example of a referential integrity solution, see [Referential Integrity](#).

This revision (1) was last Modified 2007-06-02T20:23:12 by Tetyana.

# Triggers

Triggers are widely used by RDBMS to automate some work as a response to a certain event. Similar behavior can be easily implemented in db4o with the help of callbacks. See [db4o callbacks](#) for more information.

This revision (1) was last Modified 2007-06-02T20:25:03 by Tetyana.

# Unique Constraints

Unique constraint in RDBMS is a field or combination of fields that uniquely defines a record. In an object database world unique constraints will uniquely define an object. In db4o unique constraints were first introduced in version 6.2 and are still work in progress. Current version supports unique constraints over only one field. For more information see [Unique Constraints](#)

This revision (1) was last Modified 2007-06-02T20:25:40 by Tetyana.

# Indexes

Indexes play an important role in database performance optimization. An index is a special data structure that helps to make searches more effective. In db4o, as in many relational databases, indexes are based on B-Trees. You can define an index for any object field (including private ones).

Several indexes on different fields of the same class will result in better searching performance. Unlike relational databases db4o does not allow indexes over a set of fields. For more information see [Indexing](#) and [Enable Field Indexes](#).

This revision (1) was last Modified 2007-06-02T20:26:11 by Tetyana.

# Concurrency Control

Concurrency control is a vital feature for any database. While most popular relational databases give you a choice of isolation levels and locking types, db4o goes with a default read-committed overly optimistic locking strategy. For more information see [Concurrency Control And Locking](#) and [Isolation Level For Db4o](#). [Locks In Db4o](#) explains how applications can implement customary locks in db4o.

This revision (2) was last Modified 2007-06-02T20:30:21 by Tetyana.

# Native Query Concepts

Database query is a language used to communicate with the database. For a long time databases stood the central and commanding place in software products: the applications were literally written to support the needs of a main central database. However, in the modern high-speed world of distributed computing the transfer and modification of the information plays increasingly important role and the databases more and more often take a place of a temporary storage or a point of exchange.

This evolutionary change results in some practical modifications: more attention is paid to the business logic and tools that are used for its development, language structures are automatically produced and checked by development environments and string-based logic is getting outdated and not enough efficient. Database query language is expected to follow the features of an object-oriented language: type-safe, easy to refactor and test. String-based languages like SQL, OQL, JDOQL do not meet these requirements. To fill this gap and provide developers with sufficient database query tools a new concept was developed by W.Cook and C. Rosenberger which was given a name [Native Queries](#).

In the following paragraphs, we will review the concepts of Native Queries (NQ). We will use Pilot class for all examples suggested further:

## Pilot.java

```

01  /* Copyright (C) 2007 db4o objects Inc. http://www.db4o.com */
02  package com.db4odoc.nqconcepts;
03
04  public class Pilot {
05      private String name;
06      private int points;
07
08      public Pilot(String name, int points) {
09          this.name=name;
10          this.points =points;
11      }
12
13      public String getName() {
14          return name;
15      }
16

```

```
17 public int getPoints() {  
18     return points;  
19 }  
20 }
```

More Reading:

- [Problems Of String Based Query Languages](#)
- [Native Query Characteristics](#)
- [Native Query Implementation](#)
- [Native Query Optimization](#)
- [Code Inside](#)
- [Sorting Results](#)

This revision (2) was last Modified 2007-06-02T18:23:57 by Tetyana.

# Problems Of String Based Query Languages

Let's look how a query can be expressed against the Car class in some of the object querying languages:

## OQL

```
String oql = "select * from pilot in AllPilots where pilot.points < 100";  
  
OQLQuery query = new OQLQuery(oql);  
  
Object pilots = query.execute();
```

## JDOQL

```
Query query = persistenceManager.newQuery(Pilot.class, "points < 100");  
  
Collection pilots = (Collection)query.execute();
```

## db4o SODA, using C#

```
Query query = database.Query();  
  
query.Constrain(typeof(Pilot));  
  
query.Descend("points").Constrain(100).Smaller();  
  
IList pilots = query.Execute();
```

As you can see, query parameters ("points") and constraints ("<100") are expressed as strings, which results in the following problems:

- Modern development environments do not check embedded strings for syntactic and semantic correctness. A small typo in a query (for example 10 instead of 100) will be very difficult trace, moreover it can pass unnoticed to a production version.
- Embedded strings are not affected by refactoring tools. As the application evolves and the changes are made to the field variables, the string-based queries will become obsolete and will need to be changed by hand.
- String queries address fields directly instead of using publicly accessible methods/attributes, breaking encapsulation principle.
- Embedded strings can be on the way of modern agile techniques, which encourage constant refactoring. Since string queries are difficult to refactor and maintain they will delay a decision to refactor and result in a lower-quality code.

## Problems Of String Based Query Languages

- Working with a string query language inside an OO language requires a developer to learn both and switch between them in the development cycle.
- Reusability support of OO languages (method calls, polymorphism, overriding) are not accessible to string-based queries.
- Embedded strings can be subject to injection attacks.

This revision (1) was last Modified 2007-06-02T18:25:48 by Tetyana.

# Native Query Characteristics

The [previous paragraph](#) reveals the need of a new querying technology native to modern development languages. This new technology was developed by [W.Cook and C.Rosenberger](#) and named Native Queries.

Native Queries are characterized by the following:

- 100% native: Queries are completely expressed in the implementation language (Java or c#), and they fully obey all language semantics.
- 100% object-oriented: Queries are runnable in the language itself, to allow unoptimized execution against plain collections without custom preprocessing.
- 100% type-safe: Queries are fully accessible to modern IDE features like syntax checking,

type checking, refactoring, etc.

- optimizable: It is possible to translate a native query to a persistence architecture's query language or API for performance optimization. This is done at compile time or at load time by source code or bytecode analysis and translation. However, not every construct within a Native Query can be optimized.

This revision (2) was last Modified 2007-06-02T18:34:24 by Tetyana.

# Native Query Implementation

So how are Native Queries realized in practice?

Let's return to the [Pilot](#) class. Suppose we need to find all pilots with the name starting from "M" and points over 100. In a native OO language it would be expressed as:

Java:

```
pilot.getName().startsWith("M") && pilot.getPoints() > 100
```

In order to pass this condition to a database, collection or other query processor a special object is used. In .NET2 it is a delegate, in Java5 it is a named method.

Java:

```
public abstract class Predicate <ExtentType> {  
  
    public <ExtentType> Predicate (){}  
  
    public abstract boolean match (ExtentType candidate);  
  
}  
  
new Predicate <Pilot> () {  
  
    public boolean match(Pilot pilot){  
  
        return pilot.getName().contains("M") && pilot.getPoints() > 100;  
  
    }  
  
}
```

For more information about NQ implementations in the other Java and .NET versions see [Native Query Syntax](#).

This revision (4) was last Modified 2007-06-04T17:25:45 by Tetyana.

# Native Query Optimization

Though Native Query API discussed in the [previous paragraph](#) is simple and straightforward, the real challenge is to provide a performant solution.

If the NQ code is run as is, it requires instantiation of all the members of a class. This is very slow in most cases. In order to improve the performance a special optimizer is used by db4o. The idea of the optimization is to analyze the code in a Native Query and provide an alternative in a database query language. This can be done in runtime or build time.

Obviously, optimization is not possible in cases, when a native query does not have a database query alternative. To reveal those cases [db4o Diagnostic](#) system should be used.

For more information see [Native Query Optimization](#).

This revision (1) was last Modified 2007-06-02T18:41:28 by Tetyana.

# Code Inside

Native Query expression is dealt as a normal piece of code. Therefore, any language construction is legible inside:

- variables;
- temporary objects created within a query;
- static calls;
- exception handling.

However, some restrictions do apply:

- NQ should not be used to modify the database to prevent loops;
- NQ should not use threads, as NQ are expected to be triggered in large numbers;
- NQ should be fast, therefore they should not interact with the GUI;
- NQ should follow security restrictions, as they are executed in the server and potentially can create malicious behavior there.

In a case of uncaught exception within a Native Query a null result is returned.

For more info see [Native Query Collection](#)

This revision (1) was last Modified 2007-06-02T18:42:57 by Tetyana.

# Sorting Results

Native Queries also provide a native interface for sorting of the results using Comparator/IComparer:

Java:

```
ObjectSet query(Predicate predicate, Comparator comparator);
```

For more information see [Native Query Sorting](#).

This revision (3) was last Modified 2007-06-02T18:46:03 by Tetyana.

# Object Lifecycle

This topic set explains the lifecycle of an object within db4o database. Reading through the topics you will learn to open and close a database, prepare your objects for persistence, store them to db4o, and retrieve using differing querying strategies.

More Reading:

- [Object Container](#)
- [Simple Persistence](#)
- [Class Name Format In .NET](#)
- [Querying](#)
- [Transparent Persistence](#)
- [Working With Structured Objects](#)
- [Activation](#)
- [Update Depth](#)
- [Delete Behavior](#)
- [Object Construction](#)

This revision (3) was last Modified 2007-05-07T17:56:30 by Tetyana.

# Object Container

## Contents

- [Accessing A Database](#)
- [Working With Objects](#)
- [Extended Object Container Interface](#)

Db4o gives you a simple and straightforward interface to object persistence - ObjectContainer. In .NET versions a conventional name IObjectContainer is used.

## Accessing A Database

---

ObjectContainer is your db4o database.

Java:

```
ObjectContainer container = Db4o.openFile(filename)
```

Applications using db4o versions lower than 6.1 can issue multiple calls to open the same file (without closing it). These calls will get the already existing open ObjectContainer instance and no error will be produced. However as many close calls will have to be issued to close the ObjectContainer.

This behavior was changed since db4o version 6.1: only the first call against a file can be successful. Subsequent calls that request to open a database file that is already open will get a DatabaseFileLockedException. This behavior is much more intuitive and allows to simplify the internal db4o design as well.

## Working With Objects

---

ObjectContainer interface gives you all the basic functionality to work with persistent objects. Normally you can save a new or updated object of any class using `objectContainer.set(object)`

Deletion is done with the following method:

Java:

```
container.delete(object)
```

Through `ObjectContainer#get` and `ObjectContainer#query` you get access to objects retrieval functionality.

## Object Container Features

The characteristic features of an ObjectContainer are:

- An ObjectContainer can either be a database in a single-user mode or a client connection to a db4o server.
- Every ObjectContainer owns one transaction. All work is transactional. When you open an ObjectContainer, you are in a transaction, when you commit() or rollback(), the next transaction is started immediately.
- Every ObjectContainer maintains its own references to stored and instantiated objects. In doing so, it manages object identities, and is able to achieve a high level of performance.
- ObjectContainers are intended to be kept open as long as you work against them. When you close an ObjectContainer, all database references to objects in RAM will be discarded.

Basically ObjectContainer supplies functionality, which is enough for the most common usage of db4o database.

## Extended Object Container Interface

---

Additional db4o features are provided by an interface extending ObjectContainer - ExtObjectContainer.

The idea of splitting basic and advanced functionality between 2 interfaces is:

- Keep the root package/namespace very small and well readable.
- Separate vital and optional functionality.
- Make it easy for other products to implement the basic db4o interface.
- Show an example of how a lightweight version of db4o could look.

Every ObjectContainer object is also an ExtObjectContainer. You can cast it to ExtObjectContainer or you can use #ext() method to get to the advanced features.

This revision (10) was last Modified 2007-04-23T05:58:56 by Tetyana.

# Simple Persistence

db4o makes your work with persistent objects very simple and straightforward. The only #store(object) method is used for both saving and modification of any object that exists in your model.

## QueryExample.java: storePilot

```

01  public static void storePilot() {
02 |     new File(YAPFILENAME).delete();
03 |     ObjectContainer db=Db4o.openFile(YAPFILENAME);
04 |     try {
05 |         Pilot pilot=new Pilot("Michael Schumacher", 0);
06 |         db.set(pilot);
07 |         System.out.println("Stored "+pilot);
08 |         // change pilot and resave updated
09 |         pilot.addPoints(10);
10 |         db.set(pilot);
11 |         System.out.println("Stored "+pilot);
12 |     } finally {
13 |         db.close();
14 |     }
15 |     retrieveAllPilots();
16 |
}

```

## QueryExample.java: updatePilotWrong

```

01  public static void updatePilotWrong() {
02 |     storePilot();
03 |     ObjectContainer db=Db4o.openFile(YAPFILENAME);
04 |     try {
05 |         // Even completely identical Pilot object
06 |         // won't work for update of the saved pilot
07 |         Pilot pilot = new Pilot("Michael Schumacher", 10);
}

```

```

08 |         pilot.addPoints(10);
09 |         db.set(pilot);
10 |         System.out.println("Added 10 points for "+pilot);
11 |     } finally {
12 |         db.close();
13 |     }
14 |     retrieveAllPilots();
15 |
}

```

## QueryExample.java: updatePilot

```

01 | public static void updatePilot() {
02 |     storePilot();
03 |     ObjectContainer db=Db4o.openFile(YAPFILENAME);
04 |     try {
05 |         // first retrieve the object from the database
06 |         ObjectSet result=db.get(new Pilot("Michael Schumacher", 10));
07 |         Pilot found=(Pilot)result.next();
08 |         found.addPoints(10);
09 |         db.set(found);
10 |         System.out.println("Added 10 points for "+found);
11 |     } finally {
12 |         db.close();
13 |     }
14 |     retrieveAllPilots();
15 |
}

```

Deletion is just as easy:

## QueryExample.java: deletePilot

```

01 | public static void deletePilot() {
02 |     storePilot();
03 |     ObjectContainer db=Db4o.openFile(YAPFILENAME);

```

```

04  try {
05 //      first retrieve the object from the database
06 ObjectSet result=db.get(new Pilot("Michael Schumacher", 10));
07 Pilot found=(Pilot)result.next();
08 db.delete(found);
09 System.out.println("Deleted "+found);
10 } finally {
11     db.close();
12 }
13 retrieveAllPilots();
14 }
```

The objects are identified by their references in an application cache. You do not need to implement any additional identification systems (like primary keys in RDBMS). See [Identity chapter](#) for details. The uniqueness of an object is defined only by its reference, if you will create 2 objects of the same class with exactly the same fields and save them to db4o - you will get 2 objects in your database. As you can see from the examples an object instance should be retrieved from the database before updating or deleting or you can use the newly created object if it was stored in the same session. Creating a new instance identical to the object in the database and saving it, will create a new object in the database.

Db4o does all the "dirty" work of objects transition between your classes and persistent state using [Reflection](#). No mappings or additional coding is needed from your side. If you will need to change your application model for the next version you will also be surprised with the simplicity: all the changes are done in one place - your code, and the most common operations are done completely automatically (see [Refactoring And Schema Evolution](#) chapter for details).

Please, remember that all db4o work is done within [Transaction](#), which can be committed or rolled back depending on the result you want to achieve.

This revision (14) was last Modified 2008-01-28T06:11:53 by Tetyana.

# Class Name Format In .NET

This topic applies to .NET version only

Db4o uses full class name to distinguish classes within the database file. In .NET full class name has the following format:

Namespace.ClassName , AssemblyName

Effectively that means that the same class definition within different assemblies (applications or libraries) will be recognized as two different classes by db4o. You should keep this in mind in the following cases:

- 2 or more applications working with the same database file;
- client/server application with the classes deployed on both the client and the server;
- [ASP.NET2](#) application with dynamic compilation

Let's use an example to see what happens in these cases. We will create 2 applications Test1.exe and Test2.exe. Both will have a simplest class definition:

Test1 application will store one object of Test class to the database:

Another application (Test2) will try to read this object from the same database file. To check how the Test object was actually stored in the database we will use StoredClass API:

From the example we can see that though the class has been stored to the database, it cannot be retrieved from the Test2 application, as the assembly name is different from the original.

In order to make your classes readable from another assembly you should use one of the existing workarounds:

- keep your persistent classes in a separate class library, which should be available for your application assemblies (for the example above compile the Test class into Persistent.dll);
- use db4o [Aliases](#).

This revision (16) was last Modified 2007-05-07T07:37:38 by Tetyana.

# Querying

db4o supplies three querying systems, Query-By-Example (QBE), Native Queries (NQ), and the SODA Query API.

[Queries-By-Example](#) (QBE) are appropriate as a quick start for users who are still acclimating to storing and retrieving objects with db4o, but they are quite restrictive in functionality.

[Native Queries](#) (NQ) are the main db4o query interface, recommended for general use.

[LINQ](#) queries are a convenient alternative to .NET users.

The [SODA Query](#) is the underlying internal API. It is provided for backward compatibility and it can be useful for dynamic generation of queries, where NQ are too strongly typed. There may be queries that will execute faster in SODA style, so it can be used to tune applications.

Of course, you can mix these strategies as needed.

For more information on custom query comparators see [Custom Query Comparator](#).

More Reading:

- [Query By Example](#)
- [Native Queries](#)
- [SODA Query](#)
- [LINQ](#)
- [Query Modes](#)
- [SODA Evaluations](#)
- [Sorting Query Results](#)
- [Custom Query Comparator](#)

This revision (18) was last Modified 2008-03-15T17:37:17 by Tetyana.

# Query By Example

When using *Query By Example*(QBE) you provide db4o with a template object. db4o will return all of the objects which match all non-default field values. This is done via reflecting all of the fields and building a query expression where all non-default-value fields are combined with AND expressions. Here's a simple example:

PersistentExample.java: retrievePilotByName

```
1 | private static void retrievePilotByName(Obj ectContainer container) {  
2 |     Pilot proto = new Pilot("Michael Schumacher", 0);  
3 |     Obj ectSet result = container.get(proto);  
4 |     listResult(result);  
5 | }
```

Querying this way has some obvious limitations:

- db4o must reflect all members of your example object.
- You cannot perform advanced query expressions. (AND, OR, NOT, etc.)
- You cannot constrain on values like 0 (integers), "" (empty strings), or nulls (reference types) because they would be interpreted as unconstrained.
- You need to be able to create [objects without initialized fields](#). That means you can not initialize fields where they are declared.
- You need a [constructor](#) to create objects without initialized fields.

For more information see [QBE Limitations](#). To get around all of these constraints, db4o provides the [Native Query](#) (NQ) system.

This revision (7) was last Modified 2007-10-18T06:39:16 by Tetyana.

## QBE Limitations

As it was mentioned **before** QBE has some serious limitations

Query-By-Example evaluates all non-null fields and all simple type variables that do not hold their default values against the stored objects. Check to make sure that you are not constraining the resultset by accidentally initializing variables on your template objects. Typical places could be:

- Constructors
  - Constructors in ancestors of your class
  - Static initialization
  - Static initialization in ancestors of your class

The following classes provide an example of classes that cannot be used with QBE

- Pilot1
  - Pilot1Derived
  - Pilot2
  - Pilot2Derived

The following examples show the results of QBE usage with the classes above. Note, that there are some differences between Java and .NET behavior:

- in Java QBE does not return any results when static initialization or initialization in a constructor is present;
  - in .NET QBE will return results in the above-mentioned cases, but it can't guarantee that the result will be correct if the value was changed in the constructor.

1. QBE used against a class that has arbitrary member initialization in the constructor

## QBEExample.java: test1

```
01 private static void test1() {  
02     ObjectContainer container = database();  
03     if (container != null) {  
04         try {  
05             // Pilot1 contains initialisation in the constructor  
06             Pilot1 pilot = new Pilot1("Kimi RaiKKonen");  
07             container.set(pilot);  
08             // QBE does not return any results  
09             ObjectSet result = container.get(new Pilot1("Kimi RaiKKonen"));  
10             System.out.println("Test QBE on class with member initialization in constructor");  
11             listResult(result);  
12     } catch (Exception ex) {  
13         System.out.println("System Exception: " + ex.getMessage());  
14     } finally {  
15         closeDatabase();  
16     }  
17 }  
18 }  
19 }
```

```

16 |     }
17 | }
18 L }
```

2. This example is similar to the previous one, but uses a class derived from the class in test1:

QBEEExample.java: test2

```

01 ┏━ private static void test2() {
02 |     ObjectContainer container = database();
03 ┏━ if (container != null) {
04 ┏━     try {
05 |         // Pilot1Derived derives the constructor with initialisation
06 |         Pilot1Derived pilot = new Pilot1Derived("Kimi Raikonen");
07 |         container.set(pilot);
08 |         // QBE does not return any results
09 |         ObjectSet result = container.get(new Pilot1Derived("Kimi Raikonen"));
10 |         System.out.println("Test QBE on class with member initialisation in ancestor
constructor");
11 |         listResult(result);
12 ┏━     } catch (Exception ex) {
13 |         System.out.println("System Exception: " + ex.getMessage());
14 ┏━     } finally {
15 |         closeDatabase();
16 |     }
17 | }
18 L }
```

3. This example uses QBE against a class with static member initialization:

QBEEExample.java: test3

```

01 ┏━ private static void test3() {
02 |     ObjectContainer container = database();
03 ┏━ if (container != null) {
04 ┏━     try {
05 |         // Pilot2 uses static initialisation of points member
06 |         Pilot2 pilot = new Pilot2("Kimi Raikonen");
```

```

07 |         container.set(pilot);
08 |         // QBE does not return any results
09 |         ObjectSet result = container.get(new Pilot2("Kimi Rai konnen"));
10 |         System.out.println("Test QBE on class with static member initialization");
11 |         listResult(result);
12 |     } catch (Exception ex) {
13 |         System.out.println("System Exception: " + ex.getMessage());
14 |     } finally {
15 |         closeDatabase();
16 |     }
17 | }
18 |

```

4. This example is similar to test3, but a derived class is used:

QBEEExample.java: test4

```

01 |     private static void test4() {
02 |         ObjectContainer container = database();
03 |         if (container != null) {
04 |             try {
05 |                 // Pilot2Derived is derived from class with static initialization of points member
06 |                 Pilot2Derived pilot = new Pilot2Derived("Kimi Rai konnen");
07 |                 container.set(pilot);
08 |                 // QBE does not return any results
09 |                 ObjectSet result = container.get(new Pilot2Derived("Kimi Rai konnen"));
10 |                 System.out.println("Test QBE on class derived from a class with static member
initialization");
11 |                 listResult(result);
12 |             } catch (Exception ex) {
13 |                 System.out.println("System Exception: " + ex.getMessage());
14 |             } finally {
15 |                 closeDatabase();
16 |             }
17 |         }
18 |

```

This revision (3) was last Modified 2007-09-09T10:28:56 by Tetyana.

# Pilot1

Pilot1.java

```
01 /* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */  
02  
03 package com.db4odoc.qbe;  
04  
05 public class Pilot1 {  
06     private String name;  
07  
08     private int points;  
09  
10    public Pilot1(String name) {  
11        this.name = name;  
12        this.points = 100;  
13    }  
14  
15    public String getName() {  
16        return name;  
17    }  
18  
19    public void setName(String name) {  
20        this.name = name;  
21    }  
22  
23    public int getPoints() {  
24        return points;  
25    }  
26  
27    public String toString() {  
28        return name + "/" + points;
```

```
29 |  }
30 |
31 \}
```

This revision (2) was last Modified 2007-09-09T10:23:05 by Tetyana.

# Pilot1Derived

Pilot1Derived.java

```
01 package com.db4odoc.qbe;
02
03 public class Pilot1Derived extends Pilot1 {
04 |
05     public Pilot1Derived(String name) {
06         super(name);
07         // TODO Auto-generated constructor stub
08     }
09 |
10 }
```

This revision (2) was last Modified 2007-09-09T10:24:00 by Tetyana.

# Pilot2

Pilot2.java

```
01/* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */  
02  
03 package com.db4odoc.qbe;  
04  
05public class Pilot2 {  
06     private String name;  
07  
08     private int points = 100;  
09  
10    public Pilot2(String name) {  
11        this.name = name;  
12    }  
13  
14    public String getName() {  
15        return name;  
16    }  
17  
18    public void setName(String name) {  
19        this.name = name;  
20    }  
21  
22    public int getPoints() {  
23        return points;  
24    }  
25  
26    public String toString() {  
27        return name + "/" + points;  
28    }
```

**29** |**30** ↴

This revision (2) was last Modified 2007-09-09T10:24:46 by Tetyana.

# Pilot2Derived

Pilot2Derived.java

```
1 package com.db4odoc.qbe;
2
3 public class Pilot2Derived extends Pilot2 {
4
5     public Pilot2Derived(String name) {
6         super(name);
7     }
8
9 }
```

This revision (2) was last Modified 2007-09-09T10:25:33 by Tetyana.

# Native Queries

## Contents

- [Concept](#)
- [Principle](#)
- [Simple Example](#)
- [Advanced Example](#)
- [Native Query Performance](#)

Wouldn't it be nice to pose queries in the programming language that you are using? Wouldn't it be nice if all your query code was 100% typesafe, 100% compile-time checked and 100% refactorable? Wouldn't it be nice if the full power of object-orientation could be used by calling methods from within queries? All mentioned above is achievable by using Native Queries or [LINQ](#)(if you are developing in .NET3.5)

Native queries are the main db4o query interface and they are the recommended way to query databases from your application for all platforms except .NET3.5 where LINQ is preferable. Because native queries simply use the semantics of your programming language, they are perfectly standardized and a safe choice for the future.

Native Queries are available for all platforms supported by db4o.

## Concept

---

The concept of native queries is taken from the following two papers:

- [Cook/Rosenberger, Native Queries for Persistent Objects, A Design White Paper](#)
- [Cook/Rai, Safe Query Objects: Statically Typed Objects as Remotely Executable Queries](#)

## Principle

---

Native Queries provide the ability to run one or more lines of code against all instances of a class. Native query expressions should return true to mark specific instances as part of the result set. db4o will attempt to [optimize native query](#) expressions and use [internal query processor](#) to run them against indexes and without instantiating actual objects, where this is possible.

## Simple Example

---

Let's look at how a simple native query will look like in some of the programming languages and dialects that db4o supports:

## Java5:

NQExample.java: primitiveQuery

```

1 private static void primitiveQuery(ObjectContainer container) {
2     List<Pilot> pilots = container.query(new Predicate<Pilot>() {
3         public boolean match(Pilot pilot) {
4             return pilot.getPoints() == 100;
5         }
6     });
7 }
```

## Java1.2-1.4:

PrimitiveExample.java: primitiveQuery

```

1 public static void primitiveQuery(ObjectContainer db) {
2     List<Pilot> pilots = db.query(new Predicate() {
3         public boolean match(Pilot pilot) {
4             return pilot.getPoints() == 100;
5         }
6     });
7 }
```

## Java1.1:

PrimitiveExample.java: primitiveQuery1

```

1 public static void primitiveQuery1(ObjectContainer db) {
2     List<Pilot> pilots = db.query(new PilotHundredPoints());
3 }
```

PilotHundredPoints.java

```

1 /* Copyright (C) 2004 - 2006 db4o objects Inc. http://www.db4o.com */
2 import com.db4o.query.Predicate;
3
4 public class PilotHundredPoints extends Predicate {
5     public boolean match(Pilot pilot) {
6         return pilot.getPoints() == 100;
7     }
8 }

```

A side note on the above syntax:

For all dialects without support for generics, Native Queries work by convention. A class that extends the Predicate class is expected to have a boolean #match() or #Match() method with one parameter to describe the class extent:

Java:

```
boolean match(Pilot candidate);
```

When using native queries, don't forget that modern integrated development environments (IDEs) can do all the typing work around the native query expression for you, if you use templates and autocompletion.

Here is how to configure a Native Query template with Eclipse 3.1:

From the menu, choose Window + Preferences + Java + Editor + Templates + New

As the name type "nq". Make sure that "java" is selected as the context on the right. Paste the following into the pattern field:

```
List <${extent}> list = db.query(new Predicate <${extent}> () {
    public boolean match(${extent} candidate){
        return true;
    }
});
```

Now you can create a native query with three keys: n + q + Control-Space.

Similar features are available in most modern IDEs.

For more information see [Native Query Syntax](#).

## Advanced Example

For complex queries, the native syntax is very precise and quick to write. Let's compare to a SODA query that finds all pilots with a given name or a score within a given range:

### NQExample.java: storePilots

```
1 private static void storePilots(ObjectContainer container) {
2 |     container.set(new Pilot("Michael Schumacher", 100));
3 |     container.set(new Pilot("Rubens Barrichello", 99));
4 | }
```

### NQExample.java: retrieveComplexSODA

```
01 private static void retrieveComplexSODA(ObjectContainer container) {
02 |     Query query = container.query();
03 |     query.constrain(Pilot.class);
04 |     Query pointQuery = query.descend("points");
05 |     query.descend("name").constrain("Rubens Barrichello").or(
06 |         pointQuery.constrain(new Integer(99)).greater().and(
07 |             pointQuery.constrain(new Integer(199))
08 |                 .smaller()));
09 |     ObjectSet result = query.execute();
10 |     listResult(result);
11 | }
```

## Native Query Performance

One drawback of native queries has to be pointed out: under the hood db4o tries to analyze native queries to convert them to SODA. This is not possible for all queries. For some queries it is very difficult to analyze the flowgraph. In this case db4o will have to instantiate some of the persistent objects to actually run the native query code. db4o will try to analyze parts of native query expressions to keep object instantiation to the minimum.

The development of the native query optimization processor will be an ongoing process in a close dialog with the db4o community. Feel free to contribute your results and your needs by providing feedback to our

[db4o forums.](#)

The current state of the query optimization process is detailed in the chapter on [Native Query Optimization](#)

With the current implementation, all above examples will run optimized, except for the "Arbitrary Code" example - we are working on it.

Note:

- on Java Native Query optimization requires bloat.jar, db4o-nqopt.jar and db4o-instrumentation.jar to be present in the classpath;
- on .NET Native Query optimization requires a reference to Db4obects.Db4o.Instrumentation.dll and Db4objects.Db4o.NativeQueries.dll in your project.

This revision (21) was last Modified 2008-04-14T15:42:10 by Tetyana.

# Native Query Syntax

Native queries give you a choice of several implementations. This topic will explain how and where each implementation should be used.

More Reading:

- [Java Syntax](#)
- [.NET Syntax](#)

This revision (1) was last Modified 2007-04-29T18:31:35 by Tetyana.

# Java Syntax

This topic applies to Java version only.

1. The following example shows how to use Native Query to retrieve all the objects of the specified type. This syntax can be used with or without generics.

NQSyntaxExamples.java: querySyntax1

```

01 ①②private static void querySyntax1() {
02 |     ObjectContainer container = database();
03 |     if (container != null) {
04 |         try {
05 |             List<Pilot> result = container.query(Pilot.class);
06 |             container.ext().configure().freespace();
07 |             listResult(result);
08 |         } catch (Exception ex) {
09 |             System.out.println("System Exception: " + ex.getMessage());
10 |         } finally {
11 |             closeDatabase();
12 |         }
13 |     }
14 }

```

2. In this example an anonymous predicate class is used to specify the query parameter:

NQSyntaxExamples.java: querySyntax2

```

01 ①②private static void querySyntax2() {
02 |     ObjectContainer container = database();
03 |     if (container != null) {
04 |         try {
05 |             List<Pilot> result = container.query(new Predicate<Pilot>() {
06 |                 public boolean match(Pilot pilot) {
07 |                     // each Pilot is included in the result
08 |                     return true;
09 |                 }

```

```

10 |         });
11 |         listResult(result);
12 |     } catch (Exception ex) {
13 |         System.out.println("System Exception: " + ex.getMessage());
14 |     } finally {
15 |         closeDatabase();
16 |     }
17 | }
18 |

```

3. This example shows how to use NQ to sort the query results; anonymous predicate and anonymous comparator are used:

#### NQSyntaxExamples.java: querySyntax3

```

01 | private static void querySyntax3() {
02 |     ObjectContainer container = database();
03 |     if (container != null) {
04 |         try {
05 |             List<Pilot> result = container.query(new Predicate<Pilot>() {
06 |                 public boolean match(Pilot pilot) {
07 |                     // each Pilot is included in the result
08 |                     return true;
09 |                 }
10 |             }, new Comparator<Pilot>() {
11 |                 public int compare(Pilot pilot1, Pilot pilot2) {
12 |                     return pilot1.getPoints() - pilot2.getPoints();
13 |                 }
14 |             });
15 |             listResult(result);
16 |         } catch (Exception ex) {
17 |             System.out.println("System Exception: " + ex.getMessage());
18 |         } finally {
19 |             closeDatabase();
20 |         }
21 |

```

22 L }

4. The following example shows a Native Query using external comparator and predicate. This can be useful when comparator and predicate are widely used and logically do not belong to the querying class:

NQSyntaxExamples.java: PilotPredicate

```

1 ①②③private static class PilotPredicate extends Predicate<Pilot> {
2 ④⑤    public boolean match(Pilot pilot) {
3 |        // each Pilot is included in the result
4 |        return true;
5 |    }
6 L }
```

NQSyntaxExamples.java: PilotComparator

```

1 ①②③private static class PilotComparator implements Comparator<Pilot> {
2 ④⑤    public int compare(Pilot pilot1, Pilot pilot2) {
3 |        return pilot1.getPoints() - pilot2.getPoints();
4 |    }
5 L }
```

NQSyntaxExamples.java: querySyntax4

```

01 ①②③private static void querySyntax4() {
02 |    ObjectContainer container = database();
03 ④⑤    if (container != null) {
04 ⑥⑦        try {
05 |            List<Pilot> result = container.query(new PilotPredicate(),
06 |                                              new PilotComparator());
07 |            listResult(result);
08 ⑨⑩        } catch (Exception ex) {
09 |            System.out.println("System Exception: " + ex.getMessage());
10 ⑪⑫        } finally {
11 |            closeDatabase();
12 |        }
13 |    }
14 |}
```

```

13 |     }
14 | }
```

5.

6. In java versions without generics syntax is similar:

NQSyntaxExamples.java: querySyntax5

```

01 | private static void querySyntax5() {
02 |     ObjectContainer container = database();
03 |     if (container != null) {
04 |         try {
05 |             List result = container.query(new Predicate() {
06 |                 public boolean match(Object obj) {
07 |                     // each Pilot is included in the result
08 |                     if (obj instanceof Pilot) {
09 |                         return true;
10 |                     }
11 |                     return false;
12 |                 }
13 |                 , new Comparator() {
14 |                     public int compare(Object object1, Object object2) {
15 |                         return ((Pilot) object1).getPoints()
16 |                             - ((Pilot) object2).getPoints();
17 |                     }
18 |                 });
19 |             listResult(result);
20 |         } catch (Exception ex) {
21 |             System.out.println("System Exception: " + ex.getMessage());
22 |         } finally {
23 |             closeDatabase();
24 |         }
25 |     }
26 | }
```

7. For java versions that do not provide Comparator interface (&lt;JDK1.2) db4o provides QueryComparator

interface with the same functionality:

NQSyntaxExamples.java: querySyntax6

```

01  private static void querySyntax6() {
02 |     // this example will only work with java versions without
03 |     // generics support
04 |     ObjectContainer container = database();
05 |     if (container != null) {
06 |         try {
07 |             List result = container.query(new Predicate() {
08 |                 public boolean match(Object obj) {
09 |                     // each Pilot is included in the result
10 |                     if (obj instanceof Pilot) {
11 |                         return true;
12 |                     }
13 |                     return false;
14 |                 }
15 |             }, new QueryComparator() {
16 |                 public int compare(Object pilot1, Object pilot2) {
17 |                     return ((Pilot) pilot1).getPoints()
18 |                         - ((Pilot) pilot2).getPoints();
19 |                 }
20 |             });
21 |             listResult(result);
22 |         } catch (Exception ex) {
23 |             System.out.println("System Exception: " + ex.getMessage());
24 |         } finally {
25 |             closeDatabase();
26 |         }
27 |     }
28 | }
```

This revision (3) was last Modified 2007-04-29T19:08:52 by Tetyana.

# .NET Syntax

This topic applies to .NET version only.

In .NET version of db4o we need to distinguish NQ syntax for c# (.NET1 and .NET2) and Visual Basic.

This revision (6) was last Modified 2007-09-16T16:46:51 by Tetyana.

# SODA Query

The SODA query API is db4o's low level querying API, allowing direct access to nodes of query graphs. Since SODA uses strings to identify fields, it is neither perfectly typesafe nor compile-time checked and it also is quite verbose to write.

For most applications [Native Queries](#) or [LINQ](#) will be the better querying interface. However there can be applications where dynamic generation of queries is required.

SODA is also an underlying db4o querying mechanism: all other query syntaxes are translated to SODA under the hood:

- [Query By Example](#) is translated to SODA with a single `constrain` call
- [Native Queries](#) use bytecode and IL analysis to [convert](#) to SODA
- [LINQ](#) also uses [IL analysis](#)

Understanding SODA will provide you with a better understanding of db4o in the whole and will help to write more performant queries and applications.

More Reading:

- [Building SODA Queries](#)
- [SODA Query Graph](#)
- [SODA Query API](#)
- [SODA Query Engine](#)

This revision (16) was last Modified 2008-03-02T15:33:23 by Tetyana.

# Building SODA Queries

Let's see how simple QBE queries are expressed with SODA. A new Query object is created through the `#query()` method of the ObjectContainer and we can add Constraint instances to it. To find all Pilot instances, we constrain the query with the Pilot class object.

QueryExample.java: retrieveAllPilots

```

01  private static void retrieveAllPilots() {
02 |     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
03 |     try {
04 |         Query query = container.query();
05 |         query.constraint(Pilot.class);
06 |         ObjectSet result = query.execute();
07 |         listResult(result);
08 |     } finally {
09 |         container.close();
10 |     }
11 }
```

Basically, we are exchanging our 'real' prototype for a meta description of the objects we'd like to hunt down: a **query graph** made up of query nodes and constraints. A query node is a placeholder for a candidate object, a constraint decides whether to add or exclude candidates from the result.

Our first simple graph looks like this.



We're just asking any candidate object (here: any object in the database) to be of type Pilot to aggregate our result.

To retrieve a pilot by name, we have to further constrain the candidate pilots by descending to their name field and constraining this with the respective candidate String.

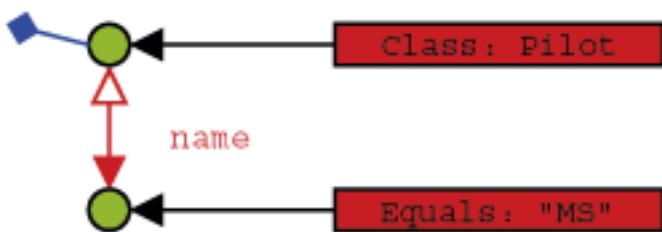
QueryExample.java: retrievePilotByName

```

1 private static void retrievePilotByName(Obj ectContainer container) {
2     Query query = container.query();
3     query.constraint(Pilot.class);
4     query.descend("name").constraint("Michael Schumacher");
5     Obj ectSet result = query.execute();
6     listResult(result);
7 }

```

What does 'descend' mean here? Well, just as we did in our 'real' prototypes, we can attach constraints to child members of our candidates.



So a candidate needs to be of type Pilot and have a member named 'name' that is equal to the given String to be accepted for the result.

Note that the class constraint is not required: If we left it out, we would query for all objects that contain a 'name' member with the given value. In most cases this will not be the desired behavior, though.

Finding a pilot by exact points is analogous. We just have to cross the Java primitive/object divide.

#### QueryExample.java: retrievePilotByExactPoints

```

1 private static void retrievePilotByExactPoints(
2     Obj ectContainer container) {
3     Query query = container.query();
4     query.constraint(Pilot.class);
5     query.descend("points").constraint(new Integer(100));
6     Obj ectSet result = query.execute();
7     listResult(result);
8 }

```

This revision (2) was last Modified 2008-03-02T14:25:40 by Tetyana.

# SODA Query Graph

SODA allows to create a query graph of any complexity by joining field object constraints. SODA usage is very generic and can be applied to any objects and conditions. The following 5 steps can be used (all steps are optional and can be repeated logically):

- create a root of a query object

Java:

```
Query queryRootNode = db.query();
```

- add constraints to any node anywhere

Java:

```
queryRootNode.constrain(Foo.class);
```

- navigate from any query node to any subordinate node

Java:

```
Query barNode = queryRootNode.descend("bar");
```

- add further constraints to any node

Java:

```
Constraint barConstraint = barNode.constrain(5);
```

- set the evaluation mode of a node

Java:

```
barConstraint().greater();
```

The API is very powerful with a small number of method calls.

The "backward" order to add constraints first and to specify the evaluation mode as a second step allows plugging complex objects into a query.

This revision (2) was last Modified 2008-03-02T14:48:05 by Tetyana.

# SODA Query API

## Contents

- [Not](#)
- [And](#)
- [Or](#)
- [Greater, Smaller, Equal <=>](#)
- [Using Default Values](#)
- [String Comparisons](#)
  - [Like](#)
  - [startsWith, endsWith](#)
  - [Case Insensitive Queries](#)
- [Contains](#)
- [Identity Comparison](#)
- [Sorting Results](#)

There are occasions when we don't want to query for exact field values, but rather for value ranges, objects not containing given member values, etc. This functionality is provided by the Constraint API.

## Not

First, let's negate a query to find all pilots who are not Michael Schumacher:

### QueryExample.java: retrieveByNegation

```

1 ①②private static void retrieveByNegation(ObjectContainer container) {
2     Query query = container.query();
3     query.constraint(Pilot.class);
4     query.descend("name").constraint("Michael Schumacher").not();
5     ObjectSet result = query.execute();
6     listResult(result);
7 }
```

## And

Where there is negation, the other boolean operators can't be too far.

### QueryExample.java: retrieveByConjunction

```

01 private static void retrieveByConjunction(Obj ectContainer container) {
02     Query query = container.query();
03     query.constrain(Pilot. class);
04     Constraint constr = query.descend("name").constrain(
05         "Michael Schumacher");
06     query.descend("points").constrain(new Integer(99))
07         .and(constr);
08     Obj ectSet result = query.execute();
09     listResult(result);
10 }

```

## Or

QueryExample.java: retrieveByDisjunction

```

1 private static void retrieveByDisjunction(Obj ectContainer container) {
2     Query query = container.query();
3     query.constrain(Pilot. class);
4     Constraint constr = query.descend("name").constrain(
5         "Michael Schumacher");
6     query.descend("points").constrain(new Integer(99)).or(constr);
7     Obj ectSet result = query.execute();
8     listResult(result);
9 }

```

## Greater, Smaller, Equal <>=

We can also constrain to a comparison with a given value.

Return pilots with more than 99 points:

QueryExample.java: retrieveByComparison

```

1 private static void retrieveByComparison(Obj ectContainer container) {
2     Query query = container.query();

```

```

3 |     query.constrain(Pilot.class);
4 |     query.descend("points").constrain(new Integer(99)).greater();
5 |     ObjectSet result = query.execute();
6 |     listResult(result);
7 | }

```

Return pilots with 99 or more points:

#### QueryExample.java: retrieveByEqualComparison

```

1  private static void retrieveByEqualComparison(ObjectContainer container) {
2 |     Query query = container.query();
3 |     query.constrain(Pilot.class);
4 |     query.descend("points").constrain(new Integer(99)).greater().equal();
5 |     ObjectSet result = query.execute();
6 |     listResult(result);
7 | }

```

## Using Default Values

The query API also allows to query for field default values.

#### QueryExample.java: retrieveByDefaultFieldValue

```

01 private static void retrieveByDefaultFieldValue(
02  ObjectContainer container) {
03 |     Pilot somebody = new Pilot("Somebody else", 0);
04 |     container.set(somebody);
05 |     Query query = container.query();
06 |     query.constrain(Pilot.class);
07 |     query.descend("points").constrain(new Integer(0));
08 |     ObjectSet result = query.execute();
09 |     listResult(result);
10 |     container.delete(somebody);
11 | }

```

# String Comparisons

## Like

This is an equivalent to SQL "like" operator:

SodaExample.java: testLike

```

01  public static void testLike() {
02 |     new File(DB40_FILE_NAME).delete();
03 |     ObjectContainer container = database();
04 |     if (container != null) {
05 |         try {
06 |             Pilot pilot = new Pilot("Test Pilot1", 100);
07 |             container.set(pilot);
08 |             pilot = new Pilot("Test Pilot2", 102);
09 |             container.set(pilot);
10 |
11 |             // Simple query
12 |             Query query1 = container.query();
13 |             query1.constrain(Pilot.class);
14 |             query1.descend("name").constrain("est");
15 |             ObjectSet result = query1.execute();
16 |             listResult(result);
17 |
18 |             // Like query
19 |             Query query2 = container.query();
20 |             query2.constrain(Pilot.class);
21 |             // All pilots with the name containing "est" will be retrieved
22 |             query2.descend("name").constrain("est").like();
23 |             result = query2.execute();
24 |             listResult(result);
25 |         } catch (Db4oException ex) {
26 |             System.out.println("Db4o Exception: " + ex.getMessage());

```

```

27 |     } catch (Exception ex) {
28 |         System.out.println("System Exception: " + ex.getMessage());
29 |     } finally {
30 |         closeDatabase();
31 |     }
32 | }
33 |

```

## startsWith, endsWith

Compares a beginning or ending of a string:

SodaExample.java: testStartsEnds

```

01 | public static void testStartsEnds() {
02 |     new File(DB40_FILE_NAME).delete();
03 |     ObjectContainer container = database();
04 |     if (container != null) {
05 |         try {
06 |             Pilot pilot = new Pilot("Test Pilot0", 100);
07 |             container.set(pilot);
08 |             pilot = new Pilot("Test Pilot1", 101);
09 |             container.set(pilot);
10 |             pilot = new Pilot("Test Pilot2", 102);
11 |             container.set(pilot);
12 |
13 |             Query query = container.query();
14 |             query.constraint(Pilot.class);
15 |             query.descend("name").constraint("T0").endsWith(false).not();
16 |             // query.descend("name").constraint("Pil").startsWith(true);
17 |             ObjectSet result = query.execute();
18 |             listResult(result);
19 |         } catch (Db4oException ex) {
20 |             System.out.println("Db4o Exception: " + ex.getMessage());
21 |         } catch (Exception ex) {

```

```

22 |     System.out.println("System Exception: " + ex.getMessage());
23 | } finally {
24 |     closeDatabase();
25 | }
26 | }
27 | }
```

## Case Insensitive Queries

By default all string querying functions use case-sensitive comparison. `startsWith` and `endsWith` allow to switch between comparison modes using a parameter. However, if you need a case-insensitive comparison for `like`, `equals` or `contains` queries, it is recommended to use [Native Queries](#) as SODA does not provide such an option.

## Contains

---

Allows to retrieve objects constraining by included value (collection). If applied to a string will behave as "Like".

SodaExample.java: testContains

```

01 | public static void testContains() {
02 |     new File(DB40_FILE_NAME).delete();
03 |     ObjectContainer container = database();
04 |     if (container != null) {
05 |         try {
06 |             ArrayList list = new ArrayList();
07 |             Pilot pilot1 = new Pilot("Test 1", 1);
08 |             list.add(pilot1);
09 |             Pilot pilot2 = new Pilot("Test 2", 2);
10 |             list.add(pilot2);
11 |             Team team = new Team("Ferrari", list);
12 |             container.set(team);
13 |
14 |             Query query = container.query();
15 |             query.constrain(Team.class);
```

```

16 |         query.descend("pilots").constraint(pilot2).contains();
17 |         ObjectSet result = query.execute();
18 |         listResult(result);
19 |     } catch (Db4oException ex) {
20 |         System.out.println("Db4o Exception: " + ex.getMessage());
21 |     } catch (Exception ex) {
22 |         System.out.println("System Exception: " + ex.getMessage());
23 |     } finally {
24 |         closeDatabase();
25 |     }
26 | }
27 |

```

## Identity Comparison

db4o database identity can also be used as a constraint. In this case only objects with the same database instance will be retrieved:

SodaExample.java: testIdentity

```

01 | public static void testIdentity() {
02 |     new File(DB4O_FILE_NAME).delete();
03 |     ObjectContainer container = database();
04 |     if (container != null) {
05 |         try {
06 |             Pilot pilot = new Pilot("Test Pilot1", 100);
07 |             Car car = new Car("BMW", pilot);
08 |             container.set(car);
09 |             // Change the name, the pilot instance stays the same
10 |             pilot.setName("Test Pilot2");
11 |             // create a new car
12 |             car = new Car("Ferrari", pilot);
13 |             container.set(car);
14 |
15 |             // Simple Query:

```

```

16 |         Query query1 = container.query();
17 |         query1.constrain(Car.class);
18 |         query1.descend("_pilot").constrain(pilot);
19 |         ObjectSet result = query1.execute();
20 |         listResult(result);
21 |
22 |         // identity query:
23 |         Query query2 = container.query();
24 |         query2.constrain(Car.class);
25 |         // All cars having pilot with the same database identity
26 |         // will be retrieved. As we only created Pilot object once
27 |         // it should mean all car objects
28 |         query2.descend("_pilot").constrain(pilot).identity();
29 |         result = query2.execute();
30 |         listResult(result);
31 |     } catch (Db4oException ex) {
32 |         System.out.println("Db4o Exception: " + ex.getMessage());
33 |     } catch (Exception ex) {
34 |         System.out.println("System Exception: " + ex.getMessage());
35 |     } finally {
36 |         closeDatabase();
37 |     }
38 | }
39 | }
```

## Sorting Results

It is also possible to have db4o sort the results.

QueryExample.java: retrieveSorted

```

01 |     private static void retrieveSorted(ObjectContainer container) {
02 |         Query query = container.query();
03 |         query.constrain(Pilot.class);
04 |         query.descend("name").orderAscending();
```

```
05 |     ObjectSet result = query.execute();
06 |     listResult(result);
07 |     query.descend("name").orderDescending();
08 |     result = query.execute();
09 |     listResult(result);
10 | }
```

All these techniques can be combined arbitrarily, of course. Please try it out. There still may be cases left where the predefined query API constraints may not be sufficient - don't worry, you can always let db4o run any arbitrary code that you provide in an Evaluation. Evaluations will be discussed in a [Evaluations chapter](#).

This revision (1) was last Modified 2008-03-02T14:36:11 by Tetyana.

# SODA Query Engine

## Contents

- [Inheritance](#)

Query Processor operates in two stages:

1. In a first stage it creates a tree of candidate object IDs using an index. The Best index is searched, which means a field index for field constraints (if available). Field index allows to create often smaller candidate tree, already filtered on some criteria. If the best index is not found, class index is used to create a candidate tree of all instances of matching class or classes.
2. In a second stage, all candidates are run against the SODA processor to run all constraints against all objects, whether the field of a constraint is indexed or not.

The first stage of the query processor operates directly on BTrees. BTrees are used for class indexes (always), and field indexes (configurable).

BTree algebra may create unions and intersections (and, or, greater, smaller) between BTree ranges, working with pointers into BTrees without ever having to scan through all index entries. A BTree node points to the object's file positions through object ID.

Query processing starts from evaluating leaf nodes of the query graph and then going on to the top level filtering or joining the results.

For a simple query:

```
query.constrain(Pilot.class);
```

class index will be used to get ID's of all Pilot objects.

For a more complex query:

```
query.constrain(Pilot.class);
```

```
Constraint constr = query.descend("name").constrain("Michael Schumacher");
```

```
query.descend("points").constrain(new Integer(100)).and(constr);
```

In case there are indexes on "name" and "points" fields, Pilot candidates will be created from "points" indexes having value of 100 and "name" indexes with value "Michael Schumacher". With BTree indexes

this search will be really fast. If there is no index all Pilot IDs will be used as candidates. After the candidates are collected all existing constraints will be tested against them to filter out results that do not match the criteria.

## Inheritance

---

In the case of inherited classes or interfaces:

- querying against parent class or interface will include results for all subclasses/implementations;
- field indexes can be used and should be defined against the parent class fields

This revision (1) was last Modified 2008-03-02T15:08:25 by Tetyana.

# LINQ

Db4o LINQ syntax is provided for .NET developers and aimed to make writing db4o code even more native and effortless. If you are already familiar with LINQ syntax, you can just start writing LINQ to query db4o. Otherwise you may want to familiarize yourself with LINQ resources on [MSDN](#).

Further reading:

- [Simple Example](#)
- [Type Safety](#)
- [Optimization](#)
- [LINQ Collection](#)

This revision (3) was last Modified 2008-03-02T14:15:37 by Tetyana.

# Simple Example

All that you need to enable LINQ in your project is to add reference to Db4objects.Db4o.Linq.dll and use it in your class:

Let's start from looking at some simple examples. First of all - fill up the database with some objects:

And use LINQ syntax to select Pilots by name and points:

We use our db4o database as a datasource, the rest of the syntax is typical for LINQ queries. Using Object-Oriented language for creating queries gives us an advantage of creating very flexible select criteria and provides a benefit of compile-checked code.

This revision (4) was last Modified 2008-03-15T17:33:42 by Tetyana.

# Type Safety

In the previous example we used generic types for our query, constraining on only one type of objects. However LINQ is flexible enough to allow querying across types:

We have both Car and Pilot objects that can correspond the selected criteria and - yes, we will get a mixed list of both. This feature is potentially dangerous as in the development cycle you may lose control of the objects that answer the criteria and get unexpected results. Be aware and use untyped LINQ queries with caution.

This revision (1) was last Modified 2008-02-12T06:22:30 by Tetyana.

# Optimization

Those of you who are familiar with db4o query processor may already suspect that LINQ queries as well as NQ and QueryByExample queries are converted to [SODA Query](#) under the hood. And that is true. The downside of this is that some of the queries cannot be converted to SODA and will run against all instances of matching class objects. Some of the cases when conversion is not possible:

- there is no SODA alternative available
- constraints on aggregates ( $\text{avg}(\text{List}) = \text{constant}$ )
- constraints to values that are not available before query executions (`Pilot.Points = Pilot.Name.Length`)

This revision (3) was last Modified 2008-03-15T17:42:12 by Tetyana.

# Query Modes

What is the best way to process queries? How to get the optimum performance for your application needs?

Those of you, who have dealt with query time and memory constraints (situations, when query result is bigger than the memory available, or when query time is longer than the whole functional operation) should know how searching a suitable solution might affect the whole application design and implementation.

Luckily db4o takes most of the trouble for itself. There are 3 query modes allowing to fine tune the balance between speed, memory consumption and availability of the results:

- immediate
- lazy
- snapshot

The query mode can be set using:

Java:

```
configuration.queries().evaluationMode(QueryEvaluationMode)
```

Let's look at each of them in more detail.

More Reading:

- [Immediate Queries](#)
- [Lazy Queries](#)
- [Snapshot Queries](#)

This revision (5) was last Modified 2007-04-23T06:31:11 by Tetyana.

# Immediate Queries

## Contents

- [Immediate Mode Pros And Cons](#)

This is the default query mode: the whole query result is evaluated upon query execution and object IDs list is produced as a result.

### QueryModesExample.java: testImmediateQueries

```

01  private static void testImmediateQueries() {
02 |     System.out
03 |         .println("Testing query performance on 10000 pilot objects in Immediate mode");
04 |     fillUpDB(10000);
05 |     Configuration configuration = Db4o.newConfiguration();
06 |     configuration.queries().evaluationMode(QueryEvaluationMode.IMMEDIATE);
07 |     ObjectContainer container = Db4o.openFile(configuration, DB4O_FILE_NAME);
08 |     try {
09 |         QueryStats stats = new QueryStats();
10 |         stats.connect(container);
11 |         Query query = container.query();
12 |         query.constrain(Pilot.class);
13 |         query.descend("points").constrain(99).greater();
14 |         query.execute();
15 |         long executionTime = stats.executionTime();
16 |         System.out.println("Query execution time: "
17 |             + executionTime);
18 |     } finally {
19 |         container.close();
20 |     }
21 }
```

Obviously object evaluation takes some time and in a case of big resultsets you will have to wait for a long time before the first result will be returned. This is especially unpleasant in a client-server setup, when query processing can block the server for seconds or even minutes.

This mode makes the whole objects result set available at once - ID list is built based on the committed state in the database. As soon as a result is delivered it won't be changed neither by changes in current transaction neither by committed changes from another transactions.

Note, that resultset contains only references to objects, you were querying for, which means that if an object field has

changed by the time of the actual object retrieval from the object set - you will get the new field value:

#### QueryModesExample.java: testImmediateChanged

```

01  private static void testImmediateChanged() {
02 |     System.out
03 |         .println("Testing immediate mode with field changes");
04 |     fillUpDB(10);
05 |     Configuration configuration = Db4o.newConfiguration();
06 |     configuration.queries().evaluationMode(QueryEvaluationMode.IMMEDIATE);
07 |     ObjectContainer container = Db4o.openFile(configuration, DB4O_FILE_NAME);
08  |     try {
09 |         Query query1 = container.query();
10 |         query1.constrain(Pilot.class);
11 |         query1.descend("points").constrain(5).smaller();
12 |         ObjectSet result1 = query1.execute();
13 |
14 |         // change field
15 |         Query query2 = container.query();
16 |         query2.constrain(Pilot.class);
17 |         query2.descend("points").constrain(2);
18 |         ObjectSet result2 = query2.execute();
19 |         Pilot pilot2 = (Pilot) result2.get(0);
20 |         pilot2.addPoints(22);
21 |         container.set(pilot2);
22 |         listResult(result1);
23  |     } finally {
24 |         container.close();
25 |     }
26 }
```

## Immediate Mode Pros And Cons

Pros:

- If the query is intended to iterate through the entire resulting ObjectSet, this mode will be slightly faster than the others.
- The query will process without intermediate side effects from changed objects (by the caller or by other transactions).

Cons:

- Query processing can block the server for a long time.
- In comparison to the other modes it will take longest until the first results are returned.

## Immediate Queries

- The ObjectSet will require a considerate amount of memory to hold the IDs of all found objects.

This revision (10) was last Modified 2006-12-04T13:21:56 by Tetyana.

# Lazy Queries

## Contents

- Pros and Cons for Lazy Queries

In Lazy Querying mode objects are not evaluated at all, instead of this an iterator is created against the best index found. Further query processing (including all index processing) will happen when the user application iterates through the resulting ObjectSet. This allows you to get the first query results almost immediately.

QueryModesExample.java: testLazyQueries

```

01  private static void testLazyQueries() {
02 |     System.out
03 |         .println("Testing query performance on 10000 pilot objects in Lazy mode");
04 |     fillUpDB(10000);
05 |     Configuration configuration = Db4o.newConfiguration();
06 |     configuration.queries().evaluationMode(QueryEvaluationMode.LAZY);
07 |     ObjectContainer container = Db4o.openFile(configuration, DB4O_FILE_NAME);
08  |     try {
09 |         QueryStats stats = new QueryStats();
10 |         stats.connect(container);
11 |         Query query = container.query();
12 |         query.constrain(Pilot.class);
13 |         query.descend("points").constrain(99).greater();
14 |         query.execute();
15 |         long executionTime = stats.executionTime();
16 |         System.out.println("Query execution time: "
17 |             + executionTime);
18  |     } finally {
19 |         container.close();
20 |     }
21 }
```

In addition to the very fast execution this method also ensures very small memory consumption, as lazy queries do not need an intermediate representation as a set of IDs in memory. With this approach a lazy query ObjectSet does not have to cache a single object or ID. The memory consumption for a query is practically zero, no matter how large the resultset is going to be.

There are some interesting effects appearing due to the fact that the objects are getting evaluated only on a request. It means that all the committed modifications from the other transactions and uncommitted modifications from the same transaction will be taken into account when delivering the result objects:

## QueryModesExample.java: testLazyConcurrent

```

01  private static void testLazyConcurrent() {
02 |     System.out
03 |         .println("Testing lazy mode with concurrent modifications");
04 |     fillUpDB(10);
05 |     Configuration configuration = Db4o.newConfiguration();
06 |     configuration.queries().evaluationMode(QueryEvaluationMode.LAZY);
07 |     ObjectContainer container = Db4o.openFile(configuration, DB4O_FILE_NAME);
08  |     try {
09 |         Query query1 = container.query();
10 |         query1.constrain(Pilot.class);
11 |         query1.descend("points").constrain(5).smaller();
12 |         ObjectSet result1 = query1.execute();
13 |
14 |         Query query2 = container.query();
15 |         query2.constrain(Pilot.class);
16 |         query2.descend("points").constrain(1);
17 |         ObjectSet result2 = query2.execute();
18 |         Pilot pilotToDelete = (Pilot) result2.get(0);
19 |         System.out.println("Pilot to be deleted: "
20 |             + pilotToDelete);
21 |         container.delete(pilotToDelete);
22 |         Pilot pilot = new Pilot("Tester", 2);
23 |         System.out.println("Pilot to be added: " + pilot);
24 |         container.set(pilot);
25 |
26 |         System.out
27 |             .println("Query result after changing from the same transaction");
28 |         listResult(result1);
29  |     } finally {
30 |         container.close();
31 |     }
32 }
```

# Pros and Cons for Lazy Queries

Pros:

- The call to `Query.execute()` will return very fast. First results can be made available to the application before the query is fully processed.
- A query will consume hardly any memory at all because no intermediate ID representation is ever created.

Cons:

- Lazy queries check candidates when iterating through the resulting `ObjectSet`. In doing so the query processor takes changes into account that may have happened since the `Query.execute()` call: committed changes from other transactions, **and uncommitted changes from the calling transaction**. There is a wide range of possible side effects:
  - The underlying index may have changed.
  - Objects themselves may have changed in the meanwhile.
  - There even is a chance of creating an endless loop. If the caller iterates through the `ObjectSet` and changes each object in a way that it is placed at the end of the index, the same objects can be revisited over and over.

**In lazy mode it can make sense to work in a way one would work with collections to avoid concurrent modification exceptions.** For instance one could iterate through the `ObjectSet` first and store all the objects to a temporary collection representation before changing objects and storing them back to db4o.

- Some method calls against a lazy `ObjectSet` will require the query processor to create a snapshot or to evaluate the query fully. An example of such a call is `ObjectSet.size()`.

Lazy mode can be an excellent choice for single transaction read use, to keep memory consumption as low as possible.

This revision (7) was last Modified 2007-11-21T17:18:40 by Eric Falsken.

# Snapshot Queries

## Contents

- Pros and Cons for Snapshot Queries

Snapshot mode allows you to get the advantages of the Lazy queries avoiding their side effects. When query is executed, the query processor chooses the best indexes, does all index processing and creates a snapshot of the index at this point in time. Non-indexed constraints are evaluated lazily when the application iterates through the ObjectSet resultset of the query.

QueryModesExample.java: testSnapshotQueries

```

01  private static void testSnapshotQueries() {
02 |     System.out
03 |         .println("Testing query performance on 10000 pilot objects in Snapshot mode");
04 |     fillUpDB(10000);
05 |     Configuration configuration = Db4o.newConfiguration();
06 |     configuration.queries().evaluationMode(QueryEvaluationMode.SNAPSHOT);
07 |     ObjectContainer container = Db4o.openFile(configuration, DB4O_FILE_NAME);
08  |     try {
09 |         QueryStats stats = new QueryStats();
10 |         stats.connect(container);
11 |         Query query = container.query();
12 |         query.constrain(Pilot.class);
13 |         query.descend("points").constrain(99).greater();
14 |         query.execute();
15 |         long executionTime = stats.executionTime();
16 |         System.out.println("Query execution time: "
17 |             + executionTime);
18  |     } finally {
19 |         container.close();
20 |     }
21 }
```

Snapshot queries ensure better performance than Immediate queries, but the performance will depend on the size of the resultset.

As the snapshot of the results is kept in memory the result set is not affected by the changes from the caller or from another transaction (compare the results of this code snippet to the one from [Lazy Queries](#) topic):

## QueryModesExample.java: testSnapshotConcurrent

```

01  private static void testSnapshotConcurrent() {
02 |     System.out
03 |         .println("Testing snapshot mode with concurrent modifications");
04 |     fillUpDB(10);
05 |     Configuration configuration = Db4o.newConfiguration();
06 |     configuration.queries().evaluationMode(QueryEvaluationMode.SNAPSHOT);
07 |     ObjectContainer container = Db4o.openFile(configuration, DB4O_FILE_NAME);
08  |     try {
09 |         Query query1 = container.query();
10 |         query1.constrain(Pilot.class);
11 |         query1.descend("points").constrain(5).smaller();
12 |         ObjectSet result1 = query1.execute();
13 |
14 |         Query query2 = container.query();
15 |         query2.constrain(Pilot.class);
16 |         query2.descend("points").constrain(1);
17 |         ObjectSet result2 = query2.execute();
18 |         Pilot pilotToDelete = (Pilot) result2.get(0);
19 |         System.out.println("Pilot to be deleted: "
20 |             + pilotToDelete);
21 |         container.delete(pilotToDelete);
22 |         Pilot pilot = new Pilot("Tester", 2);
23 |         System.out.println("Pilot to be added: " + pilot);
24 |         container.set(pilot);
25 |
26 |         System.out
27 |             .println("Query result after changing from the same transaction");
28 |         listResult(result1);
29  |     } finally {
30 |         container.close();
31 |     }
32 | }

```

## Pros and Cons for Snapshot Queries

Pros:

- Index processing will happen without possible side effects from changes made by the caller or by other transaction.

- Since index processing is fast, a server will not be blocked for a long time.

Cons:

- The entire candidate index will be loaded into memory. It will stay there until the query ObjectSet is garbage collected. In a C/S setup, the memory will be used on the server side

Client/Server applications with the risk of concurrent modifications should prefer Snapshot mode to avoid side effects from other transactions.

This revision (6) was last Modified 2007-11-21T17:34:25 by Eric Falsken.

# SODA Evaluations

*Evaluations* provide user-defined custom constraints as an instrument to run any arbitrary code in a SODA query. It gives you maximum power over the query execution, but certainly has its own drawbacks. Let's have a closer look.

More Reading:

- [Evaluation API](#)
- [Using Evaluations](#)
- [Drawbacks](#)

This revision (5) was last Modified 2006-11-15T11:32:33 by Tetyana.

# Evaluation API

The evaluation API consists of two interfaces, *Evaluation* and *Candidate*. Evaluation implementations are implemented by the user and injected into a query. During a query, they will be called from db4o with a candidate instance in order to decide whether to include it into the current (sub-)result.

The Evaluation interface contains a single method only:

```
Java: public void evaluate(Candidate candidate)
```

This will be called by db4o to check whether the object encapsulated by this candidate should be included into the current candidate set.

The Candidate interface provides three methods:

```
Java:
```

```
public Object getObject()  
  
public void include(boolean flag)  
  
public ObjectContainer objectContainer()
```

An Evaluation implementation may call `getObject()` to retrieve the actual object instance to be evaluated, it may call `include()` to instruct db4o whether or not to include this object in the current candidate set, and finally it may access the current database directly by calling `objectContainer()`.

This revision (6) was last Modified 2006-11-13T15:31:52 by Tetyana.

# Using Evaluations

For a simple example, let's take a Car class keeping a history of SensorReadout instances in a List member. Now imagine that we wanted to retrieve all cars that have assembled an even number of history entries. A quite contrived and seemingly trivial example, however, it gets us into trouble: Collections are transparent to the query API, it just 'looks through' them at their respective members.

So how can we get this done? Let's implement an Evaluation that expects the objects passed in to be instances of type Car and checks their history size.

EvenHistoryEvaluation.java

```

01 /* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */
02 package com.db4odoc.evaluations;
03
04 import com.db4o.query.*;
05
06
07 public class EvenHistoryEvaluation implements Evaluation {
08     public void evaluate(Candidate candidate) {
09         Car car=(Car)candidate.getObject();
10         candidate.include(car.getHistory().size() % 2 == 0);
11     }
12 }
```

To test it, let's add two cars with history sizes of one and two respectively:

EvaluationExample.java: storeCars

```

01 private static void storeCars(ObjectContainer container) {
02     Pilot pilot1 = new Pilot("Michael Schumacher", 100);
03     Car car1 = new Car("Ferrari");
04     car1.setPilot(pilot1);
05     car1.snapshot();
06     container.set(car1);
```

```
07 |     Pilot pilot2 = new Pilot("Rubens Barrichello", 99);  
08 |     Car car2 = new Car("BMW");  
09 |     car2.setPilot(pilot2);  
10 |     car2.snapshot();  
11 |     car2.snapshot();  
12 |     container.set(car2);  
13 | }
```

and run our evaluation against them:

#### EvaluationExample.java: queryWithEvaluation

```
1 private static void queryWithEvaluation(ObjectContainer container) {  
2     Query query = container.query();  
3     query.constraint(Car.class);  
4     query.constraint(new EvenHistoryEvaluation());  
5     ObjectSet result = query.execute();  
6     listResult(result);  
7 }
```

This revision (9) was last Modified 2006-11-13T15:36:47 by Tetyana.

# Drawbacks

While evaluations offer you another degree of freedom for assembling queries, they come at a certain cost: As you may already have noticed from the example, evaluations work on the fully instantiated objects, while 'normal' queries peek into the database file directly. So there's a certain performance penalty for the object instantiation, which is wasted if the object is not included into the candidate set.

Another restriction is that, while 'normal' queries can bypass encapsulation and access candidates' private members directly, evaluations are bound to use their external API, just as in the language itself.

One last hint for Java users: Evaluations are expected to be serializable for client/server operation. So be careful when implementing them as (anonymous) inner classes and keep in mind that those will carry an implicit reference to their surrounding class and everything that belongs to it. Best practice is to always implement evaluations as normal top level or static inner classes.

This revision (4) was last Modified 2006-11-13T15:23:52 by Tetyana.

# Sorting Query Results

Often applications need to present query results in a sorted order. There are several ways to achieve this with db4o.

This Pilot class will be used in the following examples:

Pilot.java

```
01  /* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */  
02  
03 package com.db4odoc.sorting;  
04  
05 public class Pilot {  
06     private String name;  
07  
08     private int points;  
09  
10    public Pilot(String name) {  
11        this.name = name;  
12    }  
13  
14    public Pilot(String name, int points) {  
15        this.name = name;  
16        this.points = points;  
17    }  
18  
19    public String getName() {  
20        return name;  
21    }  
22  
23    public int getPoints() {  
24        return points;  
25    }
```

```

26 |
27 |     public String toString() {
28 |         if (points == 0) {
29 |             return name;
30 |         } else {
31 |             return name + "/" + points;
32 |         }
33 |     }
34 | }

```

The database will be filled with the following Pilot objects:

#### SortingExample.java: setObjects

```

01 |     public static void setObjects() {
02 |         new File(DB40_FILE_NAME).delete();
03 |         ObjectContainer container = Db4o.openFile(DB40_FILE_NAME);
04 |         try {
05 |             for (int i = 0; i < 10; i++) {
06 |                 for (int j = 0; j < 5; j++) {
07 |                     Pilot pilot = new Pilot("Pilot #" + i, j + 1);
08 |                     container.set(pilot);
09 |                 }
10 |             }
11 |         } finally {
12 |             container.close();
13 |         }
14 |     }

```

The following topics discuss some of the possible methods and point out their advantages and disadvantages.

More Reading:

- [SODA Sorting](#)

- [Native Query Sorting](#)
- [Evaluation Results Sorting](#)

This revision (11) was last Modified 2007-06-15T10:16:29 by Tetyana.

# SODA Sorting

SODA query API gives you a possibility to sort any field in ascending or descending order and combine sorting of different fields. For example, let's retrieve the objects of the Pilot class [saved before](#), sorting "points" field in descending order and "name" field in ascending.

SortingExample.java: getObjectsSODA

```

01  public static void getObjectsSODA() {
02  |
03  |     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
04  |     try {
05  |         Query query = container.query();
06  |         query.constraint(Pilot.class);
07  |         query.descend("name").orderAscending();
08  |         query.descend("points").orderDescending();
09  |         long t1 = System.currentTimeMillis();
10  |         ObjectSet result = query.execute();
11  |         long t2 = System.currentTimeMillis();
12  |         long diff = t2 - t1;
13  |         System.out.println("Time to query and sort with SODA: "
14  |                            + diff + " ms.");
15  |         listResult(result);
16  |     } finally {
17  |         container.close();
18  |     }
19 }
```

Obvious disadvantages of this method:

- limitations of SODA queries (not type-safe and not compile-time checked);
- limitations of sorting mechanism (only alphabetical for strings, numerical for numbers and object id for objects)

The valuable advantage of this method is its high performance.

This revision (8) was last Modified 2007-10-13T13:39:44 by Tetyana.

# Native Query Sorting

Native Query syntax allows you to specify a comparator, which will be used to sort the results:

Java:

```
<TargetType> ObjectSet<TargetType> query
(Predicate<TargetType> predicate,
QueryComparator<TargetType> comparator)
```

In order to get the same results as in [SODA Sorting](#) example we will write the following code:

SortingExample.java: getObjectsNQ

```
01  public static void getObjectsNQ() {
02 |     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
03 |     try {
04 |         long t1 = System.currentTimeMillis();
05 |         ObjectSet result = container.query(
06 |             new Predicate<Pilot>() {
07 |                 public boolean match(Pilot pilot) {
08 |                     return true;
09 |                 }
10 |             }, new QueryComparator<Pilot>() {
11 |                 public int compare(Pilot p1, Pilot p2) {
12 |                     int result = p1.getPoints()
13 |                         - p2.getPoints();
14 |                     if (result == 0) {
15 |                         return p1.getName().compareTo(
16 |                             p2.getName());
17 |                     } else {
18 |                         return -result;
19 |                     }
20 |                 }
21 |             });
22 |     }
```

```
22 |     long t2 = System.currentTimeMillis();
23 |     long diff = t2 - t1;
24 |     System.out
25 |         .println("Time to execute with NQ and comparator: "
26 |             + diff + " ms.");
27 |     listResult(result);
28 | } finally {
29 |     container.close();
30 | }
31 | }
```

Advantages of NQ sorting:

- type-safe queries and sorting, compile-time checking;
- ability to implement any custom sorting algorithm

The main disadvantage is decreased performance as at current stage [optimization](#) of sorted Native Queries is not possible.

This revision (7) was last Modified 2007-12-17T16:30:08 by Tetyana.

# Evaluation Results Sorting

In some cases you may find it necessary to use Evaluations. There is no standard sorting API for the Evaluation results. But you can sort the returned result set using standard Java collection API.

For example, let's retrieve the objects of the Pilot class [saved before](#), selecting only pilots with even points and sorting them according to their name:

SortingExample.java: getObjectsEval

```

01  public static void getObjectsEval() {
02  |
03  |     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
04  |     try {
05  |         long t1 = System.currentTimeMillis();
06  |         Query query = container.query();
07  |         query.constrain(Pilot.class);
08  |         query.constrain(new Evaluation() {
09  |             public void evaluate(Candidate candidate) {
10  |                 Pilot pilot = (Pilot) candidate.getObject();
11  |                 candidate.include(pilot.getPoints() % 2 == 0);
12  |             }
13  |         });
14  |         List<Pilot> result = new ArrayList<Pilot>(query.execute());
15  |         Collections.sort(result, new Comparator<Pilot>() {
16  |             public int compare(Pilot p1, Pilot p2) {
17  |                 return p1.getName().compareTo(p2.getName());
18  |             }
19  |         });
20  |         long t2 = System.currentTimeMillis();
21  |         long diff = t2 - t1;
22  |         System.out
23  |             .println("Time to execute with Evaluation query and collection sorting: "
24  |                     + diff + " ms.");
25  |         listResult(result);
26  |     } finally {
27  |         container.close();
28  |     }

```

This sorting method can be used to sort query results when the sorting can not be added to the query (Evaluations, QBE).

This revision (9) was last Modified 2007-06-15T15:47:02 by Tetyana.

# Custom Query Comparator

Db4o allows using a custom comparator for query processing. This can be useful for replacing the standard way of selecting query results, for example, when a non-standard attribute type should be compared as string or integer.

The usage is best shown on an example.

Let's create a custom class containing string information:

**MyString.java**

```

01 /* Copyright (C) 2007 db4objects Inc. http://www.db4o.com */
02 package com.db4odoc.comparing;
03
04 class MyString {
05 |
06 |   private String _string;
07 |
08 public MyString(String string) {
09 |   _string = string;
10 | }
11 |
12 public String toString() {
13 |   return _string;
14 | }
15 }

```

MyString class will be used for an attribute in the following class:

**Record.java**

```

01 /* Copyright (C) 2007 db4objects Inc. http://www.db4o.com */
02 package com.db4odoc.comparing;
03
04 public class Record {
05 |   private MyString _record;
06 |

```

```

07 |
08 |     public Record(String record) {
09 |         _record = new MyString(record);
10 |     }
11 |
12 |     public String toString() {
13 |         return _record.toString();
14 |     }
15 |

```

Let's save some Record objects to the database:

#### CompareExample.java: storeRecords

```

01 |     private static void storeRecords(Configuration configuration) {
02 |         new File(FILENAME).delete();
03 |         ObjectContainer container = Db4o.openFile(configuration, FILENAME);
04 |         try {
05 |             Record record = new Record("Michael Schumacher, points: 100");
06 |             container.set(record);
07 |             record = new Record("Rubens Barrichello, points: 98");
08 |             container.set(record);
09 |             record = new Record("Kimi Raikonen, points: 55");
10 |             container.set(record);
11 |         } finally {
12 |             container.close();
13 |         }
14 |

```

Selecting the query results, we need to compare string values contained in MyString objects. This can be configured with the following setting:

#### CompareExample.java: configure

```

01 |     private static Configuration configure() {
02 |         Configuration configuration = Db4o.newConfiguration();
03 |         configuration.objectClass(MyString.class).compare(new ObjectAttribute() {

```

```

04 |         public Object attribute(Object original) {
05 |             if (original instanceof MyString) {
06 |                 return ((MyString) original).toString();
07 |             }
08 |             return original;
09 |         }
10 |     });
11 |     return configuration;
12 | }

```

This piece of code registers an attribute provider for special query behavior.

The query processor will compare the object returned by the attribute provider instead of the actual object, both for the constraint and the candidate persistent object.

The querying code will look like this:

#### CompareExample.java: checkRecords

```

01 |     private static void checkRecords(Configuration configuration) {
02 |         ObjectContainer container = Db4o.openFile(configuration, FILENAME);
03 |         try {
04 |             Query q = container.query();
05 |             q.constrain(new Record("Rubens"));
06 |             q.descend("_record").constraints().contains();
07 |             ObjectSet result = q.execute();
08 |             listResult(result);
09 |         } finally {
10 |             container.close();
11 |         }
12 |     }

```

Using query comparator feature we can change the standard way of selecting query results. This can be helpful for:

- querying user types using simple string or numeric representation
- encapsulating additional logic into querying algorithm.

This revision (4) was last Modified 2007-05-07T09:44:16 by Tetyana.

# Transparent Persistence

One of db4o goals is to make database transparent to the application logic. Would not it be nice after an initial registering of an object with a database with a single store() method to leave the database to manage all the future object modification? This idea went a long way from initial vague thoughts to a final implementation in db4o version 7.1 and was named Transparent Persistence. Let's look at the implementation in a more detail.

More Reading:

- [Transparent Persistence Implementation](#)
- [TP Enhanced Example](#)
- [Detailed Example](#)
- [Collection Example](#)

This revision (1) was last Modified 2008-01-15T05:56:27 by Tetyana.

# Transparent Persistence Implementation

The basic logic of Transparent Persistence (TP) is the following:

- classes available for Transparent Persistence should implement Activatable interface, which allows to bind an object in the reference cache to the current object container.
- persistent object should be initially explicitly stored to the database:

Java:

```
objectContainer.store(myObject)
```

myObject can be an object of any complexity including a linked list or a collection (currently you must use db4o-specific implementation for transparent collections: `ArrayList4`). For complex objects all field objects will be registered with the database with this call as well.

- Stored object are bound to the Transparent Persistent framework when they are instantiated in the reference cache. This happens after the initial `store()` or when an object is retrieved from the database through one of the querying mechanisms.
- Whenever a `commit()` call is issued by the user, TP framework scans for modified persistent objects and implicitly calls `store()` on them before committing the transaction. Implicit commit with the mentioned above changes also occurs when the database is closed.

Note that Transparent Persistence is based on Transparent Activation, so it is strongly recommended to study [Transparent Activation](#) documentation first.

In order to make use of Transparent Persistence you will need:

1. Enable Transparent Activation (required for binding object instances to the TP framework) on the database level:

Java:

```
configuration.add(new TransparentPersistenceSupport());
```

2. Implement Activatable/IActivatable interface for the persistent classes, either manually or through using [enhancement tools](#).
3. Call activate method at the beginning of all class methods that modify class fields:

Java:

```
activate(ActivationPurpose.WRITE)
```

Note that `TransparentPersistenceSupport` configuration implicitly adds `TransparentActivationSupport`. The fact is, that before modification each field object should be loaded into the reference cache and that is the job of TA. So TA should be utilised in any case before TP. You can also note that the way TA and TP links

into objects is absolutely identical: TP also uses the same activate call, but in this case its purpose is WRITE.

This revision (5) was last Modified 2008-01-20T08:42:21 by Tetyana.

# TP Enhanced Example

As it was mentioned [before](#) TP uses the same technology as TA. Due to that enhancement strategies and tools are the same for Transparent Persistence. Actually TP enhancement automatically includes TA enhancement, as TP relies on object activation before modification.

On Java side enhancement can be done at build-time by using a customized ant build script. For more information see [TP Enhancement On Java](#).

On .NET side TP enhancement can be done by using command-line Db4oTool or by customizing MSBuild script:

- [Db4oTool enhancement](#)
- [MSBuild enhancement](#)

This revision (1) was last Modified 2008-01-20T13:26:31 by Tetyana.

# TP Enhancement On Java

This topic applies to Java version only

TP Enhancement on Java platform can be done by customizing the ant build script to include instrumentation for persistent classes.

For a simple example we will use [SensorPanel](#) class, which represents a simple linked list. In our example application we will first store several objects of this class, then retrieve and modify them. Transparent Persistence mechanism should take care of modified objects and persist them to the database when the transaction is committed or the database is closed. As SensorPanel does not implement Activatable interface, we will need to use db4o enhancement tools to implement this interface after the class is built.

Let's look at our example code.

First, we need to configure Transparent Persistence:

TPExample.java: configureTP

```
1 | private static Configuration configureTP() {
2 |     Configuration configuration = Db4o.newConfiguration();
3 |     // add TP support
4 |     configuration.add(new TransparentPersistenceSupport());
5 |     return configuration;
6 | }
```

Now we will store a linked list of 10 elements:

TPExample.java: storeSensorPanel

```
01 | private static void storeSensorPanel() {
02 |     new File(DB4O_FILE_NAME).delete();
03 |     ObjectContainer container = database(Db4o.newConfiguration());
04 |     if (container != null) {
05 |         try {
06 |             // create a linked list with length 10
07 |             SensorPanel list = new SensorPanel().createList(10);
08 |             container.store(list);
09 |         } finally {
10 |             closeDatabase();
11 |         }
12 |     }
13 | }
```

13 }

And the following procedure will test the effect of TP:

TPExample.java: testTransparentPersistence

```

01  private static void testTransparentPersistence() {
02 |     storeSensorPanel();
03 |     Configuration configuration = configureTP();
04 |
05 |     ObjectContainer container = database(configuration);
06 |     if (container != null) {
07 |         try {
08 |             ObjectSet result = container.queryByExample(new SensorPanel(1));
09 |             listResult(result);
10 |             SensorPanel sensor = null;
11 |             if (result.size() > 0) {
12 |                 System.out.println("Before modification: ");
13 |                 sensor = (SensorPanel) result.get(0);
14 |                 // the object is a linked list, so each call to next()
15 |                 // will need to activate a new object
16 |                 SensorPanel next = sensor.getNext();
17 |                 while (next != null) {
18 |                     System.out.println(next);
19 |                     // modify the next sensor
20 |                     next.setSensor(new Integer(10 + (Integer)next.getSensor()));
21 |                     next = next.getNext();
22 |                 }
23 |                 // Explicit commit stores and commits the changes at any time
24 |                 container.commit();
25 |             }
26 |         } finally {
27 |             // If there are unsaved changes to activatable objects, they
28 |             // will be implicitly saved and committed when the database
29 |             // is closed
30 |             closeDatabase();
31 |         }
32 |     }
33 |     // reopen the database and check the modifications
34 |     container = database(configuration);

```

```

35 |     if (container != null) {
36 |         try {
37 |             ObjectSet result = container.queryByExample(new SensorPanel(1));
38 |             listResult(result);
39 |             SensorPanel sensor = null;
40 |             if (result.size() > 0) {
41 |                 System.out.println("After modification: ");
42 |                 sensor = (SensorPanel) result.get(0);
43 |                 SensorPanel next = sensor.getNext();
44 |                 while (next != null) {
45 |                     System.out.println(next);
46 |                     next = next.getNext();
47 |                 }
48 |             }
49 |         } finally {
50 |             closeDatabase();
51 |         }
52 |     }
53 |

```

Of course, if you will run the code above as is, you will see that all the changes were lost. In order to fix it we will need to build the application with a special build script:

## Build.Xml

```

01 <?xml version="1.0"?>
02
03 <!--
04 TP build time enhancement sample.
05 -->
06
07 <project name="tpexamples" default="buildall">
08
09 <!--
10 Set up the required class path for the enhancement task.
11 -->
12 <path id="db4o.enhance.path">
13   <pathel ement path="${basedir}" />
14   <fileset dir="lib">
15     <include name="**/*.jar" />
16   </fileset>

```

```

17  </path>
18
19  <!-- Define enhancement task. -->
20  <taskdef name="db4o-enhance" classname="com.db4o.instrumentation.ant.Db4oFileEnhancerAntTask" classpathref="db4o增强路径" loaderref="db4o增强加载器" />
21
22  <typedef name="transparent-persistence" classname="com.db4o.ta.instrumentation.ant.TAAntClassEditFactory" classpathref="db4o增强路径" loaderref="db4o增强加载器" />
23
24
25
26  <target name="buildall" depends="compile">
27
28      <!-- Create enhanced output directory-->
29      <mkdir dir="${basedir}/enhanced-bin" />
30      <delete dir="${basedir}/enhanced-bin" quiet="true">
31          <include name="**/*" />
32      </delete>
33
34      <db4o-enhance classstargetdir="${basedir}/enhanced-bin">
35
36          <classpath refid="db4o增强路径" />
37          <!-- Use compiled classes as an input -->
38          <sources dir="${basedir}/bin" />
39
40          <!-- Call transparent persistence enhancement -->
41          <transparent-persistence />
42
43      </db4o-enhance>
44
45  </target>
46
47  <!-- Simple compilation. Note that db4o version
48  should be adjusted to correspond to the version
49  you are using
50  -->
51  <target name="compile">
52      <javac fork="true" destdir="${basedir}/bin">
53          <classpath>
54              <path element location="${basedir}/lib/db4o-7.1.26.8872-java5.jar" />

```

```

55      </cl asspath>
56      <src path="${basedir}/src" />
57      <include name="**/*.java" />
58      </javac>
59  </target>
60
61 </project>

```

The build script relies on several jars:

- ant.jar - [Ant](#) library
- bloat-1.0.jar - bloat bytecode instrumentation library
- db4o-x.x-instrumentation.jar - db4o instrumentation library on top of bloat
- db4o-x.x-java5.jar - db4o jar
- db4o-x.x-taj.jar - db4o transparent activation support
- db4o-x.x-tools.jar - db4o tools

All these jars should be added to /lib folder in the project directory.

After running the build script above you will get /bin and /enhanced-bin folders produced in your project folder. /bin folder contains compiled application classes, whereas /enhanced-bin contains compiled and enhanced classes. For testing the result of the enhancement you can use the following batch file (to be run from /enhanced-bin folder):

```

set CLASSPATH=. ; ${PROJECT_ROOT}\lib\db4o-x.x-java5.jar

java com.db4odoc.tpbuildtime.TPExample

```

This revision (3) was last Modified 2008-01-20T14:31:55 by Tetyana.

# SensorPanel

This topic applies to Java only

SensorPanel.java

```
01 /* Copyright (C) 2004 - 2007 db4objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.tpbuilder;
04
05 public class SensorPanel {
06 |
07     private Object _sensor;
08 |
09     private SensorPanel _next;
10 |
11     public SensorPanel() {
12         // default constructor for instantiation
13     }
14     // end SensorPanel
15 |
16     public SensorPanel(int value) {
17         _sensor = new Integer(value);
18     }
19     // end SensorPanel
20 |
21     public SensorPanel getNext() {
22         return _next;
23     }
24     // end getNext
25 |
26     public Object getSensor() {
27         return _sensor;
```

```
28 }  
29 // end getSensor  
30  
31 public void setSensor(Object sensor) {  
32     _sensor = sensor;  
33 }  
34 // end setSensor  
35  
36 public SensorPanel createList(int length) {  
37     return createList(length, 1);  
38 }  
39 // end createList  
40  
41 public SensorPanel createList(int length, int first) {  
42     int val = first;  
43     SensorPanel root = newElement(first);  
44     SensorPanel list = root;  
45     while (--length > 0) {  
46         list._next = newElement(++val);  
47         list = list._next;  
48     }  
49     return root;  
50 }  
51 // end createList  
52  
53 protected SensorPanel newElement(int value) {  
54     return new SensorPanel(value);  
55 }  
56 // end newElement  
57  
58 public String toString() {  
59     return "Sensor #" + getSensor();  
60 }
```

```
61 | // end toString  
62 }
```

This revision (1) was last Modified 2008-01-20T14:11:05 by Tetyana.

# Detailed Example

Let's look at a primitive example, demonstrating manual implementation of Activatable/IActivatable interface for TP. We will use a class similar to the [one](#) used in Transparent Activation chapters.

SensorPanel.java

```
01 /* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.tpxexample;
04
05 import com.db4o.activation.*;
06 import com.db4o.ta.*;
07
08 public class SensorPanel implements Activatable {
09 |
10 |    private Object _sensor;
11 |
12 |    private SensorPanel _next;
13 |
14 /*activator registered for this class*/
15 |    transient Activator _activator;
16 |
17 public SensorPanel () {
18 |    // default constructor for instantiation
19 |}
20 | // end SensorPanel TA
21 |
22 public SensorPanel (int value) {
23 |    _sensor = new Integer(value);
24 |}
25 | // end SensorPanel TA
26 |
27 /*Bind the class to the specified object container, create the activator*/
28 public void bind(Activator activator) {
```

```
29     if (_activator == activator) {  
30         return;  
31     }  
32     if (activator != null && _activator != null) {  
33         throw new IllegalStateException();  
34     }  
35     _activator = activator;  
36 }  
37 // end bind  
38  
39     /*Call the registered activator to activate the next level,  
40      * the activator remembers the objects that were already  
41      * activated and won't activate them twice.  
42 */  
43 public void activate(ActivationPurpose purpose) {  
44     if (_activator == null)  
45         return;  
46     _activator.activate(purpose);  
47 }  
48 // end activate  
49  
50 public SensorPanel getNext() {  
51     /*activate direct members*/  
52     activate(ActivationPurpose.READ);  
53     return _next;  
54 }  
55 // end getNext  
56  
57 public Object getSensor() {  
58     /*activate direct members*/  
59     activate(ActivationPurpose.READ);  
60     return _sensor;  
61 }  
62 // end getSensor  
63
```

```
64  public void setSensor(Object sensor) {  
65      /*activate for persistence*/  
66      activate(AccivationPurpose.WRITE);  
67      _sensor = sensor;  
68  }  
69  // end setSensor  
70  
71  public SensorPanel createList(int length) {  
72      return createList(length, 1);  
73  }  
74  // end createList  
75  
76  public SensorPanel createList(int length, int first) {  
77      int val = first;  
78      SensorPanel root = newElement(first);  
79      SensorPanel list = root;  
80      while (--length > 0) {  
81          list._next = newElement(++val);  
82          list = list._next;  
83      }  
84      return root;  
85  }  
86  // end createList  
87  
88  protected SensorPanel newElement(int value) {  
89      return new SensorPanel(value);  
90  }  
91  // end newElement  
92  
93  public String toString() {  
94      return "Sensor #" + getSensor();  
95  }  
96  // end toString  
97 }
```

Note, that the only place where we can modify SensorPanel members is setSensor method/Sensor property, and that is where activate method is added.

Now we will only need to add Transparent Activation support:

## TPExample.java: configureTA

```
1 [+] private static Configuration configureTA() {
2 |     Configuration configuration = Db4o.newConfiguration();
3 |     // add TA support
4 |     configuration.add(new TransparentActivationSupport());
5 |     return configuration;
6 L }
```

Initial storing of the objects is done as usual with a single store call:

## TPExample.java: storeSensorPanel

```
01 [+] private static void storeSensorPanel() {
02 |     new File(DB4O_FILE_NAME).delete();
03 |     ObjectContainer container = database(Db4o.newConfiguration());
04 [+]     if (container != null) {
05 [+]         try {
06 |             // create a linked list with length 10
07 |             SensorPanel list = new SensorPanel().createList(10);
08 |             container.store(list);
09 [+]         } finally {
10 |             closeDatabase();
11 |         }
12 |     }
13 L }
```

Now we can test how Transparent Persistence helped us to keep the code simple. Let's select all elements from the linked SensorPanel list, modify them and store. As you remember default update depth is one, so without TP, we would have to store each member of the linked list (SensorPanel) separately. With TP enabled there is absolutely nothing to do: commit call will find all activatable objects and store those that were modified.

## TPExample.java: testTransparentPersistence

```

01  private static void testTransparentPersistence() {
02 |     storeSensorPanel();
03 |     Configuration configuration = configureTA();
04 |
05 |     ObjectContainer container = database(configuration);
06 |     if (container != null) {
07 |         try {
08 |             ObjectSet result = container.queryByExample(new SensorPanel(1));
09 |             listResult(result);
10 |             SensorPanel sensor = null;
11 |             if (result.size() > 0) {
12 |                 System.out.println("Before modification: ");
13 |                 sensor = (SensorPanel) result.get(0);
14 |                 // the object is a linked list, so each call to next()
15 |                 // will need to activate a new object
16 |                 SensorPanel next = sensor.getNext();
17 |                 while (next != null) {
18 |                     System.out.println(next);
19 |                     // modify the next sensor
20 |                     next.setSensor(new Integer(10 + (Integer)next.getSensor()));
21 |                     next = next.getNext();
22 |                 }
23 |                 // Explicit commit stores and commits the changes at any time
24 |                 container.commit();
25 |             }
26 |         } finally {
27 |             // If there are unsaved changes to activatable objects, they
28 |             // will be implicitly saved and committed when the database
29 |             // is closed
30 |             closeDatabase();
31 |         }
32 |     }
33 |     // reopen the database and check the modifications
34 |     container = database(configuration);
35 |     if (container != null) {

```

```
36 |     try {
37 |         ObjectSet result = container.queryByExample(new SensorPanel(1));
38 |         listResult(result);
39 |         SensorPanel sensor = null;
40 |         if (result.size() > 0) {
41 |             System.out.println("After modification: ");
42 |             sensor = (SensorPanel) result.get(0);
43 |             SensorPanel next = sensor.getNext();
44 |             while (next != null) {
45 |                 System.out.println(next);
46 |                 next = next.getNext();
47 |             }
48 |         }
49 |     } finally {
50 |         closeDatabase();
51 |     }
52 | }
53 | }
```

That's all. The benefits that we've got:

- clean and friendly to refactorings code;
- performance benefit: only modified objects are stored;
- hussle-free development.

This revision (2) was last Modified 2008-01-15T06:07:43 by Tetyana.

# Collection Example

In the [previous example](#) we reviewed how Transparent Persistence should be used with simple types. Let's now look how it is done with the collections.

In order to make collections TP Activatable you will need to use db4o-specific ArrayList4 collection. This collection implements .NET/Java collection interfaces, therefore it can be easily integrated with your code. The following class contains a collection of Pilot objects:

Team.java

```
01 /* Copyright (C) 2007 db4o objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.tplexample;
04
05 import java.util.*;
06
07 import com.db4o.activation.*;
08 import com.db4o.collections.*;
09 import com.db4o.ta.*;
10
11
12 public class Team implements Activatable {
13 |
14 |   private List<Pilot> _pilots = new ArrayList4<Pilot>();
15 |
16 |   String _name;
17 |
18 |   //TA Activator
19 |   transient Activator _activator;
20 |
21 |   // Bind the class to an object container
22 public void bind(Activator activator) {
23 |     if (_activator == activator) {
```

```
24 |         return;
25 |     }
26 |     if (activator != null && _activator != null) {
27 |         throw new IllegalStateException();
28 |     }
29 |     _activator = activator;
30 | }
31 |
32 | // activate object fields
33 | public void activate(ActivationPurpose purpose) {
34 |     if (_activator == null) return;
35 |     _activator.activate(purpose);
36 | }
37 |
38 | public void addPilot(Pilot pilot) {
39 |     // activate before adding new pilots
40 |     activate(ActivationPurpose.WRITE);
41 |     _pilots.add(pilot);
42 | }
43 |
44 | public void listAllPilots() {
45 |     // activate before printing the collection members
46 |     activate(ActivationPurpose.READ);
47 |
48 |     for (Iterator<Pilot> iter = _pilots.iterator(); iter.hasNext();) {
49 |         Pilot pilot = (Pilot) iter.next();
50 |         System.out.println(pilot);
51 |     }
52 | }
53 |
54 | List<Pilot> getPilots() {
55 |     activate(ActivationPurpose.READ);
56 |     return _pilots;
```

57 | }

58 | }

You can see that except for using ArrayList4, the implementation follows the same rules as in the previous [simple example](#).

Its usage has no surprises as well:

## TPCollectionExample.java: storeCollection

```

01 | private static void storeCollection() {
02 |     new File(DB40_FILE_NAME). delete();
03 |     ObjectContainer container = database(configureTP());
04 |     if (container != null) {
05 |         try {
06 |             Team team = new Team();
07 |             for (int i = 0; i < 10; i++) {
08 |                 team.addPilot(new Pilot("Pilot #" + i));
09 |             }
10 |             container.store(team);
11 |         } catch (Exception ex) {
12 |             ex.printStackTrace();
13 |         } finally {
14 |             closeDatabase();
15 |         }
16 |     }
17 | }
```

## TPCollectionExample.java: testCollectionPersistence

```

01 | private static void testCollectionPersistence() {
02 |     storeCollection();
03 |     ObjectContainer container = database(configureTP());
04 |     if (container != null) {
```

```
05 |     try {
06 |         Team team = (Team) container.queryByExample(new Team()).next();
07 |         // this method will activate all the members in the collection
08 |         Iterator<Pilot> it = team.getPilots().iterator();
09 |         while (it.hasNext()) {
10 |             Pilot p = it.next();
11 |             p.setName("Modified: " + p.getName());
12 |         }
13 |         team.addPilot(new Pilot("New pilot"));
14 |         // explicitly commit to persist changes
15 |         container.commit();
16 |     } catch (Exception ex) {
17 |         ex.printStackTrace();
18 |     } finally {
19 |         // If TP changes were not committed explicitly,
20 |         // they would be persisted with the #close call
21 |         closeDatabase();
22 |     }
23 | }
24 | // reopen the database and check the changes
25 | container = database(configureTP());
26 | if (container != null) {
27 |     try {
28 |         Team team = (Team) container.queryByExample(new Team()).next();
29 |         team.listAllPilots();
30 |     } catch (Exception ex) {
31 |         ex.printStackTrace();
32 |     } finally {
33 |         closeDatabase();
34 |     }
35 | }
36 }
```

## Collection Example

So the only thing you should remember when using TP with collections is to use ArrayList4 instead of platform-specific collection.

This revision (3) was last Modified 2008-01-26T16:31:16 by Tetyana.

# Working With Structured Objects

In real world objects are referenced by each other creating deep reference structures.

This chapter will give you an overview of how db4o deals with structured objects.

For an example we will use a simple model, where Pilot class is referenced from Car class.

Pilot.java

```
01 /* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */
02 package com.db4odoc.structured;
03
04 public class Pilot {
05     private String name;
06
07     private int points;
08
09     public Pilot(String name, int points) {
10         this.name = name;
11         this.points = points;
12     }
13
14     public int getPoints() {
15         return points;
16     }
17
18     public void addPoints(int points) {
19         this.points += points;
20     }
21
22     public String getName() {
23         return name;
24     }
```

```
25 |
26  public String toString() {
27      return name + "/" + points;
28  }
29 }
```

## Car.java

```
01 /* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */
02 package com.db4odoc.structured;
03
04 public class Car {
05     private String model;
06
07     private Pilot pilot;
08
09     public Car(String model) {
10         this.model = model;
11         this.pilot = null;
12     }
13
14     public Pilot getPilot() {
15         return pilot;
16     }
17
18     public void setPilot(Pilot pilot) {
19         this.pilot = pilot;
20     }
21
22     public String getModel() {
23         return model;
24     }
25 }
```

```
26 public String toString() {  
27     return model + "[" + pilot + "]";  
28 }  
29 }
```

More Reading:

- [Retrieving Structured Objects](#)
- [Updating Structured Objects](#)
- [Deleting Structured Objects](#)

This revision (6) was last Modified 2007-05-01T06:36:07 by Tetyana.

# Retrieving Structured Objects

## Contents

- QBE
- Native Queries
- SODA Query API

## QBE

---

To retrieve all cars a simple 'blank' prototype can be used.

### StructuredExample.java: retrieveAllCarsQBE

```

1 private static void retrieveAllCarsQBE(ObjectContainer container) {
2     Car proto = new Car(null);
3     ObjectSet result = container.get(proto);
4     listResult(result);
5 }
```

Now let's initialize the prototype to specify all cars driven by Rubens Barrichello.

### StructuredExample.java: retrieveCarByPilotQBE

```

1 private static void retrieveCarByPilotQBE(ObjectContainer container) {
2     Pilot pilotproto = new Pilot("Rubens Barrichello", 0);
3     Car carproto = new Car(null);
4     carproto.setPilot(pilotproto);
5     ObjectSet result = container.get(carproto);
6     listResult(result);
7 }
```

What about retrieving a pilot by car? You simply don't need that -if you already know the car, you can simply access the pilot field directly.

## StructuredExample.java: retrieveCarByPilotQBE

```

1  private static void retrieveCarByPilotQBE(ObjectContainer container) {
2      Pilot pilotproto = new Pilot("Rubens Barrichello", 0);
3      Car carproto = new Car(null);
4      carproto.setPilot(pilotproto);
5      ObjectSet result = container.get(carproto);
6      listResult(result);
7  }

```

## Native Queries

Using native queries with constraints on deep structured objects is straightforward, you can do it just like you would in plain other code.

Let's constrain our query to only those cars driven by a Pilot with a specific name:

## StructuredExample.java: retrieveCarsByPilotNameNative

```

01 private static void retrieveCarsByPilotNameNative(
02     ObjectContainer container) {
03     final String pilotName = "Rubens Barrichello";
04     ObjectSet results = container.query(new Predicate<Car>() {
05         public boolean match(Car car) {
06             return car.getPilot().getName().equals(pilotName);
07         }
08     });
09     listResult(results);
10 }

```

## SODA Query API

In order to use SODA for querying for a car given its pilot's name you have to descend two levels into our query.

## StructuredExample.java: retrieveCarByPilotNameQuery

```

1 private static void retrieveCarByPilotNameQuery(
2     ObjectContainer container) {
3     Query query = container.query();
4     query.constrain(Car.class);
5     query.descend("pilot").descend("name").constrain(
6         "Rubens Barrichello");
7     ObjectSet result = query.execute();
8     listResult(result);
9 }

```

You can also constrain the pilot field with a prototype to achieve the same result.

#### StructuredExample.java: retrieveCarByPilotProtoQuery

```

1 private static void retrieveCarByPilotProtoQuery(
2     ObjectContainer container) {
3     Query query = container.query();
4     query.constrain(Car.class);
5     Pilot proto = new Pilot("Rubens Barrichello", 0);
6     query.descend("pilot").constrain(proto);
7     ObjectSet result = query.execute();
8     listResult(result);
9 }

```

Descending into a query provides you with another query. Starting out from a query root you can descend in multiple directions. In practice this is the same as ascending from one child to a parent and descending to another child. The queries turn one-directional references in objects into true relations. Here is an example that queries for "a Pilot that is being referenced by a Car, where the Car model is 'Ferrari':

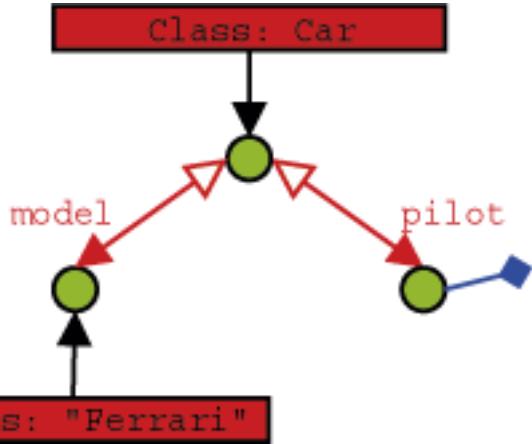
#### StructuredExample.java: retrievePilotByCarModelQuery

```

1 private static void retrievePilotByCarModelQuery(
2     ObjectContainer container) {
3     Query carquery = container.query();

```

```
4 |     carquery.constraint(Car.class);  
5 |     carquery.descend("model").constraint("Ferrari");  
6 |     Query pilotquery = carquery.descend("pilot");  
7 |     ObjectSet result = pilotquery.execute();  
8 |     listResult(result);  
9 | }
```



This revision (8) was last Modified 2006-12-29T15:53:09 by Tetyana.

# Updating Structured Objects

To update structured objects in db4o, you simply call set() on them again.

StructuredExample.java: updateCar

```

01  private static void updateCar(ObjectContainer container) {
02    ObjectSet result = container.query(new Predicate<Car>() {
03      public boolean match(Car car) {
04        return car.getModel().equals("Ferrari");
05      }
06    });
07    Car found = (Car) result.next();
08    found.setPilot(new Pilot("Somebody else", 0));
09    container.set(found);
10   result = container.query(new Predicate<Car>() {
11     public boolean match(Car car) {
12       return car.getModel().equals("Ferrari");
13     }
14   });
15   listResult(result);
16 }
```

Let's modify the pilot, too.

StructuredExample.java: updatePilotSingleSession

```

01 private static void updatePilotSingleSession(
02   ObjectContainer container) {
03   ObjectSet result = container.query(new Predicate<Car>() {
04     public boolean match(Car car) {
05       return car.getModel().equals("Ferrari");
06     }
07   });
08   Car found = (Car) result.next();
09   found.getPilot().addPoints(1);
10   container.set(found);
11   result = container.query(new Predicate<Car>() {
12     public boolean match(Car car) {
13       return car.getModel().equals("Ferrari");
14     }
15   });
16 }
```

```

14 |     }
15 |   });
16 |   listResult(result);
17 L }
```

Nice and easy, isn't it? But there is something that is not obvious in this example. Let's see what happens if we split this task in two separate db4o sessions: In the first we modify our pilot and update his car:

StructuredExample.java: updatePilotSeparateSessionsPart1

```

01 private static void updatePilotSeparateSessionsPart1(
02     ObjectContainer container) {
03     ObjectSet result = container.query(new Predicate<Car>() {
04         public boolean match(Car car) {
05             return car.getModel().equals("Ferrari");
06         }
07     });
08     Car found = (Car) result.next();
09     found.getPilot().addPoints(1);
10     container.set(found);
11 }
```

And in the second, we'll double-check our modification:

StructuredExample.java: updatePilotSeparateSessionsPart2

```

1 private static void updatePilotSeparateSessionsPart2(
2     ObjectContainer container) {
3     ObjectSet result = container.query(new Predicate<Car>() {
4         public boolean match(Car car) {
5             return car.getModel().equals("Ferrari");
6         }
7     });
8     listResult(result);
9 }
```

[/filter]

StructuredExample.java: updatePilotSeparateSessionsImprovedPart2

```
01 private static void updatePilotSeparateSessionsImprovedPart2(
```

```

02  ObjectContainer container) {
03      ObjectSet result = container.query(new Predicate<Car>() {
04          public boolean match(Car car) {
05              return car.getModel().equals("Ferrari");
06          }
07      });
08      Car found = (Car) result.next();
09      found.getPilot().addPoints(1);
10      container.set(found);
11  }

```

[/filter]

[filter=vb]

StructuredExample.vb: UpdatePilotSeparateSessionsImprovedPart3

```

1 Private Shared Sub UpdatePilotSeparateSessionsImprovedPart3(ByVal db As IObjectContainer)
2     Dim result As ObjectSet = db.Get(New Car("Ferrari"))
3     ListResult(result)
4 End Sub

```

You can also achieve expected results using:

1. ExtObjectContainer#set(object, depth) to update exact amount of referenced fields
2. Use configuration.objectClass(clazz).updateDepth(depth) setting to define sufficient update depth for a specific object
3. Use global setting for all the persisted objects:

Java:

```
configuration.updateDepth(depth);
```

However global updateDepth is not flexible enough for real-world objects having different depth of reference structures.

**ATTENTION:** Setting global update depth to the maximum value will result in serious performance penalty. Please, use this setting ONLY for debug purposes.

Note that container configuration must be set before the container is opened and/or passed to the openFile/openClient/openServer method.

This revision (9) was last Modified 2007-04-23T06:59:02 by Tetyana.

# Deleting Structured Objects

## Contents

- Recursive Deletion
- Recursive Deletion Revisited

As we have already seen, we call `delete()` on objects to get rid of them.

### StructuredExample.java: deleteFlat

```

01  private static void deleteFlat(ObjectContainer container) {
02  ObjectSet result = container.query(new Predicate<Car>() {
03  public boolean match(Car car) {
04      return car.getModel().equals("Ferrari");
05  }
06 });
07 Car found = (Car) result.next();
08 container.delete(found);
09 result = container.get(new Car(null));
10 listResult(result);
11 }
```

Fine, the car is gone. What about the pilots?

### StructuredExample.java: retrieveAllPilotsQBE

```

1  private static void retrieveAllPilotsQBE(ObjectContainer container) {
2  Pilot proto = new Pilot(null, 0);
3  ObjectSet result = container.get(proto);
4  listResult(result);
5 }
```

Ok, this is no real surprise - we don't expect a pilot to vanish when his car is disposed of in real life, too. But what if we want an object's children to be thrown away on deletion, too?

# Recursive Deletion

The problem of recursive deletion (and its solution, too) is quite similar to the [update](#) problem. Let's configure db4o to delete a car's pilot, too, when the car is deleted.

StructuredExample.java: deleteDeepPart1

```

1 private static Configuration deleteDeepPart1() {
2     Configuration configuration = Db4o.newConfiguration();
3     configuration.objectClass("com.db4o.f1.chapter2.Car")
4         .cascadeOnDelete(true);
5     return configuration;
6 }

```

StructuredExample.java: deleteDeepPart2

```

01 private static void deleteDeepPart2(ObjectContainer container) {
02     ObjectSet result = container.query(new Predicate<Car>() {
03         public boolean match(Car car) {
04             return car.getModel().equals("BMW");
05         }
06     });
07     Car found = (Car) result.next();
08     container.delete(found);
09     result = container.query(new Predicate<Car>() {
10         public boolean match(Car car) {
11             return true;
12         }
13     });
14     listResult(result);
15 }

```

Again: Note that all configuration must take place before the ObjectContainer is opened.

Another way to organize cascaded deletion is using [callbacks](#). The callbacks allow you to specify explicitly, which objects are to be deleted.

## Recursive Deletion Revisited

But wait - what happens if the children of a removed object are still referenced by other objects?

StructuredExample.java: deleteDeepRevisited

```

01  private static void deleteDeepRevisited(ObjectContainer container) {
02      ObjectSet result = container.query(new Predicate<Pilot>() {
03          public boolean match(Pilot pilot) {
04              return pilot.getName().equals("Michael Schumacher");
05          }
06      });
07      Pilot pilot = (Pilot) result.next();
08      Car car1 = new Car("Ferrari");
09      Car car2 = new Car("BMW");
10      car1.setPilot(pilot);
11      car2.setPilot(pilot);
12      container.set(car1);
13      container.set(car2);
14      container.delete(car2);
15      result = container.query(new Predicate<Car>() {
16          public boolean match(Car car) {
17              return true;
18          }
19      });
20      listResult(result);
21  }
```

StructuredExample.java: retrieveAllPilots

```

1  private static void retrieveAllPilots(ObjectContainer container) {
```

```
2 |     ObjectSet result = container.get(Pilot.class);  
3 |     listResult(result);  
4 | }
```

Currently db4o does **not** check whether objects to be deleted are referenced anywhere else, so please be very careful when using this feature. However it is fairly easy to implement referential checking on deletion using `deleting()` callback. See [Callbacks chapter](#) for more information.

This revision (10) was last Modified 2007-05-07T08:05:57 by Tetyana.

# Activation

Activation is a db4o mechanism, which controls object's fields instantiation. Why is it necessary? Let's look at an example of a database, that has a Tree structure, i.e. there is one Root object, which has N nodes, each node in its turn has K subnodes etc. Let the whole structure has M levels. What happens when you run a query, retrieving the root object? All the sub-objects will have to be created in the memory. If N,K,M are large numbers, you will most probably end up with the OutOfMemory exception.

Luckily db4o does not behave like this - when query retrieves objects, their fields are loaded into memory (activated in db4o terms) only to a certain depth - activation depth. In this case depth means "number of member references away from the original object". All the fields at lower levels (below activation depth) are set to null (for classes) or to default values (for primitive types).

Activation occurs in the following cases:

1. `ObjectSet#next()` is called on an `ObjectSet` retrieved in a query;
2. Object is activated explicitly with `ObjectContainer#activate(object, depth)`;
3. built-in collection element is accessed;
4. for the environment collections (.NET, Java) their members are activated automatically, when the collection is activated, using at least depth 1 for lists and depth 2 for maps.

More Reading:

- [Global Activation Settings](#)
- [Object-Specific Activation](#)
- [Transparent Activation Framework](#)

This revision (8) was last Modified 2006-11-14T18:20:20 by Eric Falsken.

# Global Activation Settings

Java:

```
configuration.activationDepth(activationDepth)
```

configures global activation depth, which will be used for all objects instead of the default value. This method should be called before opening a database file.

Java:

```
configuration.activationDepth(activationDepth)
```

has a similar effect, but the setting will be applied to the specific ObjectContainer and can be changed for the open database file.

ActivationExample.java: testActivationConfig

```
01  private static void testActivationConfig() {
02 |     storeSensorPanel();
03 |     Configuration configuration = Db4o.newConfiguration();
04 |     configuration.activationDepth(1);
05 |     ObjectContainer container = Db4o.openFile(configuration, DB4O_FILE_NAME);
06  ↘  try {
07 |     System.out.println("Object container activation depth = 1");
08 |     ObjectSet result = container.get(new SensorPanel(1));
09 |     listResult(result);
10  ↗  if (result.size() >0) {
11 |         SensorPanel sensor = (SensorPanel) result.get(0);
12 |         SensorPanel next = sensor.next;
13  ↗  while (next != null){
14 |             System.out.println(next);
15 |             next = next.next;
16 |         }
17 |     }
18  ↗  } finally {
19 |     container.close();
```

```
20 |      }
21 L  }
```

By configuring db4o you can have full control over activation behavior. The two extremes:

- using an activationDepth of Integer.MAX\_VALUE/Int32.MaxValue lets you forget about manual activation, but does not give you the best performance and memory footprint;
- using an activationDepth of 0 and activating and deactivating all objects manually keeps memory consumption extremely low, but needs more coding and attention.

This revision (11) was last Modified 2007-10-30T04:57:04 by Tetyana.

# Object-Specific Activation

You can tune up activation settings for specific classes with the following methods:

Java:

```
configuration.objectClass("yourClass").minimumActivationDepth(minimumDepth)
configuration.objectClass("yourClass").maximumActivationDepth(maximumDepth)
```

Cascading the activation depth to member fields, the depth value is reduced by one for the field. If the depth exceeds the maximumDepth specified for the class of the object, it is reduced to the maximumDepth. If the depth value is lower than the minimumDepth it is raised to the minimumDepth.

ActivationExample.java: testMaxActivate

```
01  private static void testMaxActivate() {
02 |     storeSensorPanel();
03 |     // note that the maximum is applied to the retrieved root object and limits activation
04 |     // further down the hierarchy
05 |     Configuration configuration = Db4o.newConfiguration();
06 |     configuration.objectClass(SensorPanel.class).maximumActivationDepth(2);
07 |
08 |     ObjectContainer container = Db4o.openFile(configuration, DB4O_FILE_NAME);
09 |     try {
10 |         System.out.println("Maximum activation depth = 2 (default = 5)");
11 |         ObjectSet result = container.get(new SensorPanel(1));
12 |         listResult(result);
13 |         if (result.size() > 0) {
14 |             SensorPanel sensor = (SensorPanel) result.get(0);
15 |             SensorPanel next = sensor.next;
16 |             while (next != null) {
17 |                 System.out.println(next);
18 |                 next = next.next;
19 |             }
20 |         }
21 |     } finally {
22 |         container.close();
23 |     }
24 }
```

ActivationExample.java: testMinActivate

```
01  private static void testMinActivate(){
```

```

02 |     storeSensorPanel();
03 |     // note that the minimum applies for *all* instances in the hierarchy
04 |     // the system ensures that every instantiated List object will have it's
05 |     // members set to a depth of 1
06 |     Configuration configuration = Db4o.newConfiguration();
07 |     configuration.objectClass(SensorPanel.class).minimumActivationDepth(1);
08 |     ObjectContainer container = Db4o.openFile(configuration, DB4O_FILE_NAME);
09 |     try {
10 |         System.out.println("Minimum activation depth = 1");
11 |         ObjectSet result = container.get(new SensorPanel(1));
12 |         listResult(result);
13 |         if (result.size() > 0) {
14 |             SensorPanel sensor = (SensorPanel) result.get(0);
15 |             SensorPanel next = sensor.next;
16 |             while (next != null) {
17 |                 System.out.println(next);
18 |                 next = next.next;
19 |             }
20 |         }
21 |     } finally {
22 |         container.close();
23 |     }
24 |

```

You can set up automatic activation for specific objects or fields:

Java:

```

configuration.objectClass("yourClass").cascadeOnActivate (bool)
configuration.objectClass("yourClass").objectField("field").cascadeOnActivate(bool)

```

Cascade activation will retrieve the whole object graph, starting from the specified object(field). This setting can lead to increased memory consumption.

ActivationExample.java: testCascadeActivate

```

01 |     private static void testCascadeActivate() {
02 |         storeSensorPanel();
03 |         Configuration configuration = Db4o.newConfiguration();
04 |         configuration.objectClass(SensorPanel.class).cascadeOnActivate(true);
05 |         ObjectContainer container = Db4o.openFile(configuration, DB4O_FILE_NAME);
06 |         try {
07 |             System.out.println("Cascade activation");
08 |             ObjectSet result = container.get(new SensorPanel(1));

```

```

09 |     listResult(result);
10 |     if (result.size() >0) {
11 |         SensorPanel sensor = (SensorPanel) result.get(0);
12 |         SensorPanel next = sensor.next;
13 |         while (next != null){
14 |             System.out.println(next);
15 |             next = next.next;
16 |         }
17 |     }
18 | } finally {
19 |     container.close();
20 | }
21 |

```

An alternative to cascade activation can be manual activation of objects:

Java: ObjectContainer#activate(object, activationDepth);

Manual deactivation may be used to save memory:

Java: ObjectContainer#deactivate(object, activationDepth);

These 2 methods give you an excellent control over object activation, but they obviously need more attention from the application side.

#### ActivationExample.java: testActivateDeactivate

```

01 | private static void testActivateDeactivate(){
02 |     storeSensorPanel();
03 |     Configuration configuration = Db4o.newConfiguration();
04 |     configuration.activationDepth(0);
05 |     ObjectContainer container = Db4o.openFile(configuration, DB4O_FILE_NAME);
06 |     try {
07 |         System.out.println("Object container activation depth = 0" );
08 |         ObjectSet result = container.get(new SensorPanel(1));
09 |         System.out.println("Sensor1: ");
10 |         listResult(result);
11 |         SensorPanel sensor1 = (SensorPanel) result.get(0);
12 |         testActivated(sensor1);
13 |
14 |         System.out.println("Sensor1 activated: ");
15 |         container.activate(sensor1, 4);
16 |         testActivated(sensor1);

```

```
17 |
18 |     System.out.println("Sensor5 activated:");
19 |     result = container.get(new SensorPanel(5));
20 |     SensorPanel sensor5 = (SensorPanel)result.get(0);
21 |     container.activate(sensor5, 4);
22 |     listResult(result);
23 |     testActivated(sensor5);
24 |
25 |     System.out.println("Sensor1 deactivated:");
26 |     container.deactivate(sensor1, 5);
27 |     testActivated(sensor1);
28 |
29 |     //          DANGER !!!.
30 |     // If you use deactivate with a higher value than 1
31 |     // make sure that you know whereto members might branch
32 |     // Deactivating list1 also deactivated list5
33 |     System.out.println("Sensor 5 AFTER DEACTIVATE OF Sensor1.");
34 |     testActivated(sensor5);
35 | } finally {
36 |     container.close();
37 | }
38 }
```

This revision (13) was last Modified 2007-10-30T05:05:26 by Tetyana.

# Transparent Activation Framework

Activation is a db4o-specific mechanism, which controls object instantiation in a query result. Activation works in several modes and is configurable on a database, object or field level. For more information see [Activation](#).

Using activation in a project with deep object hierarchies and many cross-references on different levels can make activation strategy complex and difficult to maintain. Transparent Activation (TA) project was started to eliminate this problem and make activation automatic in the same time preserving the best performance and the lowest memory consumption.

With TA enabled, objects are fetched on demand and only those that are used are being loaded.

More Reading:

- [TA Implementation](#)
- [TA Enhanced Example](#)
- [Detailed Example](#)
- [Collection Example](#)
- [Object Types In TA](#)
- [TA Diagnostics](#)

This revision (9) was last Modified 2007-10-28T14:45:04 by Tetyana.

# TA Implementation

The basic idea for Transparent Activation:

- Classes can be modified to activate objects on demand by implementing the Activatable interface.
- To add the Activatable code to classes you can choose from one of the following three options:
  - Let db4o tools add the code to your persistent classes at compile time.
  - Use a special ClassLoader to add the code to persistent classes at load time.
  - Write the Activatable code by hand.
- To instruct db4o to operate in Transparent Activation mode, call:  
`Db4o.configure().add(new TransparentActivationSupport());`
- In Transparent Activation mode when objects are returned from a query:
  - objects that implement the Activatable interface will not be activated immediately
  - objects that do not implement the Activatable interface will be fully activated. Activatable objects along the graph of members break activation.
- Whenever a field is accessed on an Activatable object, the first thing that is done before returning the field value is checking it's activation state and activating the parent object if it is not activated. Similar as in querying, members that implement Activatable will not be activated themselves. Members that do not implement Activatable will be fully activated until Activatable objects are found.

With Transparent Activation the user does not have to worry about manual activation at all. Activatable objects will be activated on demand. Objects that do not implement Activatable will always be fully activated.

The basic sequence of actions to get this scheme to work is the following:

1. Whenever an object is instantiated from db4o, the database registers itself with this object. To enable this on the database level `TransparentActivationSupport` has to be registered with the db4o configuration. On the object level an object is made available for TA by implementing the `Activatable`/`IActivatable` interface and providing the according `bind(activator)` method. The default implementation of the bind method stores the given `activator` reference for later use. Note, that only one activator can be associated with an object: trying to bind a different activator (from another object container) will result in an exception. More on this in [Migrating Between Databases](#).
2. All methods that are supposed to require activated object fields should call `activate(ActivationPurpose)`/`Activate(ActivationPurpose)` at the beginning of the message body. This method will check whether the object is already activated and if this is not the case, it will act depending on which activation reason was supplied.
3. The `ActivationPurpose` can be `READ` or `WRITE`. `READ` is used when an object field is requested for viewing by an application. In this case `Activate` method will request the container to activate the object to level 1 and set the activated flag accordingly (more on this case in the following chapters). `WRITE` activation purpose is used when an object is about to be changed; a simple example is setter methods. In this case the object is activated to depth 1 and registered for update. More on `ActivationPurpose`. Write in [Transparent Persistence](#).

This implementation requires quite many modifications to the objects. That is why db4o provides an automated TA implementation through bytecode instrumentation. With this approach all the work for TA is done behind the scenes.

Automatic and manual TA approaches are discussed in detail in the following examples.

- [TA Enhanced Example](#)
- [Detailed Example](#)
- [Collection Example](#)

This revision (20) was last Modified 2008-01-12T10:54:17 by Tetyana.

# TA Enhanced Example

## Contents

- [TA Enhancement In Java](#)

As it was mentioned [before](#) you can inject TA awareness in your persistent classes without modifying their original code. In the current scenario this means:

- generate the Activatable interface declaration;
- add `bind(objectContainer)` method implementation;
- generate a field to keep a reference to the corresponding Activator instance;
- generate `activate()` call at the beginning of every method.

These tasks can be fulfilled in the classes bytecode by using [Enhancement Tools](#).

## TA Enhancement In Java

---

TA can be enabled by bytecode injection of the above-mentioned code into the persistent classes when they are loaded or built. (Currently persistent classes have to be "tagged" by providing an appropriate `ClassFilter` instance.) In addition to this db4o also explicitly needs to be configured to use the Transparent Activation instrumentation of the persistent classes (`TransparentActivationSupport`).

Transparent Activation functionality requires the following jars:

- bloat-1.0.jar
- db4o-x.x-tools.jar
- db4o-x.x-taj.jar
- db4o-x.x-instrumentation.jar

The following topics explain how TA enhancement can be applied to built classes:

- [TA Enhancement At Loading Time](#)
- [TA Enhancement At Build Time](#)

This revision (23) was last Modified 2008-01-20T14:03:38 by Tetyana.

# TA Enhancement At Loading Time

TA Instrumentation at loading time is the most convenient as the classes do not have to be modified, only a separate runner class should be created to enable special instrumenting classloader to deal with the classes.

Let's look at an example.

We will use [SensorPanel](#) class from the [Activation](#) example.

The following configuration should be used (note that reflector set-up is not necessary for the loading time instrumentation):

TAInstrumentationExample.java: configureTA

```

01 private static Configuration configureTA() {
02 |     Configuration configuration = Db4o.newConfiguration();
03 |     configuration.add(new TransparentActivationSupport());
04 |     // configure db4o to use instrumenting classloader
05 |     // This is required for build time optimization!
06 |     configuration.reflectWith(new JdkReflector(
07 |         TAIInstrumentationExample.class.getClassLoader()));
08 |
09 |     return configuration;
10 }
```

The main method should provide the testing code:

TAInstrumentationExample.java: main

```

1 public static void main(String[] args) {
2 |     testActivation();
3 }
```

TAInstrumentationExample.java: storeSensorPanel

```

01 private static void storeSensorPanel() {
02 |     new File(DB40_FILE_NAME).delete();
03 |     ObjectContainer container = database(configureTA());
04 if (container != null) {
05 try {
06 |     // create a linked list with length 10
07 |     SensorPanel list = new SensorPanel().createList(10);
08 |     container.store(list);
09 } finally {
```

```

10 |         closeDatabase();
11 |     }
12 | }
13 L }
```

## TAInstrumentationExample.java: testActivation

```

01 private static void testActivation() {
02     storeSensorPanel();
03     Configuration configuration = configureTA();
04     activateDiagnostics(configuration);
05
06     ObjectContainer container = database(configuration);
07     if (container != null) {
08         try {
09             Query query = container.query();
10             query.constrain(SensorPanel.class);
11             query.descend("_sensor").constrain(new Integer(1));
12             ObjectSet result = query.execute();
13             listResult(result);
14         if (result.size() > 0) {
15             SensorPanel sensor = (SensorPanel) result.get(0);
16             SensorPanel next = sensor._next;
17             while (next != null) {
18                 System.out.println(next);
19                 next = next._next;
20             }
21         }
22     } finally {
23         closeDatabase();
24     }
25 }
26 L }
```

A separate class should be used to run the instrumented example. This class creates a filter to point to the classes that should be instrumented, in this case ByNameClassFilter is used. You can see other filters in ClassFilter hierarchy. A special BloatClassEdit is created to instruct, which type of instrumentation will be used (InjectTransparentActivationEdit in our case). This configuration together with the path of the classes to be instrumented is passed to Db4oInstrumentationLauncher.

## TAInstrumentationRunner.java

```

01 /* Copyright (C) 2007 db4objects Inc. http://www.db4o.com */
02
```

```

03 package com.db4odoc.taexamples.instrumented;
04
05 import java.io.File;
06 import java.net.URL;
07
08 import com.db4o.instrumentation.clazzfilter.ByNameClassFilter;
09 import com.db4o.instrumentation.core.BloatClassEdit;
10 import com.db4o.instrumentation.core.ClassFilter;
11 import com.db4o.instrumentation.main.Db4oInstrumentationLauncher;
12 import com.db4o.ta.instrumentation.InjectTransparentActivationEdit;
13
14
15 public class TAIInstrumentationRunner {
16 |
17     public static void main(String[] args) throws Exception {
18         // list the classes that need to be instrumented
19         ClassFilter filter = new ByNameClassFilter(new String[] { SensorPanel.class.getName()
() });
20         // inject TA awareness
21         BloatClassEdit edits[] = new BloatClassEdit[] { new InjectTransparentActivationEdit
(filter) };
22         // get URL for the classloader
23         URL[] urls = { new File("e:\\sb4o\\trunk\\reference\\bin").toURI().toURL() };
24         Db4oInstrumentationLauncher.launch(edits, urls, TAIInstrumentationExample.class.getName(),
() , new String[] {} );
25 |
26     }
27     // end main
28 |
29 }
```

You can run the example by running Db4oInstrumentationLauncher, which will start TAIInstrumentationExample in a correct configuration.

This revision (2) was last Modified 2007-10-28T18:16:59 by Tetyana.

# TA Enhancement At Build Time

This topic applies to Java version only.

In the [previous topic](#) we discussed how TA can be enabled on classes while they are loaded. In this topic we will look at even more convenient and performant way of enhancing classes to support TA: during application build time.

For our example we will take the same classes as in the [previous example](#), with the exception of TAInstrumentationRunner class, which won't be needed for build-time enhancement. Basically, we will move all the enhancing functionality of TAInstrumentationRunner into the build script. For this example we will create an ant script, which should be run after the classes (or jar) is built.

For simplistic example our build script should:

- Use classes, created by normal build script
- Create a new enhanced-bin folder for the enhanced classes
- Use TAAntClassEditFactory to create InjectTransparentActivationEdit (can be based on class filter)
- Call Db4oFileEnhancerAntTask#execute, which will call Db4oClassInstrumenter#enhance passing the previously created InjectTransparentActivationEdit to instrument classes with TA.

All these can be done with the following script:

## Build.Xml

```

01 <?xml version="1.0"?>
02
03 <!--
04   TA build time enhancement sample.
05 -->
06
07 <project name="taenhance" default="buildall">
08
09 <!--
10   Set up the required class path for the enhancement task.
11   In a production environment, this will be composed of jars, of course.
12 -->
13 <path id="db4o.enhance.path">
14   <pathelment path="${basedir}" />
15   <fileset dir="lib">
16     <include name="**/*.jar"/>
17   </fileset>
18 </path>
19
20 <!-- Define enhancement task. -->
21 <taskdef

```

```

22   name="db4o-enhance"
23   classname="com.db4o.instrumentation.ant.Db4oFileEnhancerAntTask"
24   classpathref="db4o-enhance.path"
25   loaderref="db4o-enhance.loader" />
26
27 <typedef
28   name="transparent-activation"
29   classname="com.db4o.ta.instrumentation.ant.TAAntClassEditFactory"
30   classpathref="db4o-enhance.path"
31   loaderref="db4o-enhance.loader" />
32
33
34
35 <target name="buildall">
36
37     <!-- Create enhanced output directory-->
38     <mkdir dir="${basedir}/enhanced-bin" />
39     <delete dir="${basedir}/enhanced-bin" quiet="true">
40         <include name="**/*" />
41     </delete>
42
43     <db4o-enhance classTargetDir="${basedir}/enhanced-bin" jarTargetDir="${basedir}/enhanced-lib">
44
45         <classpath refid="db4o-enhance.path" />
46             <!-- Use compiled classes as an input -->
47         <sources dir="${basedir}/bin" />
48
49             <!-- Call transparent activation enhancement -->
50         <transparent-activation />
51
52     </db4o-enhance>
53
54 </target>
55
56
57
58 </project>
```

In order to test this script:

- Create a new project, consisting of TAInstrumentationExample and SensorPanel classes from the [previous example](#)
  - Add lib folder to the project root and copy the following jars from db4o distribution:
    - ant.jar
    - bloat-1.0.jar
    - db4o-x.x-instrumentation.jar
    - db4o-x.x-java5.jar
    - db4o-x.x-taj.jar
    - db4o-x.x-tools.jar
- (Note, that the described functionality is only valid for db4o releases after 7.0)
- Build the project with your IDE or any other build tools (it is assumed that the built class files go to the project's bin directory)
  - Copy build.xml into the root project folder and execute it

Successfully executed build script will produce an instrumented copy of the project classes in enhanced-bin folder. You can check the results by running the following batch file from bin and enhanced-bin folders:

```
set CLASSPATH=. ; {$PROJECT_ROOT}\lib\db4o-x.x-java5.jar  
java com.db4odoc.taexamples.enhancer.TAInstrumentationExample
```

(In enhanced version the warning about classes that do not support TA should disappear).

Of course, the presented example is very simple and limited in functionality. In fact you can do a lot more things using the build script:

- Add NQ optimization in the same enhancer task
- Use ClassFilter to select classes for enhancement
- Use regex to select classes for enhancement
- Use several source folders
- Use jar as the source for enhancement

An example of the above features can be found in our [Project Spaces](#).

This revision (8) was last Modified 2007-11-27T19:39:00 by Tetyana.

# TA Enhancement With Db4oTool

This topic applies to .NET version only

TA instrumentation can be done by applying

[Db4oTool](#) utility to the ready .NET assemblies:

```
Db4oTool -ta assembly
```

Let's look at a simple example. We will use SensorPanel class from [Activation](#) example:

In your code you will need to add Transparent Activation support to the configuration:

Compile and run the application. Now, you can add TA support by using the following command-line:

```
Db4oTool -ta TAExamples.exe
```

use -vv option for verbose output:

```
Db4oTool -ta -vv TAExamples.exe
```

You can also apply type filter to TA enable only selected types:

```
Db4oTool.exe -vv -ta -by-name:S* TAExamples.exe
```

Db4oTool uses .NET regex to parse the -by-name parameter, in the example above all types starting with "S" will be TA enabled.

Run TA enabled assembly and compare results to the previous run.

This revision (2) was last Modified 2007-12-03T19:09:19 by Tetyana.

# SensorPanel

SensorPanel.java

```
01 /* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.taexamples.instrumented;
04
05 public class SensorPanel {
06 |
07     public Object _sensor;
08 |
09     public SensorPanel _next;
10 |
11     public SensorPanel() {
12         // default constructor for instantiation
13     }
14 |
15     public SensorPanel(int value) {
16         _sensor = new Integer(value);
17     }
18 |
19     public SensorPanel createList(int length) {
20         return createList(length, 1);
21     }
22 |
23     public SensorPanel createList(int length, int first) {
24         int val = first;
25         SensorPanel root = newElement(first);
26         SensorPanel list = root;
27         while (--length > 0) {
28             list._next = newElement(++val);
```

```
29 |         list = list._next;
30 |     }
31 |     return root;
32 | }
33 |
34 [+] protected SensorPanel newElement(int value) {
35 |     return new SensorPanel(value);
36 | }
37 |
38 [+] public String toString() {
39 |     return "Sensor #" + _sensor;
40 | }
41 }
```

This revision (1) was last Modified 2007-10-28T17:00:30 by Tetyana.

# SensorPanelTA

deprecated

This revision (2) was last Modified 2007-10-28T16:59:51 by Tetyana.

# Detailed Example

Let's look at the manual Transparent Activation implementation. This example will help you to understand how TA is implemented under the hood.

We will take the example class from the [Activation](#) chapter and modify it to enable TA:

- implement Activatable interface (bind method)
- add \_activator variable to keep the current activator;
- create activate() method;
- call activate() method each time field objects are required.

SensorPanelTA.java

```

01  /* Copyright (C) 2004 - 2007 db4objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.taexamples;
04
05 import com.db4o.activation.*;
06 import com.db4o.ta.*;
07
08 public class SensorPanelTA /*must implement Activatable for TA*/ implements Activatable {
09 |
10 |     private Object _sensor;
11 |
12 |     private SensorPanelTA _next;
13 |
14 |     /*activator registered for this class*/
15 |     transient Activator _activator;
16 |
17 |     public SensorPanelTA() {
18 |         // default constructor for instantiation
19 |     }
20 |
21 |     public SensorPanelTA(int value) {
22 |         _sensor = new Integer(value);
23 |     }
24 |
25 |     /*Bind the class to the specified object container, create the activator*/
26 |     public void bind(Activator activator) {
27 |         if (_activator == activator) {

```

```
28 |         return;
29 |     }
30 |     if (activator != null && _activator != null) {
31 |         throw new IllegalStateException();
32 |     }
33 |     _activator = activator;
34 | }
35 |
36 |     /*Call the registered activator to activate the next level,
37 |      * the activator remembers the objects that were already
38 |      * activated and won't activate them twice.
39 | */
40 | public void activate(ActivationPurpose purpose) {
41 |     if (_activator == null)
42 |         return;
43 |     _activator.activate(purpose);
44 | }
45 |
46 | public SensorPanelTA getNext() {
47 |     /*activate direct members*/
48 |     activate(ActivationPurpose.READ);
49 |     return _next;
50 | }
51 |
52 | public Object getSensor() {
53 |     /*activate direct members*/
54 |     activate(ActivationPurpose.READ);
55 |     return _sensor;
56 | }
57 |
58 | public SensorPanelTA createList(int length) {
59 |     return createList(length, 1);
60 | }
61 |
62 | public SensorPanelTA createList(int length, int first) {
63 |     int val = first;
64 |     SensorPanelTA root = newElement(first);
65 |     SensorPanelTA list = root;
66 |     while (--length > 0) {
67 |         list._next = newElement(++val);
```

```

68 |         list = list._next;
69 |     }
70 |     return root;
71 | }
72 |
73 @@ protected SensorPanelTA newElement(int value) {
74 |     return new SensorPanelTA(value);
75 | }
76 |
77 @@ public String toString() {
78 |     return "Sensor #" + getSensor();
79 | }
80 }

```

As you can see from the example class we can call `activate()` to activate the field objects. However, transparent activation is currently not available directly on field variables, you will have to wrap them into methods.

Let's create a configuration for transparent activation:

TAExample.java: `configureTA`

```

1 @@ private static Configuration configureTA() {
2 |     Configuration configuration = Db4o.newConfiguration();
3 |     // add TA support
4 |     configuration.add(new TransparentActivationSupport());
5 |     // activate TA diagnostics to reveal the classes that are not TA-enabled.
6 |     activateDiagnostics(configuration);
7 |     return configuration;
8 }

```

We can test TA using the configuration above:

TAExample.java: `storeSensorPanel`

```

01 @@ private static void storeSensorPanel() {
02 |     new File(DB4O_FILE_NAME).delete();
03 |     ObjectContainer container = database(Db4o.newConfiguration());
04 @@ if (container != null) {
05 @@     try {
06 |         // create a linked list with length 10
07 |         SensorPanelTA list = new SensorPanelTA().createList(10);
08 |         container.store(list);

```

```

09 } finally {
10 |     closeDatabase();
11 |
12 | }
13 L }
```

## TAExample.java: testActivation

```

01 private static void testActivation() {
02 |     storeSensorPanel();
03 |     Configuration configuration = configureTA();
04 |
05 |     ObjectContainer container = database(configuration);
06 |     if (container != null) {
07 |         try {
08 |             ObjectSet result = container.queryByExample(new SensorPanelTA(1));
09 |             listResult(result);
10 |             if (result.size() > 0) {
11 |                 SensorPanelTA sensor = (SensorPanelTA) result.get(0);
12 |                 // the object is a linked list, so each call to next()
13 |                 // will need to activate a new object
14 |                 SensorPanelTA next = sensor.getNext();
15 |                 while (next != null) {
16 |                     System.out.println(next);
17 |                     next = next.getNext();
18 |                 }
19 |             }
20 |         } finally {
21 |             closeDatabase();
22 |         }
23 |     }
24 L }
```

This revision (9) was last Modified 2007-09-30T10:39:35 by Tetyana.

# Collection Example

Db4o provides proprietary implementations for Map and List interfaces. Both implementations, when instantiated as a result of a query, are transparently activated when internal members are required to perform an operation. Db4o implementations provide an important advantage over JDK collections when running in transparent activation mode, based on the ability to control their activation.

ArrayList4 implements the List interface using an array to store elements. When an ArrayList4 instance is activated all the elements of the array are loaded into memory. On the other hand, ArrayMap4 implements the Map interface using two arrays to store keys and values. When an ArrayMap4 instance is activated all the elements of the arrays are loaded into memory.

We will use a Team class with a collection of Pilot objects:

Team.java

```
01 /* Copyright (C) 2007 db4objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.taexamples;
04
05 import java.util.*;
06
07 import com.db4o.activation.*;
08 import com.db4o.collections.*;
09 import com.db4o.ta.*;
10
11
12 public class Team implements Activatable {
13 |
14 |     List<Pilot> _pilots = new ArrayList4<Pilot>();
15 |
16 |     String _name;
17 |
18 |     //TA Activator
19 |     transient Activator _activator;
```

```
20 |
21 | // Bind the class to an object container
22|⊕⊕ public void bind(Activator activator) {
23|⊕⊕     if (_activator == activator) {
24|         return;
25|     }
26|⊕⊕     if (activator != null && _activator != null) {
27|         throw new IllegalStateException();
28|     }
29|     _activator = activator;
30| }
31|
32| // activate object fields
33|⊕⊕ public void activate(ActivationPurpose purpose) {
34|     if (_activator == null) return;
35|     _activator.activate(purpose);
36| }
37|
38|⊕⊕ public void addPilot(Pilot pilot) {
39|     // activate before adding new pilots
40|     activate(ActivationPurpose.WRITE);
41|     _pilots.add(pilot);
42| }
43|
44|⊕⊕ public void listAllPilots() {
45|     // activate before printing the collection members
46|     activate(ActivationPurpose.READ);
47|
48|⊕⊕     for (Iterator<Pilot> iter = _pilots.iterator(); iter.hasNext();) {
49|         Pilot pilot = (Pilot) iter.next();
50|         System.out.println(pilot);
51|     }
52| }
```

53 }

Pilot.java

```
01 /* Copyright (C) 2007 db4o objects Inc. http://www.db4o.com */  
02  
03 package com.db4odoc.taexamples;  
04  
05 import com.db4o.activation.*;  
06 import com.db4o.ta.Activatable;  
07  
08 public class Pilot implements Activatable {  
09 |  
10 |     private String _name;  
11 |  
12 |     transient Activator _activator;  
13 |  
14 public Pilot(String name) {  
15 |         _name = name;  
16 |     }  
17 |  
18 |     // Bind the class to an object container  
19 public void bind(Activator activator) {  
20     if (_activator == activator) {  
21         return;  
22     }  
23     if (activator != null && _activator != null) {  
24         throw new IllegalStateException();  
25     }  
26     _activator = activator;  
27 }  
28 |
```

```

29 | // activate the object fields
30 | public void activate(ActivationPurpose purpose) {
31 |     if (_activator == null)
32 |         return;
33 |     _activator.activate(purpose);
34 | }
35 |
36 | public String getName() {
37 |     // even simple String needs to be activated
38 |     activate(ActivationPurpose.READ);
39 |     return _name;
40 | }
41 |
42 | public String toString() {
43 |     // use getName method, which already contains activation call
44 |     return getName();
45 | }
46 |}

```

Store and retrieve.

#### TAExample.java: storeCollection

```

01 | private static void storeCollection() {
02 |     new File(DB40_FILE_NAME).delete();
03 |     ObjectContainer container = database(configureTA());
04 |     if (container != null) {
05 |         try {

```

```

06 |     Team team = new Team();
07 |     for (int i = 0; i < 10; i++) {
08 |         team.addPilot(new Pilot("Pilot #" + i));
09 |
10 |     container.store(team);
11 |     container.commit();
12 | } catch (Exception ex) {
13 |     ex.printStackTrace();
14 | } finally {
15 |     closeDatabase();
16 | }
17 |
18 }

```

## TAExample.java: testCollectionActivation

```

01 | private static void testCollectionActivation() {
02 |     storeCollection();
03 |     ObjectContainer container = database(configureTA());
04 |     if (container != null) {
05 |         try {
06 |             Team team = (Team) container.queryByExample(new Team()).next();
07 |             // this method will activate all the members in the collection
08 |             team.listAllPilots();
09 |         } catch (Exception ex) {
10 |             ex.printStackTrace();
11 |         } finally {
12 |             closeDatabase();
13 |         }
14 |     }
15 }

```

This revision (12) was last Modified 2007-10-31T14:03:50 by norberto.goussies.

# Object Types In TA

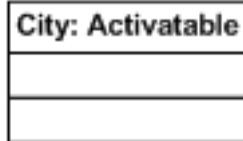
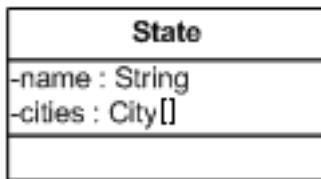
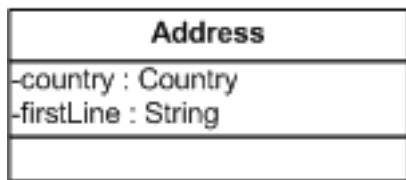
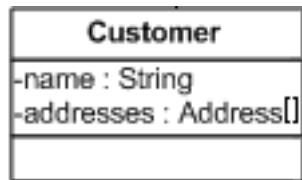
When working in TA enabled environment you must remember that db4o treats Activatable (TA Aware) and non Activatable (other) types differently.

In general we can distinguish the following types:

- Value types with no identity (char, boolean, integer etc). These types are handled internally by db4o engine and behave the same in TA enabled and disabled modes.
- Activatable types, as it is clear from the name, implement Activatable interface and are responsible for their own activation.
- Non Activatable type - all the other types, including user types or third-party classes.

As it was mentioned [before](#) in TA enabled mode non-Activatable types are fully activated whereas Activatable types have 0 activation depth and are getting activated as requested.

Let's look at an example model below, which includes Activatable and non-Activatable classes:



Querying and traversing in TA enabled mode:

Java:

```
Customer c = container.get(Customer.class).next();
```

At this point the following paths should be already activated (Customer is not Activatable):

```
c.name  
c.addresses  
c.addresses[N].firstLine  
c.addresses[N].country - available but not activated (Activatable type).
```

Country.getState would cause the Country object to be activated

Java:

```
State state = c.getAddress(0).country().getState(someZipCode);
```

At this point the following paths become activated

```
c.addresses[0].country.states  
c.addresses[0].country.states[N].name  
c.addresses[0].country.states[N].city  
c.addresses[0].country.states[N].cities[N] - available but not activated (Activatable type)
```

The following general rules apply:

1. Arrays of Arrays of non Activatable types: non Activatable behavior
2. Arrays of Arrays of Activatable types: non Activatable behavior except for leaves
3. JDK collections: non Activatable behavior
4. Value types with references to non Activatable reference types and to Activatable reference types: the non Activatable path should be activated fully; Activatable path stops activation.

This revision (7) was last Modified 2007-12-03T18:08:01 by Tetyana.

# TA Diagnostics

You can use [Diagnostics](#) to get runtime information about classes with and without TA support. Add a call to the following method in the `configureTA()` method and run the example from the [previous topic](#):

TAExample.java: activateDiagnostics

```
01  private static void activateDiagnostics(Configuration configuration) {  
02 |     // Add diagnostic listener that will show all the classes that are not  
03 |     // TA aware.  
04  |     configuration.diagnostic().addListener(new DiagnosticListener() {  
05  |         public void onDiagnostic(Diagnostic diagnostic) {  
06  |             if (!(diagnostic instanceof NotTransparentActivationEnabled)) {  
07 |                 return;  
08 |             }  
09 |             System.out.println(diagnostic.toString());  
10 |         }  
11 |     });  
12 }
```

The example should show you diagnostic messages about the classes without TA support. In this case it should be `Image` class (`Pilot._image`) and `BlobImpl` (used in `Image` class).

This revision (6) was last Modified 2007-06-06T12:26:08 by Tetyana.

# Update Depth

Update depth term is used to determine a number of levels of member objects down the hierarchy, which will be updated automatically, when the top-level object is updated. Understanding of update depth and its performance impact is especially important for working with deep object graphs and collections.

The default update depth value is 0, which means that `objectContainer#set(object)` method will only update the object passed as a parameter and any changes to its member objects will be lost.

Update depth can be changed globally or for a specific class.

Global update depth settings:

Java:

```
configuration.updateDepth(depth)
```

Class-specific update settings:

Java:

```
configuration.objectClass(ListObject.class).cascadeOnUpdate(flag);
```

```
configuration.objectClass(ListObject.class).updateDepth(depth);
```

Here `depth` parameter specifies the number of member object levels down the hierarchy, which will be updated when the top-level object is saved.

Flag parameter determines whether update operation should be cascaded to all the member objects.

Further reading:

[Update Depth for Structured Objects](#)

[Collections Update Depth](#)

This revision (7) was last Modified 2008-03-29T10:29:38 by Tetyana.

# Collections Update Depth

When you work with collections you should pay a special attention to your update depth setting, as the performance impact of this setting increases with the number of objects in the collection.

For example let's consider a class like this:

Java:

```
class ListObject {  
  
    List <DataObject> data;  
  
}
```

Let's assume that ListObject has a data list of 1000 DataObjects.

## Update depth = 1

data field object (List) will be updated if ListObject is saved.

## Update depth = 2

data object (List) and all 1000 DataObjects in the list will be updated if ListObject is saved.

It is easy to see that after a certain update depth value all the list objects are getting updated, which produces a serious performance penalty. The following examples show how to avoid the performance drop and still get the expected results.

More Reading:

- [Insert And Remove](#)
- [Updating List Objects](#)

This revision (4) was last Modified 2007-02-22T13:06:38 by Tetyana.

# Insert And Remove

When an object is inserted or removed from the list, only the list object needs to be updated, the objects in the list are not going to be changed. That means that for the object from the [previous topic](#) we will need to set update depth = 1.

Let's fill up the database with 2 long lists of objects:

ListOperationsExample.java: fillUpDb

```

01 private static void fillUpDb()
02 {
03     int listCount = 2;
04     int dataCount = 50000;
05     long elapsedTime = 0;
06     new File(DBFILE).delete();
07     ObjectContainer db = Db4o.openFile(DBFILE);
08     try
09     {
10         long t1 = System.currentTimeMillis();
11
12         for (int i = 0; i < listCount; i++)
13         {
14             ListObject lo = new ListObject();
15             lo.setName("list" + String.format("%3d", i));
16             for (int j = 0; j < dataCount; j++)
17             {
18                 DataObject dataObject = new DataObject();
19                 dataObject.setName("data" + String.format("%5d", j));
20                 dataObject.setData( System.currentTimeMillis() + " ---- Data Object"
" + String.format("%5d", j));
21                 lo.getData().add(dataObject);
22             }
23             db.set(lo);
24         }
25         long t2 = System.currentTimeMillis();
26         elapsedTime = t2 - t1;
27     }
28     finally
29     {
30         db.close();
31     }
32     System.out.println("Completed " + listCount + " lists of " + dataCount + "

```

```
objects each.");
33 |         System.out.println("Elapsed time: " + elapsedTime + " ms.");
34 }
```

We will remove an object from one list and insert it into another:

ListOperationsExample.java: removeInsert

```
01 private static void removeInsert()
02 {
03     ObjectContainer db = Db4o.openFile(DBFILE);
04     long timeElapsed = 0;
05     try
06     {
07         // set activation depth to 1 for the quickest execution
08         db.txn().configure().updateDepth(1);
09         List<ListObject> result = db.<ListObject>query(ListObject.class);
10         if (result.size() == 2)
11         {
12             // retrieve 2 ListObjects
13             ListObject lo1 = result.get(0);
14             ListObject lo2 = result.get(1);
15             DataObject dataObject = lo1.getData().get(0);
16             // move the first object from the first
17             // ListObject to the second ListObject
18             lo1.getData().remove(dataObject);
19             lo2.getData().add(dataObject);
20
21             System.out.println("Removed from the first list, count is " + lo1.
22             getData().size());
23             System.out.println("Added to the second list, count is " + lo2.getData()
24             ().size());
25             long t1 = System.currentTimeMillis();
26             // save ListObjects. UpdateDepth = 1 will ensure that
27             // the DataObject list is saved as well
28             db.set(lo1);
29             db.set(lo2);
30             db.commit();
31             long t2 = System.currentTimeMillis();
32             timeElapsed = t2 - t1;
33         }
34 }
```

```

33 |         finally
34 |     {
35 |         db.close();
36 |     }
37 |     System.out.println("Storing took: " + timeElapsed + " ms.");
38 | }

```

Remember, if an object is removed from the list and is not going to be used any more, it should be deleted manually:

```
db.Delete(dataObject)
```

Let's check if the results are correct:

ListOperationsExample.java: checkResults

```

01 private static void checkResults()
02 {
03     ObjectContainer db = Db4o.openFile(DBFILE);
04     try
05     {
06         List<ListObject> result = db.<ListObject>query(ListObject.class);
07         if (result.size() > 0)
08         {
09             // activation depth should be enough to activate
10             // ListObject, DataObject and its list members
11             int activationDepth = 3;
12             db.ext().configure().activationDepth(activationDepth);
13             System.out.println("Result count was " + result.size() + " looping with
activation depth" + activationDepth);
14             for (int i = 0; i < result.size(); i++)
15             {
16                 ListObject lo = (ListObject)result.get(i);
17                 System.out.println("ListObj " + lo.getName() + " has " + ((lo.
getData() == null) ? "<null>" : lo.getData().size()) + " objects");
18                 System.out.println((lo.getData() != null && lo.getData().size() >
0) ? lo.getData().get(0).toString() : "<null>" + " at index 0");
19             }
20         }
21     }
22     finally
23     {
24         db.close();
25     }

```

You will see that insert/remove operation takes much less time with the correct update depth setting.

This revision (8) was last Modified 2007-06-15T15:56:59 by Tetyana.

# Updating List Objects

As we discussed [before](#) updating list members using update depth is quite inefficient. An alternative approach can be retrieving and updating each object from the list separately:

ListOperationsExample.java: updateObject

```

01 private static void updateObject()
02 {
03     long timeElapsed = 0;
04
05     ObjectContainer db = Db4o.openFile(DBFILE);
06     try
07     {
08         // we can set update depth to 0
09         // as we update only the current object
10         db.extent().configure().updateDepth(0);
11         List<ListObject> result = db.<ListObject>query(ListObject.class);
12         if (result.size() == 2)
13         {
14             ListObject lo1 = result.get(0);
15             // Select a DataObject for update
16             DataObject dataobject = lo1.getData().get(0);
17             dataobject.setName("Updated");
18             dataobject.setData(System.currentTimeMillis() + " - - - Updated Object ");
19
20             System.out.println("Updated list " + lo1.getName() + " dataobject " +
21 lo1.getData().get(0));
22             long t1 = System.currentTimeMillis();
23             // save only the DataObject. List of DataObjects will
24             // automatically include the new value
25             db.set(dataobject);
26             db.commit();
27             long t2 = System.currentTimeMillis();
28             timeElapsed = t2 - t1;
29         }
30     }
31     finally
32     {
33         db.close();
34     }
}

```

```
34 |     System.out.println("Storing took: " + timeElapsed + " ms. ") ;  
35 | }
```

In this case only the object of interest is updated, which takes much less time than updating the whole list.  
This revision (4) was last Modified 2007-06-15T15:55:34 by Tetyana.

# Delete Behavior

Db4o delete interface is very simple:

```
objectContainer#Delete(object)
```

Any object, stored to db4o, can be deleted in this way. However, in many cases you may want to delete not only the passed object, but also its dependent objects.

The following topics discuss how to make the deletion easy and effective in different situations:

[Deleting Structured Objects](#)

[Deleting Collection Members](#)

[Deleting Collections](#)

[Cascaded Behavior](#)

[Referential Integrity](#)

This revision (4) was last Modified 2007-05-07T08:24:24 by Tetyana.

# Deleting Collection Members

For the following examples we will use [DataObject](#) and [ListObject](#) classes. The database will be filled up with the [FillUpDb](#) method.

If you want to delete all members in a list you can use remove list function.

ListDeletingExample.java: removeTest

```

01  private static void removeTest() {
02 |     // set update depth to 1 as we only
03 |     // modify List field
04 |     Configuration configuration = Db4o.newConfiguration();
05 |     configuration.objectClass(ListObject.class).updateDepth(1);
06 |     ObjectContainer container = Db4o.openFile(configuration, DB4O_FILE_NAME);
07 |     try {
08 |         List<ListObject> result = container.<ListObject>query(ListObject.class);
09 |         if (result.size() > 0) {
10 |             // retrieve a ListObject
11 |             ListObject lo1 = result.get(0);
12 |             // remove all the objects from the list
13 |             lo1.getData().removeAll(lo1.getData());
14 |             container.set(lo1);
15 |         }
16 |     } finally {
17 |         container.close();
18 |     }
19 |     // check DataObjects in the list
20 |     // and DataObjects in the database
21 |     container = Db4o.openFile(DB4O_FILE_NAME);
22 |     try {
23 |         List<ListObject> result = container.<ListObject>query(ListObject.class);
24 |         if (result.size() > 0) {
25 |             ListObject lo1 = result.get(0);
26 |             System.out.println("DataObjects in the list: " +
27 |                 + lo1.getData().size());

```

```

28 |     }
29 |     List<DataObject> removedObjects = container
30 |         .<DataObject> query(DataObject.class);
31 |     System.out.println("DataObjects in the database: "
32 |         + removedObjects.size());
33 | } finally {
34 |     container.close();
35 | }
36 |

```

However as you will see from the example above, removed objects are not deleted from the database. Here you should be very careful: if you want to delete DataObjects, which were removed from the list you must be sure that they are not referenced by existing objects. Check

[Referential Integrity](#) article for more information.

The following example shows how to delete DataObjects from the database as well as from the list:

#### ListDeletingExample.java: removeAndDeleteTest

```

01 | private static void removeAndDeleteTest() {
02 |     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
03 |     try {
04 |         // set update depth to 1 as we only
05 |         // modify List field
06 |         container.ext().configure().objectClass(ListObject.class).updateDepth(1);
07 |         List<ListObject> result = container.<ListObject> query(ListObject.class);
08 |         if (result.size() > 0) {
09 |             // retrieve a ListObject
10 |             ListObject lo1 = result.get(0);
11 |             // create a copy of the objects list
12 |             // to memorize the objects to be deleted
13 |             List tempList = new ArrayList(lo1.getData());
14 |             // remove all the objects from the list
15 |             lo1.getData().removeAll(lo1.getData());
16 |             // and delete them from the database
17 |             Iterator<DataObject> it = tempList.iterator();
18 |             while (it.hasNext()) {

```

```
19 |         container. delete(it.next());
20 |     }
21 |
22 |     container.set(101);
23 | }
24 | } finally {
25 |     container.close();
26 | }
27 | // check DataObjects in the list
28 | // and DataObjects in the database
29 | container = Db4o.openFile(DB4O_FILE_NAME);
30 | try {
31 |     List<ListObject> result = container.<ListObject>query(ListObject.class);
32 |     if (result.size() > 0) {
33 |         ListObject lo1 = result.get(0);
34 |         System.out.println("DataObjects in the list: " +
35 |             + lo1.getData().size());
36 |     }
37 |     List<DataObject> removedObjects = container
38 |         .<DataObject>query(DataObject.class);
39 |     System.out.println("DataObjects in the database: " +
40 |         + removedObjects.size());
41 | } finally {
42 |     container.close();
43 | }
44 | }
```

This revision (7) was last Modified 2007-05-07T08:21:42 by Tetyana.

# Deleting Collections

As it was discussed in [Deleting Structured Objects](#) chapter, deleting a top-level object does not mean deleting all of the member objects. The same rule applies for collections. The recommendation would be to use cascadeOnDelete setting for a collection, which should be deleted with all its members.

For the following example we will use [DataObject and ListObject classes](#). The database will be filled up with the [FillUpDb](#) method.

ListDeletingExample.java: deleteTest

```

01  private static void deleteTest() {
02 |     ObjectContainer container = Db4o.openFile(DB40_FILE_NAME);
03 |     try {
04 |         // set cascadeOnDelete in order to delete member objects
05 |         container.ext().configure().objectClass(ListObject.class).cascadeOnDelete(
06 |             true);
07 |         List<ListObject> result = container.<ListObject>query(ListObject.class);
08 |         if (result.size() > 0) {
09 |             // retrieve a ListObject
10 |             ListObject lo1 = result.get(0);
11 |             // delete the ListObject with all the field objects
12 |             container.delete(lo1);
13 |         }
14 |     } finally {
15 |         container.close();
16 |     }
17 |     // check ListObjects and DataObjects in the database
18 |     container = Db4o.openFile(DB40_FILE_NAME);
19 |     try {
20 |         List<ListObject> listObjects = container
21 |             .<ListObject>query(ListObject.class);
22 |         System.out.println("ListObjects in the database: "
23 |             + listObjects.size());
24 |         List<DataObject> dataObjects = container
25 |             .<DataObject>query(DataObject.class);
26 |         System.out.println("DataObjects in the database: "

```

```
27 |           + dataObjects.size());  
28 | } finally {  
29 |     container.close();  
30 | }  
31 | }
```

Please, remember that there is no referential integrity check on delete: deleted objects might be referenced from elsewhere in your code.

This revision (4) was last Modified 2007-05-04T12:56:43 by Tetyana.

# Example Classes

DataObject.java

```
01 /* Copyright (C) 2007 db4objects Inc. http://www.db4o.com */  
02  
03 package com.db4odoc.listdeleting;  
04  
05 public class DataObject {  
06     String _name;  
07     String _data;  
08  
09     public DataObject()  
10    { }  
11  
12     public String getName()  
13    {  
14         return _name;  
15    }  
16  
17     public void setName(String name)  
18    {  
19         _name = name;  
20    }  
21  
22     public String getData()  
23    {  
24         return _data;  
25    }  
26  
27     public void setData(String data){  
28         _data = data;
```

```

29 |         }
30 |
31 |     public String toString()
32 |     {
33 |         return String.format("%s/%s}", _name, _data);
34 |     }
35 |

```

## ListObject.java

```

01 /* Copyright (C) 2007 db4objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.listdeleting;
04
05 import java.util.ArrayList;
06 import java.util.List;
07
08
09 public class ListObject {
10     String _name;
11     List<DataObject> _data;
12
13     public ListObject()
14     {
15         _data = new ArrayList<DataObject>();
16     }
17
18     public String getName() {
19         return _name;
20     }
21
22     public void setName(String name) {
23         _name = name;

```

```
24 |     }
25 |
26 |     public List<DataObject> getData() {
27 |         return _data;
28 |     }
29 |
30 |     public void setData(List<DataObject> data) {
31 |         _data = data;
32 |     }
33 }
```

[/filter]

This revision (1) was last Modified 2007-02-25T16:12:00 by Tetyana.

# Filling The Database

ListDeletingExample.java: fillUpDb

```

01  private static void fillUpDb(int listCount) {
02 |     int dataCount = 50;
03 |     long elapsedTime = 0;
04 |     new File(DB40_FILE_NAME).delete();
05 |     ObjectContainer container = Db4o.openFile(DB40_FILE_NAME);
06 |     try {
07 |         long t1 = System.currentTimeMillis();
08 |
09 |         for (int i = 0; i < listCount; i++) {
10 |             ListObject lo = new ListObject();
11 |             lo.setName("list" + String.format("%3d", i));
12 |             for (int j = 0; j < dataCount; j++) {
13 |                 DataObject dataObject = new DataObject();
14 |                 dataObject.setName("data" + String.format("%5d", j));
15 |                 dataObject.setData(System.currentTimeMillis()
16 |                     + " ---- Data Object " + String.format("%5d", j));
17 |                 lo.getData().add(dataObject);
18 |             }
19 |             container.set(lo);
20 |         }
21 |         long t2 = System.currentTimeMillis();
22 |         elapsedTime = t2 - t1;
23 |     } finally {
24 |         container.close();
25 |     }
26 |     System.out.println("Completed " + listCount + " lists of " + dataCount
27 |                     + " objects each.");
28 |     System.out.println("Elapsed time: " + elapsedTime + " ms.");
29 | }
```

This revision (2) was last Modified 2007-02-25T17:40:23 by Tetyana.

# Object Construction

Sometimes you may find that db4o refuses to store instances of certain classes, or appears to store them, but delivers incomplete instances on queries. To understand the problem and the alternative solutions at hand, we'll have to take a look at the way db4o "instantiates" objects when retrieving them from the database.

More Reading:

- [Creating objects](#)
- [Configuration](#)
- [Troubleshooting](#)
- [Examples](#)

This revision (5) was last Modified 2006-11-14T18:47:38 by Eric Falsken.

# Creating objects

## Contents

- [Using a constructor](#)
- [Bypassing the constructor](#)
- [Using a translator](#)

Db4o currently knows three ways of creating and populating an object from the database. The approach to be used can be configured globally and on a per-class basis.

## Using a constructor

---

The most obvious way is to call an appropriate constructor. Db4o does *not* require a public or no-args constructor. It can use any constructor that accepts default (null/0) values for all of its arguments without throwing an exception. Db4o will test all available constructors on the class (including private ones) until it finds a suitable one.

What if no such constructor exists?

## Bypassing the constructor

---

Db4o can also bypass the constructors declared for this class using platform-specific mechanisms. (For Java, this option is only available on JREs >= 1.4.) This mode allows reinstantiating objects whose class doesn't provide a suitable constructor. However, it will (silently) break classes that rely on the constructor to be executed, for example in order to populate transient members.

*If this option is available in the current runtime environment, it will be the default setting.*

## Using a translator

---

If none of the two approaches above is suitable, db4o provides a way to specify in detail how instances of a class should be stored and reinstantiated by implementing the Translator interface and registering this implementation for the offending class. [Translators chapter](#) cover this topic in detail.

This revision (7) was last Modified 2007-05-07T08:41:16 by Tetyana.

# Configuration

The instantiation mode can be configured globally or on a per class basis.

[Java:configuration.callConstructors(true)]

[/filter]

This will configure db4o to use constructors to reinstantiate any object from the database. (The default is *false*).

[filter=java]

[Java: configuration.objectClass(Foo.class).callConstructor(true)]

This will configure db4o to use constructor calls for this class and all its subclasses.

This revision (7) was last Modified 2007-04-23T07:49:50 by Tetyana.

# Troubleshooting

At least for development code, it is always a good idea to instruct db4o to check for available constructors at storage time. (If you've configured db4o to use constructors at all.)

Java: configuration.exceptionsOnNotStorable(true)

If this setting triggers exceptions in your code, or if instances of a class seem to lose members during storage, check the involved classes (especially their constructors) for problems similar to the one shown in the following section.

This revision (6) was last Modified 2007-04-23T07:50:36 by Tetyana.

# Examples

## C1.java

```
01 /* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */  
02  
03 package com.db4odoc.constructors;  
04  
05 class C1 {  
06     private String s;  
07  
08     private C1(String s) {  
09         this.s=s;  
10    }  
11  
12     public String toString() {  
13         return s;  
14    }  
15 }
```

This class cannot be cleanly reinstantiated by db4o: both approaches will fail, so one has to resort to configuring a translator.

This revision (9) was last Modified 2007-05-07T08:43:14 by Tetyana.

# Implementation Strategies

This topic set explains the specifics of db4o implementation. Going through the topics will help you to reveal the real power of db4o and learn to apply it in your environment.

More Reading:

- [Maintenance](#)
- [Type Handling](#)
- [TypeHandlers](#)
- [Using Annotations](#)
- [Enhancement Tools](#)
- [Using Attributes](#)
- [Unique Constraints](#)
- [Object Callbacks](#)
- [Callbacks](#)
- [Translators](#)
- [Db4o Reflection API](#)
- [IO Adapter](#)
- [Db4o meta-information](#)
- [LINQ Collection](#)
- [Native Query Collection](#)
- [Data Binding](#)
- [Refactoring and Schema Evolution](#)
- [Aliases](#)
- [Encryption](#)
- [Class Mapping](#)
- [IDs and UUIDs](#)
- [Freespace Management System](#)
- [Reporting](#)
- [Exceptions](#)

This revision (2) was last Modified 2007-05-07T18:01:50 by Tetyana.

# Maintenance

db4o is designed to minimize maintenance tasks to the absolute minimum. The stored class schema adapts to the application automatically as it is being developed. db4o "understands" the addition and removal of fields which allows it to continue to work against modified classes without having to reorganize the database file. Internally db4o works with a superset of all class versions previously used.

However there are two recommended maintenance tasks, that can both be fully automated remotely with API calls. They will be reviewed in the following chapters.

More Reading:

- [Defragment](#)
- [Updating Db4o File Format](#)
- [System Info](#)
- [Backup](#)

This revision (3) was last Modified 2007-05-07T17:46:52 by Tetyana.

# Defragment

Db4o database file is structured as sets of free and occupied slots, very much like a file system - and just like a file system it can be fragmented, resulting in a file that is larger than it needs to be.

Defragment tool helps to fix this problem, creating a new database file and copying all objects from the current database file to the new database file. All indexes are recreated. The resulting database file will be smaller and faster. It is recommended to apply defragmentation on a regular basis to achieve better performance.

More Reading:

- [How To Use Defragmentation](#)
- [Defragmentation Configuration](#)
- [Tracking Defragmentation Errors](#)
- [Defragmentation Examples](#)

This revision (21) was last Modified 2007-04-01T09:15:57 by Tetyana.

# How To Use Defragmentation

The simplest way to defragment a db4o file would be:

Java:

```
Defragment.defrag("filename")
```

The file must not be opened by another process during defragmentation!

This will move the file *filename* to *filename.backup*, then create a defragmented version of this file in the original position. If the backup file already exists, this will throw an IOException and no action will be taken.

You can also specify the backup filename manually:

Java:

```
Defragment.defrag(filename, backupfile);
```

For more detailed configuration of the defragmentation process, you can use a [DefragmentConfig](#) instance within the following methods:

Java:

```
Defragment.defrag(configuration);
```

```
Defragment.defrag(configuration, listener);
```

You can use listener parameter to track problems during defragmentation process. For more information see [Tracking Defragmentation Errors](#).

Defragmentation can throw IOException in the following situations:

- backup file exists
- database file not found
- database file is opened by another process

This revision (2) was last Modified 2007-04-01T09:38:30 by Tetyana.

# Defragmentation Configuration

## Contents

- Original File
- Backup File
- Mapping
- Class Filter
- Database Configuration
- Commit Frequency
- Upgrading

DefragmentConfig class allows you fine-tune the defragmentation process. This topic discusses different settings available through DefragmentConfig.

## Original File

---

The path to the file to be defragmented. Can be specified in the constructor:

```
configuration = new DefragmentConfig(origPath)
```

## Backup File

---

```
configuration = new DefragmentConfig(origPath, backupPath)
```

The path to the backup of the original file. If this file exists before the defragmentation, an IOException will be thrown and no action taken.

If you want the backup file to be deleted automatically before the defragment run, specify:

Java:

```
configuration.forceBackupDelete(true)
```

## Mapping

---

You can also specify the desired Mapping to be used internally:

```
configuration = new DefragmentConfig(origPath, backupPath, mapping)
```

mapping is an object of a class implementing ContextIDMapping interface. Mapping is used to keep track of objects moved during defragmentation. Db4o provides 2 mapping classes.

**TreeIDMapping** - default in-memory mapping. Will increase the memory usage, but is a faster alternative. Set up [objectCommitFrequency](#) to control memory usage.

**BTreeIDMapping** - mapping is done in a separate file using B-tree method. Reduces the memory usage, but is a much slower option.

## Class Filter

---

Defragmentation process uses `StoredClassFilter accept` method to define which classes should be left in a database after the defragmentation. By default, all classes are left. However, you can use `AvailableClassFilter` to get rid of the deleted classes instances:

Java:

```
configuration.storedClassFilter(new AvailableClassFilter())
```

In this case only the classes known to the classloader will be left in the database, the rest will be deleted.

## Database Configuration

---

For db4o configurations that influence low-level file layout details, it is important to provide the defragmentation process with the copy of db4o configuration:

Java:

```
configuration.db4oConfig(db4oConfiguration)
```

For more information about db4o configuration see [Configuration](#).

## Commit Frequency

---

Commit frequency sets the number of processed objects that should trigger an immediate commit of the target file. By default, frequency = 0 and commit never happens.

Java:

```
configuration.objectCommitFrequency(frequency)
```

This method can be used to reduce memory usage during defragmentation.

## Upgrading

---

You can upgrade your database file together with the defragmentation:

Java:

```
configuration.upgradeFile(tempFile)
```

This method can be used to reduce memory usage during defragmentation, however it will make it slower.

This revision (8) was last Modified 2007-11-21T13:19:14 by Tetyana.

# Tracking Defragmentation Errors

DefragmentListener/IDefragmentListener interface is provided to track system structure problems during a defragmentation process. DefragmentListener provides the following method, which will be called, when a problem is detected:

Java:

```
void notifyDefragmentInfo(DefragmentInfo info);
```

For an example of the listener implementation see [Defragmentation Examples](#).

This revision (2) was last Modified 2007-04-01T09:39:55 by Tetyana.

# Defragmentation Examples

The simplest defragmentation can look like this:

DefragmentExample.java: simplestDefragment

```

1  private static void simplestDefragment() {
2    try {
3      Defragment.defrag(DB_FILE);
4    } catch (IOException ex) {
5      System.out.println(ex.toString());
6    }
7 }
```

The following example shows how to implement defragmentation listener:

DefragmentExample.java: defragmentWithListener

```

01  private static void defragmentWithListener() {
02    DefragmentConfig config=new DefragmentConfig(DB_FILE);
03    try {
04      Defragment.defrag(config, new DefragmentListener() {
05        public void notifyDefragmentInfo(DefragmentInfo info) {
06          System.err.println(info);
07        }
08      });
09    } catch (Exception ex) {
10      System.out.println(ex.toString());
11    }
12 }
```

The following example will run defragment using TreeIDMapping and commit frequency of 1 commit per 5000 objects. The backup file will be deleted if already exists, only available to the classloader classes will be left in the database (java version) and the file will be upgraded if necessary.

DefragmentExample.java: configuredDefragment

```

01  private static void configuredDefragment() {
02    DefragmentConfig config=new DefragmentConfig(DB_FILE, BACKUP_FILE, new TreeIDMapping());
03    config.objectCommitFrequency(5000);
04    config.db4oConfig(Db4o.cloneConfiguration());
05    config.forceBackupDelete(true);
06    config.storedClassFilter(new AvailableClassFilter());
```

```
07 |     config.upgradeFile(DB_FILE + ".upg");  
08 |     try {  
09 |         Defragment.defrag(config);  
10 |     } catch (Exception ex){  
11 |         System.out.println(ex.toString());  
12 |     }  
13 | }
```

This revision (2) was last Modified 2007-04-01T09:49:42 by Tetyana.

# Updating Db4o File Format

The db4o database file format is a subject to change to allow progress for performance and additional features.

db4o does not support downgrades back to previous versions of database files.

In order to prevent accidental upgrades when using different db4o versions or ObjectManager, db4o does not upgrade databases by default.

Database upgrading can be turned on with the following configuration switch:

Java:

```
Db4o.configure().allowVersionUpdates(true)
```

Please note that, once the database file version is updated, there is no way to get back to the older version of the database file

If a database file is opened successfully with the new db4o version, the upgrade of the file will take place automatically. You can simply upgrade database files by opening and closing a db4o database once with code like the following:

**UpdateExample.java**

```
01 /* Copyright (C) 2007 db4o objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.versionupdate;
04
05 import com.db4o.Db4o;
06 import com.db4o.ObjectContainer;
07
08 public class UpdateExample {
09 |
10     public static void main(String[] args) {
11 |        Db4o.configure().allowVersionUpdates(true);
12 |        ObjectContainer objectContainer = Db4o.openFile(args[0]);
13 |        objectContainer.close();
14 |        System.out.println("The database is ready for the version " + Db4o.version());
15 |    }
16 |
17 }
```

Recommendations for upgrading:

- backup your database file to be able to switch back.
- check the content of database files before and after upgrading, using com.db4o.tools.Statistics.
- [Defragmenting](#) a database file with the new db4o version after upgrading can make the database more efficient.

This revision (4) was last Modified 2007-05-04T14:33:39 by Tetyana.

# System Info

SystemInfo is a utility class, providing system information about db4o database.

Currently SystemInfo includes the following methods:

Method	Functionality
freespaceSize	<p>returns the freespace size in the database in bytes. When db4o stores modified objects, it allocates a new slot for it. During commit the old slot is freed. Free slots are collected in the <a href="#">freespace manager</a>, so they can be reused for other objects.</p> <p>This method returns a sum of the size of all free slots in the database file.</p> <p>To reclaim freespace run <a href="#">Defragment</a>.</p>
freespaceEntryCount	returns the number of entries in the <a href="#">Freespace Manager</a> . A high value for the number of freespace entries is an indication that the database is fragmented and that <a href="#">Defragment</a> should be run.
totalSize	Returns the total size of the database on disk.

In order to get SystemInfo instance you must issue the following call:

Java:

```
SystemInfo info = container.ext().systemInfo();
```

SystemInfo can be used for different maintenance services. The example below presents a simple program, which can be scheduled to run automatically at certain intervals for a database check up. (To download the complete example, please press "download" button in the right upper corner of the code block).

SystemInfoExample.java: testSystemInfo

```

01 ①②private static void testSystemInfo() {
02 |   long dbSize = _container.ext().systemInfo().totalSize();
03 |   long fsSize = _container.ext().systemInfo().freespaceSize();
04 ③④    if (dbSize > MAX_DB_SIZE) {
05 |      System.out.println("Attention! Database file size is over the limit. Maintenance
required");
06 |    }
07 |    _logPS.println("Total database size: " + dbSize);
08 ⑤⑥    if (fsSize > MAX_FS_SIZE) {
09 |      System.out.println("Attention! Freespace size is over the limit. Maintenance
required");

```

```
10 |     }
11 |     _logPS.println("Database freespace size: " + fsSize);
12 |     _logPS.println("Database freespace entries: " + _container.ext().systemInfo().
freespaceEntryCount());
13 L }
```

This revision (2) was last Modified 2007-08-17T20:21:30 by Tetyana.

# Backup

db4o supplies hot backup functionality to backup single-user databases and client-server databases while they are running.

The respective API calls for backups are:

Java:

```
ObjectContainer.ext().backup(String path)  
ObjectServer.ext().backup(String path)
```

The methods can be called while an ObjectContainer/ObjectServer is open and they will execute with low priority in a dedicated thread, with as little impact on processing performance as possible.

It is recommended to backup the current development state of an application (ideally source code and bytecode) along with the database files since the old code can make it easier to work with the old data.

This revision (4) was last Modified 2006-11-14T17:13:44 by Tetyana.

# Type Handling

This topic set discusses special db4o classes.

More Reading:

- [Collections](#)
- [Blobs](#)
- [Static Fields And Enums](#)
- [Delegates And Events](#)
- [Final Fields](#)

This revision (2) was last Modified 2007-05-07T14:47:29 by Tetyana.

# Collections

[Translators](#) chapter of the documentation explains why translators are necessary to store and retrieve some types of classes. Db4o uses translators internally to manage storing and retrieving of collections.

Java collections were first implemented in JDK 1.2. Before that Vector implementation was used to store growable arrays of objects. Under the hood, when collection object is stored to the database different actions are taken for different versions of java:

1. before JDK1.2: collection class is translated with TVector class;
2. after: collection is translated with TCollection, TMap, THashtable translators.

In fact, the functionality of those translators is pretty much the same. On storing collection is transferred to an array of objects ( Object[] ) and that array gets stored to the database file.

Map (Java), HashTable and SortedList for .NET objects are stored as an array of objects of special type:

```
public class Entry
{
    public Object key;
    public Object value;
    . . .
}
```

*OnInstantiate* method of collection translators creates a new instance of respective collection or map and restores its values from the saved object array.

Unfortunately this implementation is not very efficient for searches/updates of a certain value in a collection, as the whole collection should be instantiated to access any of its elements.

More Reading:

- [Built-in db4o collections](#)
- [Fast collections](#)
- [Collections Or Arrays](#)

This revision (11) was last Modified 2007-05-07T09:29:51 by Tetyana.

# Built-in db4o collections

Db4o provides its own built-in collection implementation to improve performance and decrease memory consumption:

Java:

```
ObjectContainer.ext().collections.newLinkedList();
ObjectContainer.ext().collections.newHashMap();
ObjectContainer.ext().collections.newIdentityHashMap();
```

The LinkedList implementation only holds the first and the last elements in RAM, all the other objects are loaded on demand. Apparently, this implementation provides good performance for sequential traversal, but it is still quite inefficient for random selection/update.

The HashMap implementation only holds an array of hash values in RAM and loads keys and objects on demand.

Db4o collections also provide the following functionality, which helps a programmer to produce expected results with as little work as possible:

- Newly added objects are automatically persisted.
- Collection elements are automatically activated, when they are needed. The activation depth is configurable with *Db4oCollection#activationDepth(int)*
- Removed objects can be deleted automatically, if the list is configured with *Db4oCollection#deleteRemoved(boolean)*

Weak Reference system ensures that objects are freed from RAM, as soon as there are no references to them.

These implementations can be faster for some cases and slower for others.

This revision (7) was last Modified 2007-02-21T06:03:33 by Tetyana.

# Fast collections

Db4o's solution for the best collection performance and lowest memory consumption is to implement them directly on top of BTrees without an intermediate "stored-object-db4o" layer (P1Object, P1Collection, P2LinkedList).

This task is still under development, but already it makes sense to be ready to switch to the new fast collections seamlessly.

Current recommendation for collection usage with db4o is:

- Declare members of persistent classes as interface (java.util.List / System.Collections.IList).
- Create central factory method to implement concrete collection (can be switched to fast collection implementation easily).

Please, avoid the following realizations, which will make the switching more difficult:

- Declaring concrete implementations as fields in persistent classes
- Deriving from JDK collection classes
- Using third-party non-standard collections

Let's look at application design, which will allow you to upgrade your application to fast collections with the least effort.

In our example we will save a list of pilots as members of one team. To make it simple let's use the following factory class to get the proper list implementation:

## CollectionFactory.java

```
01 /* Copyright (C) 2004 - 2007 db4objects Inc. http://www.db4o.com */  
02  
03 package com.db4odoc.lists;  
04  
05 import java.util.ArrayList;  
06 import java.util.List;  
07  
08 public class CollectionFactory {  
09     public static List newList(){  
10         return new ArrayList();  
11     }  
12 }
```

```
11 }  
12 }
```

The concrete class returned by the CollectionFactory can be changed to any other collection implementation (fast collection) with the minimum coding effort.

We will use the following class as a team of pilots:

### Team.java

```
01 /* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */  
02  
03 package com.db4odoc.lists;  
04  
05 import java.util.List;  
06  
07  
08 public class Team {  
09 |   private List pilots;  
10 |   private String name;  
11 |  
12 public Team(){  
13 |     pilots = CollectionFactory newList();  
14 | }  
15 |  
16 public void setName(String name){  
17 |   this.name = name;  
18 | }  
19 |  
20 public String getName(){  
21 |   return name;  
22 | }  
23 |  
24 public void addPilot(Pilot pilot){
```

```
25 |     pilots.add(pilot);  
26 | }  
27 |  
28 [+] public Pilot getPilot(int index){  
29 |     return (Pilot)pilots.get(index);  
30 | }  
31 |  
32 [+] public void removePilot(int index){  
33 |     pilots.remove(index);  
34 | }  
35 |  
36 [+] public void updatePilot(int index, Pilot newPilot){  
37 |     pilots.set(index, newPilot);  
38 | }  
39 }
```

The idea of the new fast collection implementation is to allow select/update of collection elements without an intermediate "stored-object-db4o" layer. This will allow random activation and fast querying, thus providing a considerable performance improvement especially on big collections holding deep object graphs.

This revision (6) was last Modified 2006-11-13T18:51:54 by Tetyana.

# Collections Or Arrays

If you are planning an application with db4o, you may be asking yourself, what is better to use: collections or arrays? In the current implementation it is not really a difficult choice, as collections internally are stored as arrays, which is explained in [Collections](#) chapter. You can base your solution on the overall system design, entrusting db4o to handle the internals efficiently in both cases.

However if you are looking into the future you might want to design your system in a way that can easily be switched to the next db4o planned feature: Fast Collections. For more details see [Fast collections](#). Please, feel free to use db4o Jira to track [the feature](#) progress and be notified on the latest updates.

This revision (1) was last Modified 2007-08-26T08:47:07 by Tetyana.

# Blobs

## Contents

- [Blob](#)
- [byte\[\] array](#)

In some cases user has to deal with large binary objects (BLOBs) such as images, video, music, which should be stored in a structured way, and retrieved/queried easily. There are several challenges associated with this task:

- storage location;
- loading into RAM;
- querying interface;
- objects' modification;
- information backup;
- client/server processing.

Db4o provides you with a flexibility of using 2 different solutions for this case:

1. Blob (Java package: com.db4o.types.Blob, .NET namespace: Db4oTypes.IBlob)
2. byte[] arrays stored inside the database file

These two solutions' main features in comparison are represented below:

## Blob

---

1. every Blob gets it's own file
2. C/S communication runs asynchronous in separate thread
3. special code is necessary to store and load
4. no concerns about activation depth

## byte[] array

---

1. data in the same file
2. C/S communication runs in the normal communication thread
3. transparent handling without special concerns
4. control over activation depth may be necessary

Storing data in a byte[] array works just as storing usual objects, but this method is not always applicable/desirable. First of all, the size of the db4o file can grow over the limit (256 GB) due to the BLOB data

added. Secondly, object activation and client/server transferring logic can be an additional load for your application.

More Reading:

- [Db4o Blob Implementation](#)

This revision (6) was last Modified 2006-11-14T18:45:12 by Eric Falsken.

# Db4o Blob Implementation

Built-in db4o Blob type helps you to get rid of the problems of byte[] array, though it has its own drawbacks. Pros and Cons for the points, mentioned above:

1. every Blob gets it's own file
  - + main database file stays a lot smaller
  - + backups are possible over individual files
  - + the BLOBs are accessible without db4o
  - multiple files need to be managed
2. C/S communication runs asynchronous in separate thread
  - + asynchronous storage allows the main application thread to continue its work, while blobs are being stored
3. special code is necessary to store and load
  - it is more difficult to move objects between db4o database files
4. no concerns about activation depth
  - + big objects won't be loaded into memory as part of the activation process

Let's look, how it works.

First, BLOB storage should be defined:

```
Java: Db4o.configure().setBlobPath(storageFolder);
```

```
[/filter]
```

where storageFolder is a String value representing local or server path to store BLOBs. If that value is not defined, db4o will use the default folder "blobs" in user directory.

We will use a modified Car class, which holds reference to the car photo:

**Car.java**

```

01 /* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */
02 package com.db4odoc.blobs;
03
04 public class Car {
05     private String model;
06     private CarImage img;
07
08
09     public Car(String model) {
10         this.model =model;
11         img=new CarImage();
12         img.setFile(model+".jpg");
13     }
14
15     public CarImage getImage() {
16         return img;
17     }
18
19     public String toString() {
20         return model +"(" + img.getFile() + ")";
21     }
22 }
```

[filter=vb]VB: IBlob.ReadFrom( File )

As reading is done in a dedicated thread, you can use Blob#getStatus() in a loop to create a progress window.

The same applies to the write operation, which copies BLOB, stored with db4o, to the specified filesystem location.

Let's store some cars together with their images in our database:

**BlobExample.java: storeCars**

```

01  private static void storeCars() {
02 |     new File(DB4O_FILE_NAME).delete();
03 |     ObjectContainer container=Db4o.openFile(DB4O_FILE_NAME);
04 |     try {
05 |         Car car1=new Car("Ferrari");
06 |         container.set(car1);
07 |         storeImage(car1);
08 |         Car car2=new Car("BMW");
09 |         container.set(car2);
10 |         storeImage(car2);
11 |     } finally {
12 |         container.close();
13 |     }
14 |
}

```

CarImage is stored in the database just like normal object, no BLOB data is transferred before explicit call (Blob#readFrom in CarImage#readFile method), which copies the images to the storageFolder.

Please, note, that CarImage reference should be set to the database before uploading actual data. To get the images back to the filesystem we can run a usual query:

#### BlobExample.java: retrieveCars

```

01  private static void retrieveCars() {
02 |     ObjectContainer container=Db4o.openFile(DB4O_FILE_NAME);
03 |     try {
04 |         Query query = container.query();
05 |         query.constrain(Car.class);
06 |         ObjectSet result = query.execute();
07 |         getImages(result);
08 |     } finally {
09 |         container.close();
10 |     }
11 |
}

```

and get BLOB data using retrieved Car references:

**BlobExample.java: getImages**

```
01  private static void getImages(ObjectSet result) {  
02 	while(result.hasNext()) {  
03 	Car car = (Car)(result.next());  
04 	System.out.println(car);  
05 	CarImage img = car.getImage();  
06 	try {  
07 	img.writeFile();  
08 	} catch (java.io.IOException ex) {  
09 	System.out.print(ex.getMessage());  
10 }  
11 }  
12 }
```

Retrieved images are placed in CarImage.outFolder ("blobs\out").

So query interface operates on references - no BLOB data is loaded into memory until explicit call (Blob#writeTo). This also means, that activationDepth does not affect Blob objects and best querying performance is achieved without additional coding.

This revision (10) was last Modified 2006-11-13T18:30:48 by Tetyana.

# Static Fields And Enums

How to deal with static fields and enumerations? Do they belong to your application code or to the database? Let's have a look.

More Reading:

- [Static fields API](#)
- [Usage of static fields](#)
- [Java enumerations](#)

This revision (5) was last Modified 2006-11-18T14:17:49 by Tetyana.

# Static fields API

By default db4o does not persist static fields. Normally this is not necessary as static values are set for a class, not for an object. However you can set up db4o to store static fields if you need to implement constant or enumeration:

Java:

```
Db4o.configure().objectClass(Foo.class).persistStaticFieldValues()
```

Do not use this option unnecessarily, as it will slow down the process of opening database files and the stored objects will occupy space in the database file.

This option does not have any effect on primitive types (int, boolean, etc). Use their object alternatives instead (Integer, Boolean, etc).

When this setting is on for a specific class, all non-primitive-typed static field values of this class are stored the first time an object of the class is stored, and restored, every time a database file is opened afterwards, after class meta information is loaded for this class (when the class objects are retrieved with a query, for example).

A good example of non-primitive constant type is type-safe enumeration implementation:

PilotCategories.java

```
01 /* Copyright (C) 2004 - 2007 db4objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.staticfields;
04
05 public class PilotCategories {
06     private String qualification = null;
07
08     public static PilotCategories WINNER = new PilotCategories(
09         "WINNER");
10
11     public static PilotCategories TALENTED = new PilotCategories(
12         "TALENTED");
13
14     public static PilotCategories AVERAGE = new PilotCategories(
15         "AVERAGE");
16
17     public static PilotCategories DISQUALIFIED = new PilotCategories(
18         "DISQUALIFIED");
19
20     private PilotCategories(String qualification) {
```

```

21 |     this.qualification = qualification;
22 | }
23 |
24 @@ public PilotCategories() {
25 |
26 | }
27 |
28 @@ public void testChange(String qualification) {
29 |     this.qualification = qualification;
30 | }
31 |
32 @@ public String toString() {
33 |     return qualification;
34 | }
35 |
36 }

```

Let's use it with

Pilot.java

```

01 /* Copyright (C) 2004 - 2007 db4objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.staticfields;
04
05 @@public class Pilot {
06 |     private String name;
07 |
08 |     private PilotCategories category;
09 |
10 @@ public Pilot(String name, PilotCategories category) {
11 |     this.name = name;
12 |     this.category = category;
13 | }
14 |
15 @@ public PilotCategories getCategory() {
16 |     return category;
17 | }
18 |
19 @@ public String getName() {

```

```

20 |     return name;
21 | }
22 |
23 |     public String toString() {
24 |         return name + "/" + category;
25 |     }
26 | }

```

## StaticFieldExample.java: setPilots

```

01 |     private static void setPilots() {
02 |         new File(DB40_FILE_NAME).delete();
03 |         ObjectContainer container = database();
04 |         if (container != null) {
05 |             try {
06 |                 container.set(new Pilot("Michael Schumacher",
07 |                                         PilotCategories.WINNER));
08 |                 container.set(new Pilot("Rubens Barrichello",
09 |                                         PilotCategories.TALENTED));
10 |             } finally {
11 |                 closeDatabase();
12 |             }
13 |         }
14 |     }

```

We can try to save pilots with the default db4o settings:

## StaticFieldExample.java: checkPilots

```

01 |     private static void checkPilots() {
02 |         ObjectContainer container = database();
03 |         if (container != null) {
04 |             try {
05 |                 ObjectSet result = container.query(Pilot.class);
06 |                 for (int x = 0; x < result.size(); x++) {
07 |                     Pilot pilot = (Pilot) result.get(x);
08 |                     if (pilot getCategory() == PilotCategories.WINNER) {
09 |                         System.out.println("Winner pilot: " + pilot);
10 |                     } else if (pilot getCategory() == PilotCategories.TALENTED) {
11 |                         System.out.println("Talented pilot: " + pilot);

```

```

12     } else {
13         System.out.println("Uncategorized pilot: " + pilot);
14     }
15 }
16     } finally {
17     closeDatabase();
18 }
19 }
20 }
```

That does not work however. We will have to explicitly point out, which class's static fields we want to save:

StaticFieldExample.java: configure

```

1 private static void configure() {
2     System.out.println("Saving static fields can be turned on for individual classes.");
3
4     _configuration = Db4o.newConfiguration();
5     _configuration.objectClass(PilotCategories.class)
6         .persistStaticFieldValues();
7 }
```

Try to save and check pilots again - you should see that with this configuration enumeration values are actually correctly bound to their runtime values.

As it was mentioned before, it is important to keep static values in one place and do not allow different objects to modify them. If we try to change static value from the referencing object:

StaticFieldExample.java: updatePilots

```

01 private static void updatePilots() {
02     System.out
03         .println("Updating PilotCategory in pilot reference:");
04     ObjectContainer container = database();
05     if (container != null) {
06         try {
07             ObjectSet result = container.query(Pilot.class);
08             for (int x = 0; x < result.size(); x++) {
09                 Pilot pilot = (Pilot) result.get(x);
10                 if (pilot.getCategory() == PilotCategories.WINNER) {
11                     System.out.println("Winner pilot: " + pilot);
```

```

12 |         PilotCategories pc = pilot.getCategory();
13 |         pc.testChange("WINNER2006");
14 |         container.set(pilot);
15 |     }
16 | }
17 | printCategories(container);
18 | } finally {
19 |     closeDatabase();
20 | }
21 | }
22 | }

```

the value just does not change. You can check it with the `checkPilots` method above.

In order to update static field we will have to do that explicitly:

StaticFieldExample.java: `updatePilotCategories`

```

01 | private static void updatePilotCategories() {
02 |     System.out.println("Updating PilotCategories explicitly:");
03 |     ObjectContainer container = database();
04 |     if (container != null) {
05 |         try {
06 |             ObjectSet result = container.query(PilotCategories.class);
07 |             for (int x = 0; x < result.size(); x++) {
08 |                 PilotCategories pc = (PilotCategories) result.get(x);
09 |                 if (pc == PilotCategories.WINNER) {
10 |                     pc.testChange("WINNER2006");
11 |                     container.set(pc);
12 |                 }
13 |             }
14 |             printCategories(container);
15 |         } finally {
16 |             closeDatabase();
17 |         }
18 |     }
19 |     System.out.println("Change the value back:");
20 |     container = database();
21 |     if (container != null) {
22 |         try {
23 |             ObjectSet result = container.query(PilotCategories.class);

```

```

24 |         for (int x = 0; x < result.size(); x++) {
25 |             PilotCategories pc = (PilotCategories) result.get(x);
26 |             if (pc == PilotCategories.WINNER) {
27 |                 pc.testChange("WINNER");
28 |                 container.set(pc);
29 |             }
30 |         }
31 |         printCategories(container);
32 |     } finally {
33 |         closeDatabase();
34 |     }
35 | }
36 |

```

Please, check the result with the `checkPilots` method. You will see that the reference has changed correctly.

What about deletion? Similar to update we cannot delete static fields from the referenced object, but we can delete them directly from the database:

#### StaticFieldExample.java: addDeleteConfiguration

```

1 | private static void addDeleteConfiguration() {
2 |     if (_configuration != null){
3 |         _configuration.objectClass(Pilot.class).cascadeOnDelete(true);
4 |     }
5 |

```

#### StaticFieldExample.java: deleteTest

```

01 | private static void deleteTest() {
02 |     // use delete configuration
03 |     ObjectContainer container = database();
04 |     if (container != null){
05 |         try {
06 |             System.out.println("Deleting Pilots :");
07 |             ObjectSet result = container.query(Pilot.class);
08 |             for (int x = 0; x < result.size(); x++) {
09 |                 Pilot pilot = (Pilot) result.get(x);
10 |                 container.delete(pilot);
11 |             }

```

```
12 |         printCategories(container);
13 |         System.out.println("Deleting PilotCategories :");
14 |         result = container.query(PilotCategories.class);
15 |         for (int x = 0; x < result.size(); x++) {
16 |             container.delete(result.get(x));
17 |         }
18 |         printCategories(container);
19 |     } finally {
20 |         closeDatabase();
21 |     }
22 |
23 }
```

This revision (14) was last Modified 2007-09-16T11:59:39 by Tetyana.

# Usage of static fields

As an object database db4o can take advantage of programming language specifics such as static modifier. Why and where should it be used?

Usually static fields are used to store enums and constants. Obviously these objects can be stored in application code only, keeping database file smaller and decreasing memory consumption at runtime. But it can be not the best option in the case when the constant (enum) value can be changed in application lifecycle: the references from all the database objects will have to be updated explicitly.

Db4o suggests another approach to keeping constant values. For a class

**Car.java**

```

1 /* Copyright (C) 2004 - 2007 db4objects Inc. http://www.db4o.com */
2
3 package com.db4odoc.staticfields;
4
5 import java.awt.*;
6
7 public class Car {
8     Color color;
9 }
```

the color field can be set to Color enumeration value like that:

**StaticFieldExample.java: setCar**

```

01 private static void setCar() {
02     new File(DB4O_FILE_NAME).delete();
03     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
04     try {
05         Car car = new Car();
06         car.color = Color.GREEN;
07         container.set(car);
08     } finally {
```

```
09 |         container.close();  
10 |     }  
11 | }
```

Now, when ObjectContainer is reopened and the green car is retrieved from the database, the static instances in RAM will be associated with the previously persisted instances using #bind() under the hood. So that the following check is possible:

```
car.color == Color.GREEN
```

This also means that ,if Color.GREEN constant will get another internal value (RGB(0,255,10) instead of RGB(0,255,0) for instance), all the references from the database will be associated with the new value.

Static field values are associated with their persistent identities only once, when an ObjectContainer is opened. After that they are not stored, unless the developer does it deliberately. Objects instantiation from the database does not create any more instances of static values.

Since each static field exists only once in the VM, there are no versioning, locking or multiuser access problems.

This revision (12) was last Modified 2007-09-16T11:52:00 by Tetyana.

# Java enumerations

This topic applies to Java version only

Enumerated types were brought into Java with the JDK 1.5 release. In fact they represent a class with static fields similar to the one reviewed in the [Static fields paragraph](#).

Qualification.java

```

01 /* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.enums;
04
05 public enum Qualification {
06 |
07     WINNER("WINNER"),
08     PROFESSIONAL("PROFESSIONAL"),
09     TRAINEE("TRAINEE");
10
11     private String qualification;
12
13     private Qualification(String qualification) {
14         this.qualification = qualification;
15     }
16
17     public void testChange(String qualification) {
18         this.qualification = qualification;
19     }
20
21     public String toString() {
22         return qualification;
23     }
24 }
```

db4o takes care about storing enumeration objects automatically without any additional settings:

EnumExample.java: setPilots

```

01 private static void setPilots(){
02     new File(DB40_FILE_NAME).delete();
03     ObjectContainer container=Db4o.openFile(DB40_FILE_NAME);
```

```

04 |     try {
05 |         container.set(new Pilot("Michael Schumacher", Qualification.WINNER));
06 |         container.set(new Pilot("Rubens Barrichello", Qualification.PROFESSIONAL));
07 |     } finally {
08 |         container.close();
09 |     }
10 |

```

## EnumExample.java: checkPilots

```

01 |     private static void checkPilots() {
02 |         ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
03 |         try {
04 |             ObjectSet result = container.query(Pilot.class);
05 |             System.out.println("Saved pilots: " + result.size());
06 |             for(int x = 0; x < result.size(); x++) {
07 |                 Pilot pilot = (Pilot)result.get(x);
08 |                 if (pilot.getQualification() == Qualification.WINNER) {
09 |                     System.out.println("Winner pilot: " + pilot);
10 |                 } else if (pilot.getQualification() == Qualification.PROFESSIONAL) {
11 |                     System.out.println("Professional pilot: " + pilot);
12 |                 } else {
13 |                     System.out.println("Uncategorized pilot: " + pilot);
14 |                 }
15 |             }
16 |         } finally {
17 |             container.close();
18 |         }
19 |

```

Another specific feature of enums in db4o: deletion is not possible:

## EnumExample.java: deletePilots

```

01 |     private static void deletePilots() {
02 |         System.out.println("Qualification enum before delete Pilots");
03 |         printQualification();
04 |         Configuration configuration = Db4o.newConfiguration();
05 |         configuration.objectClass(Pilot.class).objectField("qualification").cascadeOnDelete(true);

```

```

06 |     ObjectContainer container=Db4o.openFile(configuration, DB40_FILE_NAME);
07 |
08 |     try {
09 |         ObjectSet result = container.query(Pilot.class);
10 |         for(int x = 0; x < result.size(); x++) {
11 |             Pilot pilot = (Pilot)result.get(x);
12 |             container.delete(pilot);
13 |         }
14 |     } finally {
15 |         container.close();
16 |     }
17 |     System.out.println("Qualification enum after delete Pilots");
18 |     printQualification();
19 |

```

## EnumExample.java: checkPilots

```

01 |     private static void checkPilots(){
02 |         ObjectContainer container=Db4o.openFile(DB40_FILE_NAME);
03 |         try {
04 |             ObjectSet result = container.query(Pilot.class);
05 |             System.out.println("Saved pilots: " + result.size());
06 |             for(int x = 0; x < result.size(); x++) {
07 |                 Pilot pilot = (Pilot)result.get(x);
08 |                 if (pilot.getQualification() == Qualification.WINNER) {
09 |                     System.out.println("Winner pilot: " + pilot);
10 |                 } else if (pilot.getQualification() == Qualification.PROFESSIONAL) {
11 |                     System.out.println("Professional pilot: " + pilot);
12 |                 } else {
13 |                     System.out.println("Uncategorized pilot: " + pilot);
14 |                 }
15 |             }
16 |         } finally {
17 |             container.close();
18 |         }
19 |

```

Deletion of references does not automatically delete the enum. Even explicit deletion does not work:

## EnumExample.java: deleteQualification

```

01 | private static void deleteQualification() {
02 |     System.out.println("Explicit delete of Qualification enum");
03 |     Configuration configuration = Db4o.newConfiguration();
04 |     configuration.objectClass(Qualification.class).cascadeOnDelete(true);
05 |     ObjectContainer container=Db4o.openFile(configuration, DB40_FILE_NAME);
06 |     try {
07 |         ObjectSet result = container.query(Qualification.class);
08 |         for(int x = 0; x < result.size(); x++) {
09 |             Qualification pq = (Qualification)result.get(x);
10 |             container.delete(pq);
11 |         }
12 |     } finally {
13 |         container.close();
14 |     }
15 |     printQualification();
16 |

```

Enum update works in the same way as for normal static objects - updated enum should be explicitly saved to database (#set (enum)).

#### EnumExample.java: updateQualification

```

01 | private static void updateQualification() {
02 |     System.out.println("Updating WINNER qualification constant");
03 |     ObjectContainer container=Db4o.openFile(DB40_FILE_NAME);
04 |     try {
05 |         Query query = container.query();
06 |         query.constrain(Qualification.class);
07 |         query.descend("qualification").constrain("WINNER");
08 |         ObjectSet result = query.execute();
09 |         for(int x = 0; x < result.size(); x++) {
10 |             Qualification qualification = (Qualification)result.get(x);
11 |             qualification.testChange("WINNER2006");
12 |             container.set(qualification);
13 |         }
14 |     } finally {
15 |         container.close();
16 |     }
17 |     printQualification();
18 |

```

You can use either build-in Java enums or write your own. Db4o will take care of keeping object references unique and database file as small as possible.

This revision (10) was last Modified 2006-12-29T17:20:50 by Tetyana.

# Delegates And Events

This topic applies to .NET version only

This revision (8) was last Modified 2007-05-07T09:36:25 by Tetyana.

# Final Fields

This topic applies to java version only

This topic will give you an overview of some specifics concerning final fields usage in persistent objects.

More Reading:

- [Final Fields Specifics](#)
- [Possible Solutions](#)

This revision (1) was last Modified 2007-01-22T19:07:38 by Tetyana.

# Final Fields Specifics

This topic applies to java version only

Db4o uses reflection to store and retrieve objects from the database file. In the case of final fields db4o needs a successful call to java.lang.Field#setAccessible to allow write access to those fields. Unfortunately different Java versions produce different results in this case. To be more specific:

- In (Sun) JDK 1.1-1.2 java.lang.Field#setAccessible call will be successful for the fields with the final modifier.
- This behavior was changed for JDK1.3-1.4 as the API documentation for java.lang.Field#set() made a quite clear distinction between 'Java language access control' (visibility modifiers, affected by setAccessible()) and final fields (not affected by setAccessible()). For more information refer to [java bug 4250960](#).
- The behavior of java.lang.Field#setAccessible method was changed again for JDK5 and JDK6. The access to final fields was made manageable by setAccessible() call to accommodate for the extended semantics of the final modifier for the revised Java memory model. The API documentation of java.lang.Field#set() was changed accordingly. See [java bug 5044412](#).

You can use the following example code to check final fields behavior with different java versions:

TestFinal.java

```

01 /* Copyright (C) 2007 db4o objects Inc. http://www.db4o.com */
02 package com.db4odoc.finalfields;
03 import java.io.File;
04
05 import com.db4o.Db4o;
06 import com.db4o.ObjectContainer;
07 import com.db4o.ObjectSet;
08
09 public class TestFinal
10 {
11     private static final String DB4O_FILE_NAME = "reference.db4o";
12     // non-final fields
13     public int _i;
14     public String _s;
15     // final fields storing the same values as above
16     public final int _final_i;
17     public final String _final_s;
18
19     public static void main(String[] args)
20     {
21         new File(DB4O_FILE_NAME).delete();
22         ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
23         try {

```

```

24 |         TestFinal test = new TestFinal(1, "test");
25 |         container.set(test);
26 |         System.out.println("Added: " + test);
27 |     } finally {
28 |         // Close does implicit commit and refreshes the reference cache
29 |         container.close();
30 |     }
31 |     container = Db4o.openFile(DB4O_FILE_NAME);
32 |     try {
33 |         ObjectSet result = container.get(null);
34 |         listResult(result);
35 |     } finally {
36 |         container.close();
37 |     }
38 | }
39 | // end main
40 |
41 | public TestFinal(int i, String s)
42 | {
43 |     // initialize final and non-final fields with the same values
44 |     _i = i;
45 |     _s = s;
46 |     _final_i = i;
47 |     _final_s = s;
48 | }
49 | // end TestFinal
50 |
51 | public String toString()
52 | {
53 |     return "Int - " + _i + "; String - " + _s + "; FINAL Int - " + _final_i + "; FINAL
String - " + _final_s;
54 | }
55 | // end toString
56 |
57 | private static void listResult(ObjectSet result)
58 | {
59 |     while(result.hasNext()) {
60 |         System.out.println(result.next());
61 |     }
62 | }

```

```
63 | // end listResult  
64 }
```

If you are using Eclipse it is easy to switch between java versions - you can switch to the versions lower than the one installed on your computer without having to install them all. For example if you are using JDK6 you can easily test your project on JDK1.1 - 1.4 and JDK5. Just go to the project properties, select "Java Build Path" on the left panel and "Libraries" tab on the right panel. Remove the system library currently used. Select "Add library->JRE System Library"; on the next screen check the "Execution Environment" radio button and select the desired environment from the list.

Don't forget to use the appropriate db4o version for the selected java environment version. See [db4o on Java Platforms](#) for more information.

This revision (5) was last Modified 2007-05-07T09:39:10 by Tetyana.

# Possible Solutions

This topic applies to java version only

Of course, if you only use JDK5 or 6 there are no worries about the final fields at all. But if you do not want to stick to the definite java version and need to have the flexibility of switching to different java versions you currently have 2 solutions:

- avoid using the final modifier in the persistent objects;
- use translator.

An example of the final fields translator can look like this:

FinalFieldTranslator.java

```
01 /* Copyright (C) 2007 db4objects Inc. http://www.db4o.com */
02 package com.db4odoc.finalfields;
03
04 import com.db4o.*;
05 import com.db4o.config.*;
06
07 // Translator allowing to store final fields on any Java version
08 public class FinalFieldTranslator implements ObjectConstructor {
09 |
10 |     public Object onStore(ObjectContainer container,
11 |             Object applicationObject) {
12 |         System.out.println("onStore for " + applicationObject);
13 |         TestFinal notStorable = (TestFinal) applicationObject;
14 |         // final fields values are stored to an array of objects
15 |         return new Object[] { new Integer(notStorable._final_i),
16 |             notStorable._final_s };
17 |     }
18 |
19 |     public Object onInstantiate(ObjectContainer container,
20 |             Object storedObject) {
```

```

21 |     System.out.println("onInstantiate for " + storedObject);
22 |     Object[] raw = (Object[]) storedObject;
23 |     // final fields values are restored from the array of objects
24 |     int i = ((Integer) raw[0]).intValue();
25 |     String s = (String) raw[1];
26 |     return new TestFinal(i, s);
27 |
28 |
29 |     public void onActivate(ObjectContainer container,
30 |         Object applicationObject, Object storedObject) {
31 |         System.out.println("onActivate for " + applicationObject
32 |             + " / " + storedObject);
33 |
34 |
35 |         public Class storedClass() {
36 |             return Object[].class;
37 |         }
38 |

```

The following call should be issued before opening the ObjectContainer to connect the translator to the TestFinal class:

```
Db4o.configure().objectClass(TestFinal.class).translate(new
FinalFieldTranslator());
```

This revision (3) was last Modified 2007-01-22T19:20:48 by Tetyana.

# TypeHandlers

One of the most important and convenient things that db4o provides is the ability to store any object just as it is: no interfaces to be implemented, no custom fields, no attributes/annotations - nothing, just a plain object. However, it is not as simple as it may seem - objects are getting more and more complex and sometimes the generic solution is not good enough for specific objects.

This problem was recognized by db4o team long ago, and various solutions were provided to customize the way an object is stored: configuration #readAs method, [Translators](#), transient [fields in Java](#) and [.NET](#) and [classes](#), custom marshallers etc. However all these means were rather fixing the symptoms but not the disease itself. And the fact is that there is no single generic way to store just any available or future object in the best possible way. But luckily we don't even need it - all we need is a simple way to write a specific persistence solution for any custom object, and now db4o provides this way through a pluggable TypeHandler4/ITypeHandler4 interface:

Java:

```
configuration.registerTypeHandler(TypeHandlerPredicate, TypeHandler4);
```

In the method above TypeHandler4 interface provides methods that define how an object is converted to a low-level byte-array and back and how it behaves in a query:

TypeHandler4.java

```
01  /* Copyright (C) 2004 - 2007 db4objects Inc. http://www.db4o.com */  
02  
03 package com.db4o.internal;  
04  
05 import com.db4o.ext.*;  
06 import com.db4o.internal.fieldhandlers.*;  
07 import com.db4o.marshall.*;  
08  
09  
10  **  
11 | * @exclude  
12 | */  
13  public interface TypeHandler4 extends Comparable, FieldHandler {  
14 |
```

## TypeHandlers

```
15 |     void delete(DeleteContext context) throws Db4oIOException;
16 |
17 |     void defragment(DefragmentContext context);
18 |
19 |     Object read(ReadContext context);
20 |
21 |     void write(WriteContext context, Object obj);
22 |
23 |}
```

## Comparable4.java

```
01 /* Copyright (C) 2004 db4o objects Inc. http://www.db4o.com */
02
03 package com.db4o.internal;
04
05 import com.db4o.foundation.*;
06
07
08 /**
09 * @exclude
10 */
11 public interface Comparable4 {
12
13     PreparedComparison prepareComparison(Object obj);
14
15 }
```

TypeHandlerPredicate provides a #match method, which returns true for objects that should be handled with the specified TypeHandler.

Type handler functionality is best explained on a [working example](#).

Usecases and other benefits of the pluggable typehandler interface are reviewed [here](#).

This revision (6) was last Modified 2008-02-04T21:44:30 by dlouwers.

# Custom Typehandler Example

For a custom typehandler example we will try to write a very simple typehandler for StringBuffer(java) and StringBuilder(.NET). These classes are basically just another representation of String, so we can look at the StringHandler implementation in db4o source.

To keep it simple we will skip information required for indexing - please look at IndexableTypeHandler in db4o sources to get more information on how to handle indexes.

The first thing should be #write method, which determines how the object is persisted:

StringBufferHandler.java: write

```

1 public void write(WriteContext context, Object obj) {
2     String str = ((StringBuffer) obj).toString();
3     WriteBuffer buffer = context;
4     buffer.writeInt(str.length());
5     writeToBuffer(buffer, str);
6 }
```

StringBufferHandler.java: writeToBuffer

```

1 private static void writeToBuffer(WriteBuffer buffer, String str){
2     final int length = str.length();
3     char[] chars = new char[length];
4     str.getChars(0, length, chars, 0);
5     for (int i = 0; i < length; i++) {
6         buffer.writeByte((byte) (chars[i] & 0xff));
7         buffer.writeByte((byte) (chars[i] >> 8));
8     }
9 }
```

As you can see from the code above, there are 3 steps:

1. Get the buffer from WriteContext/I WriteContext
2. Write the length of the StringBuffer/StringBuilder
3. Transfer the object to char array and write them in Unicode

Next step is to read the same from the buffer. It is just opposite to the write method:

StringBufferHandler.java: read

```

1 public Object read(ReadContext context) {
2     ReadBuffer buffer = context;
3     String str = "";
```

```

4|     int length = buffer.readInt();
5|     if (length > 0) {
6|         str = readBuffer(buffer, length);
7|     }
8|     return new StringBuffer(str);
9| }
```

## StringBufferHandler.java: readBuffer

```

1| private static String readBuffer(ReadBuffer buffer, int length){
2|     char[] chars = new char[length];
3|     for(int ii = 0; ii < length; ii++){
4|         chars[ii] = (char)((buffer.readByte() & 0xff) | ((buffer.readByte() & 0xff) << 8));
5|     }
6|     return new String(chars, 0, length);
7| }
```

Delete is simple - we just reposition the buffer offset to the end of the slot:

## StringBufferHandler.java: delete

```

1| public void delete(DeleteContext context) {
2|     context.readSlot();
3| }
```

Try to experiment with the #delete method by implementing cascade on delete. Use FirstClassObjectHandler as an example.

We are done with the read/write operations. But as you remember, in order to read an object, we must find it through a query, and that's where we will need a #compare method (well, you do not need it if your query does not contain any comparison criteria, but this is normally not the case):

## StringBufferHandler.java: prepareComparison

```

1| public PreparedComparison prepareComparison(Context ctx, final Object obj) {
2|     return new PreparedComparison() {
3|         public int compareTo(Object target) {
4|             return compare((StringBuffer)obj, (StringBuffer)target);
5|         }
6|     };
7| }
```

## StringBufferHandler.java: compare

```

01 private final int compare(StringBuffer a_compare, StringBuffer a_with) {
02     if (a_compare == null) {
03         if (a_with == null) {
04             return 0;
05         }
06         return -1;
07     }
08     if (a_with == null) {
09         return 1;
10     }
11     char c_compare[] = new char[a_compare.length()];
12     a_compare.getChars(0, a_compare.length() - 1, c_compare, 0);
13     char c_with[] = new char[a_with.length()];
14     a_with.getChars(0, a_with.length() - 1, c_with, 0);
15
16     return compareChars(c_compare, c_with);
17 }

```

## StringBufferHandler.java: compareChars

```

1 private static final int compareChars(char[] compare, char[] with) {
2     int min = compare.length < with.length ? compare.length : with.length;
3     for (int i = 0; i < min; i++) {
4         if (compare[i] != with[i]) {
5             return compare[i] - with[i];
6         }
7     }
8     return compare.length - with.length;
9 }

```

The last method left: #defragment. This one only moves the offset to the beginning of the object data, i.e. skips Id and size information (to be compatible to older versions):

## StringBufferHandler.java: defragment

```

1 public void defragment(DefragmentContext context) {
2     // To stay compatible with the old marshaller family
3     // In the marshaller family 0 number 8 represented
4     // length required to store ID and object length information
5     context.incrementOffset(8);
6 }

```

This Typehandler implementation can be tested with a class below. Please, pay special attention to #configure method, which adds StringBufferHandler/StringBuilderHandler to the database configuration:

## TypehandlerExample.java

```

001 package com.db4odoc.typehandler;
002
003 import java.io.File;
004 import java.io.IOException;
005
006 import com.db4o.Db4o;
007 import com.db4o.ObjectContainer;
008 import com.db4o.ObjectSet;
009 import com.db4o.config.Configuration;
010 import com.db4o.defragment.Defragment;
011 import com.db4o.ext.DatabaseFileLockedException;
012 import com.db4o.query.Query;
013 import com.db4o.reflect.ReflectClass;
014 import com.db4o.reflect.generic.GenericReflector;
015 import com.db4o.reflect.jdk.JdkReflector;
016 import com.db4o.typehandlers.TypeHandlerPredicate;
017
018 public class TypehandlerExample {
019
020     private final static String DB4O_FILE_NAME = "reference.db4o";
021     private static ObjectContainer _container = null;
022
023
024     public static void main(String[] args) throws IOException {
025         testReadWriteDelete();
026         //testDefrag();
027         testCompare();
028     }
029     // end main
030
031     private static Configuration configure() {
032         Configuration configuration = Db4o.newConfiguration();
033         // add a custom typehandler support
034
035         TypeHandlerPredicate predicate = new TypeHandlerPredicate() {
036             public boolean match(ReflectClass classReflector, int version) {
037                 GenericReflector reflector = new GenericReflector(

```

```

038 |                     null, new JdkReflector(Thread.currentThread().getContextClassLoader()));
039 |             ReflectClass clazz = reflector.forName(StringBuffer.class.getName());
040 |             boolean res = clazz.equals(classReflector);
041 |             return res;
042 |         }
043 |     };
044 |
045 |     configuration.registerTypeHandler(predicate, new StringBufferHandler());
046 |     return configuration;
047 | }
048 | // end configure
049 |
050 |
051 | private static void testReadWriteDelete(){
052 |     storeCar();
053 |     // Does it still work after close?
054 |     retrieveCar();
055 |     // Does deletion work?
056 |     deleteCar();
057 |     retrieveCar();
058 | }
059 | // end testReadWriteDelete
060 |
061 | private static void retrieveCar() {
062 |     ObjectContainer container = database(configure());
063 |     if (container != null){
064 |         try {
065 |             ObjectSet result = container.query(Car.class);
066 |             Car car = null;
067 |             if (result.hasNext()){
068 |                 car = (Car)result.next();
069 |             }
070 |             System.out.println("Retrieved: " + car);
071 |         } finally {
072 |             closeDatabase();
073 |         }
074 |     }
075 | }
076 | // end retrieveCar
077 |
078 | private static void deleteCar() {

```

```

079 |     ObjectContainer container = database(configure());
080 |     if (container != null){
081 |         try {
082 |             ObjectSet result = container.query(Car.class);
083 |             Car car = null;
084 |             if (result.hasNext()){
085 |                 car = (Car)result.next();
086 |             }
087 |             container.delete(car);
088 |             System.out.println("Deleted: " + car);
089 |         } finally {
090 |             closeDatabase();
091 |         }
092 |     }
093 | }
094 | // end deleteCar
095 |
096 |     private static void storeCar() {
097 |         new File(DB40_FILE_NAME).delete();
098 |         ObjectContainer container = database(configure());
099 |         if (container != null){
100 |             try {
101 |                 Car car = new Car("BMW");
102 |                 container.store(car);
103 |                 car = (Car)container.query(Car.class).next();
104 |                 System.out.println("Stored: " + car);
105 |
106 |             } finally {
107 |                 closeDatabase();
108 |             }
109 |         }
110 |     }
111 | // end storeCar
112 |
113 |     private static void testCompare() {
114 |         new File(DB40_FILE_NAME).delete();
115 |         ObjectContainer container = database(configure());
116 |         if (container != null){
117 |             try {
118 |                 Car car = new Car("BMW");
119 |                 container.store(car);

```

```

120 |         car = new Car("Ferrari");
121 |         container.store(car);
122 |         car = new Car("Mercedes");
123 |         container.store(car);
124 |         Query query = container.query();
125 |         query.constrain(Car.class);
126 |         query.descend("model").orderAscending();
127 |         Object result = query.execute();
128 |         listResult(result);
129 |
130 } finally {
131     closeDatabase();
132 }
133 }
134
135 // end testCompare
136
137 public static void testDefrag() throws IOException{
138     new File(DB40_FILE_NAME + ".backup").delete();
139     storeCar();
140     Defragment.defrag(DB40_FILE_NAME);
141     retrieveCar();
142 }
143 // end testDefrag
144
145 private static ObjectContainer database(Configuration configuration) {
146     if (_container == null) {
147         try {
148             _container = Db4o.openFile(configuration, DB40_FILE_NAME);
149         } catch (DatabaseFileLockedException ex) {
150             System.out.println(ex.getMessage());
151         }
152     }
153     return _container;
154 }
155 // end database
156
157 private static void closeDatabase() {
158     if (_container != null) {
159         _container.close();
160         _container = null;

```

```
161 |      }
162 |  }
163 |  // end closeDatabase
164 |
165 |
166 |  private static void listResultSet(ObjectSet result) {
167 |      System.out.println(result.size());
168 |      while(result.hasNext()) {
169 |          System.out.println(result.next());
170 |      }
171 |  }
172 |  // end listResultSet
173 |
174 |}
```

This revision (2) was last Modified 2008-02-03T11:32:35 by Tetyana.

# Pluggable Typehandler Benefits

As the name suggests Pluggable Typehandler allows anybody to write custom typehandlers, and thus control the way the class objects are stored to the database and retrieved in a query. Why would you do this? There can be various reasons:

- You know a more performant way to convert objects to byte array or to compare them.
- You need to store only part of the object's information and want to skip unneeded fields to keep the database smaller. You can also do the same using [Transient](#) marker, but this is only possible for classes with available code. Using custom typehandler you can configure partial storage for any third-party class.
- You need to keep information that will allow you to restore fields that cannot be stored as is, for example: references to environmental variables (like time zone), proxy objects or variables of temporary state (like current memory usage). Previously, this job was done by [Translators](#), but certainly custom Typehandler gives you more control and unifies the approach.
- You want to customize the way Strings are stored (special encodings).
- You need to perform a complex refactoring on-the-fly (use typehandler versioning)
- You want to cipher each object before putting it into the database
- You want to implement cascade on delete through the typehandler

Other not so common and more difficult in realization behaviours that can be realized with the new Typehandler:

- Custom handling of platform-specific generic collections
- Fast collection handlers that operate on a lower level and scale for large collections
- Customary indexes
- Versioning of typehandlers (can be used for refactoring and db4o version upgrades)

Of course, writing typehandlers is not totally simple, but once you understand how to do that - you will also gain a much deeper understanding of db4o itself. You can start with a [simple example](#) provided in this documentation and continue by looking into existing db4o typehandler implementations: StringHandler, VariableLengthTypeHandler, IndexableTypeHandler etc.

This revision (1) was last Modified 2008-02-03T09:36:04 by Tetyana.

# Using Annotations

This topic applies to Java version only

JDK1.5 platform introduced new metadata feature called [annotations](#). Annotations allow programmers to decorate Java code with their attributes, which can be used afterwards for automatic code generation, documentation, security checking etc. Annotations do not directly affect program semantics, but they do affect the way programs are treated by tools and libraries, which can in turn affect the semantics of the running program.

Annotations can make your code cleaner, protecting you from common errors (using deprecated API, typos in overriding methods) and taking part of you work.

You can use annotations to affect db4o behavior. At present we provide only one annotation:

`@Indexed`

This annotation can be applied to class fields

Car.java

```

01 /* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.annotations;
04
05 import com.db4o.config.annotations.Indexed;
06
07
08 public class Car {
09     @Indexed
10     private String model;
11     private int year;
12 }
```

and its functionality is equivalent to the db4o configurations setting:

```
Db4o.configure().objectClass(clazz).objectField("fieldName").indexed(true)
```

This revision (6) was last Modified 2007-05-07T09:56:15 by Tetyana.

# Enhancement Tools

Enhancement tools provide a convenient framework for application (jar, dll, exe) or classes modification to support db4o-specific functionality. Enhancement tools can work on a ready application or library and apply the improvements at load or build time.

The tools functionality is provided through bytecode Instrumentation (BI). Bytecode instrumentation is a process of inserting of special, usually short, sequences of bytecode at designated points of Java or .NET class. BI is typically used for profiling or monitoring, however the range of use of bytecode instrumentation is not limited by this tasks: BI can be applied anywhere where a specific functionality should be plugged into the ready built classes.

db4o Enhancement Tools currently have 2 usecases for bytecode instrumentation:

- [Native Query Optimization](#);
- [Transparent Activation](#)
- [Transparent Persistence](#)

In NQ optimization case bytecode instrumentation is used as a more performant alternative to a run-time optimization. When an NQ is optimized the user and compiler-friendly syntax of NQ predicate is replaced with a query-processor-friendly code (bytecode in the case of BI). Obviously, optimization process can take some time, therefore it can be a good choice to use pre-instrumented classes, then to let the optimization be executed each time it is required by application.

In TA case, classes are required to implement Activatable interface to support transparent activation. In many cases you won't want to "pollute" your proprietary classes with some additional interface, or even won't be able to do so if you use a third party classes library. That's where BI comes handy: Activatable interface will be implemented on your existing classes by applying bytecode instrumentation. Another advantage of this approach - you can still work on your "clean" classes, just do not forget to run BI afterwards.

Bytecode instrumentation in Java can be run at build time (also known as static instrumentation). In this case a special (build) script calls BI on the classes before packaging them to jar, or on the jar itself (classes are extracted, instrumented and jarred again). This is the fastest solution as no time is spent on bytecode instrumentation at runtime.

Another method is to use BI at load time. In this case instrumenting information is inserted into the classes by a specific instrumenting classloader just before they are loaded into the VM.

The following topics discuss BI implementation for db4o needs in more detail and explain the tools and API that should be used for BI tasks.

- [Enhancement For Java](#)

- [Enhancement For .NET](#)

This revision (7) was last Modified 2008-01-20T09:25:47 by Tetyana.

# Enhancement For Java

db4o enhancement framework relies on the following jars:

bloat-1.0	Third-party bytecode instrumentation library
db4o-x.x-instrumentation	Instrumentation library on top of bloat
db4o-x.x-tools	Enhancement and other utilities

In addition

- for TA /TP instrumentation enhancement db4o-x.x-taj.jar should be used (contains TA /TP instrumentation classes);
- for NQ optimization db4o-x.x-nqopt.jar is used (provides instrumentation functionality for NQ).

The basic steps required to enhance classes are:

1. Create ClassFilter instance to select the classes for enhancement. ClassFilter is an interface in db4oinstrumentation project and is implemented by several classes, like AcceptAllClassesFilter, ByNameClassFilter and others (see ClassFilter hierarchy for a list of all implementations).
2. Create BloatClassEdit array of classes capable of editing class bytecode. BloatClassEdit is an interface in db4oinstrumentation project. Among its implementations are TranslateNQToSODAEedit (implements NQ optimization) and InjectTransparentActivationEdit (injects TA/TP awareness). Filter can be used in some of the edit classes (InjectTransparentActivationEdit).
3. For load-time instrumentation the edit classes created above are passed to Db4oInstrumentationLauncher together with the application entry point class. Db4oInstrumentationLauncher is a public class in db4oinstrumentation project, which creates a special instrumenting classloader and uses it to load the application's main class.
4. For build time instrumentation Db4oFileEnhancerAntTask is used to create an enhancer task in Ant, which must call the class edit classes inside. Db4oFileEnhancerAntTask is a class extending Ant task in db4oinstrumentation project. It loads and instruments the classes using class edits supplied as parameters to the enhancer task and copies the resulted classes to the output directory. It can also work on Jars instead of classes.

The examples below shows how enhancer works at load and build time:

- [TA Enhancement at Loading Time](#)
- [TA Enhancement at Build Time](#)
- [TP Enhancement at Build Time](#)
- [NQ Enhancement at Loading Time](#)

- [NQ Enhancement at Build Time](#)
- [Complex Example](#)

This revision (9) was last Modified 2008-01-20T13:43:59 by Tetyana.

# Complex Example

This topic applies to Java version only

The following example shows some advanced enhancement features:

- Instrumentation of jarred items
- Annotation based class filter
- TA and NQ enhancement in one go
- Load-time TA enhancement for collections

More Reading:

- [Model Classes](#)
- [Load Time Enhancement](#)
- [Build Time Enhancement](#)

This revision (1) was last Modified 2007-11-07T06:40:13 by Tetyana.

# Model Classes

This topic applies to Java version only

Two simple classes Pilot and Id are pre-packed in a Pilot.jar:

Pilot.java

```

01  /* Copyright (C) 2007 db4o objects Inc. http://www.db4o.com */
02  package enhancement.model;
03
04  public class Pilot {
05      String _name;
06      Id _id;
07
08      public Pilot(String name, Id id) {
09          _name = name;
10          _id = id;
11      }
12
13      public String get_name() {
14          return _name;
15      }
16      public void set_name(String name) {
17          this._name = name;
18      }
19      public Id get_id() {
20          return _id;
21      }
22      public void set_id(Id id) {
23          this._id = id;
24      }
25
26      public String toString() {
27          return _name + ": " + _id;
28      }
29 }
```

Id.java

```

01 /* Copyright (C) 2007 db4o objects Inc. http://www.db4o.com */
02 package enhancement.model;
03
04 public class Id {
05 |   String _id;
06 |
07   public Id(String id){
08 |    _id = id;
09 |  }
10 |
11   public String toString(){
12 |    return _id;
13 |  }
14 }

```

Linked collection of Car objects shows collection enhancement:

Car.java

```

01 package enhancement.model;
02
03 import tacustom.*;
04
05 @Db4oPersistent
06 public class Car {
07 |
08 |   private String _model = null;
09 |   Pilot _pilot;
10 |
11   public Car(String content, Pilot pilot) {
12 |    _model = content;
13 |    _pilot = pilot;
14 |  }
15 |
16   public String content() {
17 |    return _model;
18 |  }
19 |
20   public void content(String content) {

```

```

21 |     _model = content;
22 | }
23 |
24 | @Override
25 | public String toString() {
26 |     return _model + "/" + _pilot;
27 | }
28 |
29 | public String getModel() {
30 |     return _model;
31 | }
32 |
33 | public Pilot getPilot() {
34 |     return _pilot;
35 | }
36 }

```

## MaintenanceQueue.java

```

001 /* Copyright (C) 2007 db4objects Inc. http://www.db4o.com */
002
003 package enhancement.model;
004
005 import java.util.*;
006
007 import tacustom.*;
008
009 @Db4oPersistent
010
011 public class MaintenanceQueue<Item> {
012 |
013     public MaintenanceQueue<Item> _next;
014 |
015     private Item _value;
016 |
017     public MaintenanceQueue(Item value) {
018         _value = value;
019     }
020 |

```

```
021  public static MaintenanceQueue<Integer> newList(int depth) {  
022      if (depth == 0) {  
023          return null;  
024      }  
025      MaintenanceQueue<Integer> head = new MaintenanceQueue<Integer>(depth);  
026      head._next = newList(depth - 1);  
027      return head;  
028  }  
029  
030  /**  
031   * Overrides this method to assert that <code>other</code> is only  
032   * activated with depth 1.  
033   */  
034  @SuppressWarnings("unchecked")  
035  public boolean equals(Object other) {  
036      return ((MaintenanceQueue<Item>) other)._next == null;  
037  }  
038  
039  public boolean hasNext() {  
040      return _next != null;  
041  }  
042  
043  public MaintenanceQueue<Item> next() {  
044      return _next;  
045  }  
046  
047  public int size() {  
048      if (_next == null) {  
049          return 1;  
050      }  
051      return _next.size() + 1;  
052  }  
053  
054  public Item get(int idx) {  
055      if (idx == 0) {  
056          return value();  
057      }  
058      return _next.get(idx - 1);  
059  }
```

```
060 |
061 [+] public Item value() {
062 |     return _value;
063 | }
064 |
065 [+] public void add(Item item) {
066 [+]     if(_next != null) {
067 |         _next.add(item);
068 |     }
069 [+]     else {
070 |         _next = new MaintenanceQueue<Item>(item);
071 |     }
072 | }
073 |
074 [+] public Iterator<Item> iterator() {
075 |     return new LinkedListIterator<Item>(this);
076 | }
077 |
078 [+] public String toString() {
079 |     return "LinkedList: " + _value;
080 | }
081 |
082 [+] public static <Item> MaintenanceQueue<Item> add(MaintenanceQueue<Item> list, Item
item) {
083 [+]     if(list == null) {
084 |         return new MaintenanceQueue<Item>(item);
085 |     }
086 |     list.add(item);
087 |     return list;
088 | }
089 |
090 [+] private final static class LinkedListIterator<Item> implements Iterator<Item> {
091 |     private MaintenanceQueue<Item> _current;
092 |
093 [+]     public LinkedListIterator(MaintenanceQueue<Item> list) {
094 |         _current = list;
095 |     }
096 |
097 [+]     public boolean hasNext() {
```

```
098 |         return _current != null;
099 |     }
100 |
101 |     public Item next() {
102 |         Item item = _current.value();
103 |         _current = _current.next();
104 |         return item;
105 |     }
106 |
107 |     public void remove() {
108 |         throw new UnsupportedOperationException();
109 |     }
110 | }
111 |
112 |     public static <Item> Iterator<Item> iterator(MaintenanceQueue<Item> list) {
113 |         return (list == null ? new LinkedListIterator<Item>(null) : list.iterator());
114 |     }
115 |
116 }
```

This revision (4) was last Modified 2007-11-07T06:52:13 by Tetyana.

# Load Time Enhancement

This topic applies to Java version only

The following code is used to store and retrieve MaintenanceQueue objects containing references to Car, Pilot and Id objects:

EnhancerMain.java: main

```

01  public static void main(String[] args) {
02 |     new File(DB40_FILE_NAME).delete();
03 |     ObjectContainer db = Db4o.openFile(configuration(), DB40_FILE_NAME);
04 |     MaintenanceQueue<Car> queue = null;
05 |     for(int number = 0; number < DEPTH; number++) {
06 |         queue = MaintenanceQueue.add(queue, new Car("Car" + number,
07 |             new Pilot("Pilot #" + number, new Id("110021" + number))));
08 |     }
09 |     db.set(queue);
10 |     db.close();
11 |
12 |     db = Db4o.openFile(configuration(), DB40_FILE_NAME);
13 |     EventRegistry registry = EventRegistryFactory.createObjectContainer(db);
14 |     registry.activated().addListener(new EventListener4() {
15 |         public void onEvent(Event4 event, EventArgs args) {
16 |             ObjectEventArgs objArgs = (ObjectEventArgs) args;
17 |             System.out.println("ACTIVATED: " + objArgs.object());
18 |         }
19 |     });
20 |     ((ObjectContainerBase)db).getNativeQueryHandler().addListener(new
Db4oQueryExecutionListener() {
21 |         public void notifyQueryExecuted(NQOptimizationInfo info) {
22 |             System.out.println(info);
23 |         }
24 |     });
25 |
26 |     List<MaintenanceQueue<Car>> result = db.query(new Predicate<MaintenanceQueue<Car>>() {
27 |         @Override
28 |         public boolean match(MaintenanceQueue<Car> queue) {
29 |             return queue.value().getModel().equals("Car0");
30 |         }
31 |     });
32 |     System.out.println(result.size());

```

```

33 |     //for (Iterator<MaintenanceQueue<Car>> i = result.iterator(); i.hasNext();) {
34 |         MaintenanceQueue<Car> carQueue = result.get(0);
35 |         Car car = carQueue.value();
36 |         System.out.println(car);
37 |
38 |         Pilot pilot = car.getPilot();
39 |         System.out.println(pilot);
40 |         while (carQueue.hasNext()) {
41 |             carQueue = carQueue.next();
42 |             car = carQueue.value();
43 |             System.out.println(car);
44 |
45 |             pilot = car.getPilot();
46 |             System.out.println(pilot);
47 |         }
48 |     //}
49 |     db.close();
50 |     new File(DB40_FILE_NAME).delete();
51 | }

```

## EnhancerMain.java: configuration

```

1 | private static Configuration configuration() {
2 |     Configuration config = Db4o.newConfiguration();
3 |     config.add(new TransparentActivationSupport());
4 |     // NOTE: required for load time instrumentation!
5 |     config.reflectWith(new JdkReflector(EnhancerMain.class.getClassLoader()));
6 |     return config;
7 | }

```

Please, run this method to see that in TA mode all the objects are fully activated immediately. Also NQ info reports that the queries run dynamically optimized.

In order to use TA advantages (lazy activation), we launch the application through an instrumenting classloader. The following configuration options are available:

- A *ClassFilter* specifies which classes should be instrumented. In the example, we are using a filter that will only accept classes whose fully qualified name starts with a given prefix. The instrumentation API already comes with a variety of other filter implementations, and it's easy to create custom filters.
- A sequence of *ClassEdits*. A ClassEdit is a single instrumentation step. In the example, we are applying two steps: First, we preoptimize all Native Query Predicates, then we instrument for Transparent Activation. Note that the order of steps is significant: Switching the order would leave the generated NQ optimization code unaware of TA. The db4otools package provides a convenience launcher with a hardwired sequence for combined NQ/TA instrumentation.
- The *classpath* for the instrumented classes, represented by a sequence of URLs. This must contain all classes "reachable" from the classes to be instrumented - the easiest way probably is to provide the full application class path here. The classes to be instrumented need not be listed here, they are implicitly added to this classpath, anyway.

## EnhancerStarter.java: main

```
1 public static void main(String[] args) throws Exception {
2     ClassFilter filter = new ByNameClassFilter("enhancement.", true);
3     Bl ootClassEdit[] edits = { new TranslateNQToSODAEdit() , new
4     InjectTransparentActivationEdit(filter) };
5     URL[] urls = { new File("/work/workspaces/db4o/tatest/bin").toURI().toURL() };
6     Db4oInstrumentationLauncher.launch(edits, urls, EnhancerMain.class.getName(), new
String[]{} );
7 }
```

Try this code now - if everything is correct you will see that the objects are getting activated as they are requested. NQ info also should say that the queries are preoptimized.

Note that for load time instrumentation to work, the application code has to make sure db4o operates on the appropriate classloader for the persistent model classes.

This revision (4) was last Modified 2007-11-08T16:34:43 by Patrick Roemer.

# Build Time Enhancement

This topic applies to Java version only

For build-time enhancement you will need to put the following jars into the lib folder of EnhancementExample project:

- bloat-1.0.jar
- db4o-x.x-classedit.jar
- db4o-x.x-java5.jar
- db4o-x.x-nqopt.jar
- db4o-x.x-taj.jar
- db4o-x.x-tools.jar
- pilot.jar
- tacustom.jar

pilot.jar supplies Pilot and Id classes presented [before](#)

tacustom.jar provides an annotation-based class filter. Using this jar we can mark the classes that should be enhanced with @Db4oPersistent annotation to filter them from the other classes.

You can use the following script to enhance your classes:

**build.xml**

```

001 <?xml version="1.0"?>
002
003 <!--
004   TA and NQ build time instrumentation sample.
005 -->
006
007 <project name="enhance_db4o" default="run-enhanced">
008
009 <!-- The classpath needed for the enhancement process, including the application classpath
-->
010 <path id="db4o.enhance.path">
011   <fileset dir="lib">
012     <include name="**/*.jar"/>
013   </fileset>
014 </path>
015
016 <!-- Define enhancement tasks (from resource in db4otools.jar). -->
017 <typedef
018   resource="instrumentation-def.properties"
019   classpathref="db4o.enhance.path"
020   loaderref="db4o.enhance.loader" />
021

```

```

022 <!-- A custom filter that selects classes with the @Db4oPersistent annotation -->
023 <typedef
024   name="annotation-filter"
025   classname="tacustom.AnnotationClassFilter"
026   classpathref="db4o增强路径"
027   loaderref="db4o增强器" />
028
029 <!-- Example for a regexp pattern for selecting classes to be instrumented. -->
030 <regexp pattern="^enhancement\..model\." id="re.model.only" />
031
032
033
034 <target name="compile">
035
036   <mkdir dir="${basedir}/bin" />
037   <delete dir="${basedir}/bin" quiet="true">
038     <include name="**/*" />
039   </delete>
040
041   <javac srcdir="${basedir}/src" destdir="${basedir}/bin" source="1.5" target="1.5">
042     <classpath refid="db4o增强路径" />
043   </javac>
044
045 </target>
046
047
048 <target name="enhance" depends="compile">
049
050   <!-- Prepare the target folders -->
051   <mkdir dir="${basedir}/enhanced-bin" />
052   <delete dir="${basedir}/enhanced-bin" quiet="true">
053     <include name="**/*" />
054   </delete>
055   <mkdir dir="${basedir}/enhanced-lib" />
056   <delete dir="${basedir}/enhanced-lib" quiet="true">
057     <include name="**/*" />
058   </delete>
059
060   <db4o-instrument classTargetDir="${basedir}/enhanced-bin" jarTargetDir="${basedir}/
061   enhanced-lib">

```

```
062 <classpath refid="db4o.enhance.path" />
063 <!-- Fileset for original class files to be instrumented -->
064 <sources dir="${basedir}/bin">
065   <include name="enhancement/**/*.class" />
066 </sources>
067 <!-- Fileset for original jars to be instrumented -->
068 <jars dir="${basedir}/lib">
069   <include name="pilot.jar" />
070 </jars>
071
072 <!-- Instrument Native Query predicates -->
073 <native-query-step />
074
075           <!-- Instrument TA field access. -->
076 <transparent-activation-step>
077   <!-- Instrument classes that are annotated as @Db4oPersistent. -->
078   <annotation-filter />
079   <!-- Instrument classes from the specified paths only. -->
080   <regexp refid="re.model.only" />
081     <regexp pattern="^enhancement\.model\." />
082 </transparent-activation-step>
083
084 </db4o-instrument>
085
086 </target>
087
088
089 <target name="run-unenhanced" depends="compile">
090
091   <java classname="enhancement.EnhancerMain" failonerror="true">
092     <classpath>
093       <path element location="${basedir}/bin" />
094       <path element location="${basedir}/lib/pilot.jar" />
095       <path refid="db4o.enhance.path" />
096     </classpath>
097   </java>
098
099 </target>
100
101 <target name="run-enhanced" depends="enhance">
102
```

```

103  <j ava classname="enhancement. EnhancerMain" failonerror="true">
104    <cl asspath>
105      <pathel ement location="${basedir}/enhanced-bin" />
106      <pathel ement location="${basedir}/enhanced-lib/pilot.jar" />
107      <path refid="db4o. enhance. path" />
108    </cl asspath>
109  </j ava>
110
111 </target>
112
113
114 </proj ect>

```

The core part of this script is inside the *db4o-instrument* task, which is imported by the first typedef instruction. (The second typedef imports the custom annotation class filter.) The *classTargetDir* and *jarTargetDir* attributes specify the target folders where instrumented class files and instrumented jar files should be created, respectively.

The nested *classpath* is just a normal Ant path type and should cover the full application classpath. In the example, we are using one single classpath for task definition and application for convenience - in a real project, these are better kept separate, of course. The nested *sources* FileSet specifies the location of the class files to be instrumented. Similarly, the *jars* FileSet specifies the location of jar files to be instrumented. Both are optional (providing neither *sources* nor *jars* doesn't make much sense, of course). If one is left out, the corresponding target folder attribute is not required, either.

The remaining nested arguments specify the instrumentation steps to be processed. For Native Query optimization, there is no further configuration - it will simply try to instrument all Predicate implementations. Transparent Activation instrumentation allows to specify more fine-grained filters to select the classes to be instrumented. This can be Ant regular expression types or arbitrary custom *ClassFilters*. These are OR-ed together and used to further constrain the implicit filter provided by the *sources/jars* FileSets. In the example, we are constraining TA instrumentation to classes that are either annotated with the *@Db4oPersistent* annotation, or whose fully qualified name matches the given regexes.

After running the *enhance* target, the instrumented model classes should appear in the enhanced-bin folder, and an instrumented version of the pilot.jar should have been created in the enhanced-lib folder.

For rather straightforward projects you can alternatively use the *db4o-enhance* task variant that provides a default setting for joint NQ/TA instrumentation (but doesn't allow fine-grained configuration for the single instrumentation steps in return). This is demonstrated by the following build script for the same sample project.

build-simple.xml

```

01 <?xml version="1.0"?>
02
03 <!-- 
04 Simple TA and NQ build time instrumentation sample.
05
06 This version uses db4o-enhance instead of db4o-instrument. db4o-enhance provides a default
07 configuration for NQ/TA instrumentation, while db4o-instrument requires to configure
08 (and optionally fine-tune) the single instrumentation steps. Other than that, the
09 configuration options for the two are identical.

```

```
10  -->
11
12 <project name="enhance_db4o" default="run-enhanced">
13
14 <!-- The classpath needed for the enhancement process, including the application classpath
-->
15 <path id="db4o.enhance.path">
16   <pathelment path="${basedir}" />
17   <fileset dir="lib">
18     <include name="**/*.jar"/>
19   </fileset>
20 </path>
21
22 <!-- Define enhancement tasks (from resource in db4otools.jar). -->
23 <typedef
24   resource="instrumentation-def.properties"
25   classpathref="db4o.enhance.path" />
26
27
28 <target name="compile">
29
30   <mkdir dir="${basedir}/bin" />
31   <delete dir="${basedir}/bin" quiet = "true">
32     <include name="**/*"/>
33   </delete>
34
35   <javac srcdir="${basedir}/src" destdir="${basedir}/bin">
36     <classpath refid="db4o.enhance.path" />
37   </javac>
38
39 </target>
40
41
42 <target name="enhance" depends="compile">
43
44   <!-- Prepare the target folders -->
45   <mkdir dir="${basedir}/enhanced-bin" />
46   <delete dir="${basedir}/enhanced-bin" quiet = "true">
47     <include name="**/*"/>
48   </delete>
49   <mkdir dir="${basedir}/enhanced-lib" />
```

```
50  <delete dir="${basedir}/enhanced-lib" quiet="true">
51    <include name="**/*" />
52  </delete>
53
54  <db4o-enhance classTargetDir="${basedir}/enhanced-bin" jarTargetDir="${basedir}/enhanced-
55 lib">
56
57    <classpath refid="db4o enhance path" />
58    <!-- Fileset for original class files to be instrumented -->
59    <sources dir="${basedir}/bin" />
60    <!-- Fileset for original jars to be instrumented -->
61    <jars dir="${basedir}/lib">
62      <include name="pilot.jar" />
63    </jars>
64
65  </db4o-enhance>
66
67
68
69 <target name="run-unenhanced" depends="compile">
70
71  <java classname="enhancement.EnhancerMain" failonerror="true">
72    <classpath>
73      <pathel ement location="${basedir}/bin" />
74      <pathel ement location="${basedir}/lib/pilot.jar" />
75      <path refid="db4o enhance path" />
76    </classpath>
77  </java>
78
79 </target>
80
81
82 <target name="run-enhanced" depends="enhance">
83
84  <java classname="enhancement.EnhancerMain" failonerror="true">
85    <classpath>
86      <pathel ement location="${basedir}/enhanced-bin" />
87      <pathel ement location="${basedir}/enhanced-lib/pilot.jar" />
88      <path refid="db4o enhance path" />
89    </classpath>
```

```
90  </j ava>
91
92  </target>
93
94
95  </proj ect>
```

This revision (6) was last Modified 2007-11-21T13:37:27 by Tetyana.

# Enhancement For .NET

This topic applies to .NET version only

.NET Enhancement functionality resides in Db4oTool project. Db4oTool is a command-line utility, which can be used to perform different bytecode instrumentation tasks. Currently, Db4oTool can be used for [Native Query Optimization](#) and [Transparent Activation Instrumentation](#).

Db4oTool project includes a command-line utility (Db4oTool.exe) and Db4oTool.MSBuild.dll, which can be used for command-line and build-time instrumentation accordingly.

Further reading:

- Enhancement from a [command-line](#)
- [Build Time Enhancement](#)

This revision (4) was last Modified 2007-12-01T13:11:06 by Tetyana.

# Build Time Enhancement

This topic applies to .NET version only

Build-time enhancement for .NET can be done by adding a special enhancement task to MSBuild file. If you are working with Visual Studio 2005, you will typically have to correct the automatically generated \*.csproj or \*.vbproj file. Db4o enhancement task for .NET (Db4oEnhancerMSBuildTask class) is included in Db4oTool.Msbuild.dll from Db4oTool distribution. Db4oEnhancerMSBuildTask applies both NQ optimization and TA/TP instrumentations in one go.

More Reading:

- [TA and NQ Example Code](#)
- [TP Example Code](#)
- [Example Enhancement](#)
- [SensorPanel](#)

This revision (3) was last Modified 2008-01-20T09:31:33 by Tetyana.

# TA and NQ Example Code

This topic applies to .NET version only

Let's look at an example. We will use [SensorPanel](#) class from [Activation](#) example, which represents a simple linked list.

In order to check if SensorPanel is TA enabled in the runtime we will add the following Diagnostic listener:

10 linked instances of SensorPanel will be stored and retrieved:

The whole project code can be downloaded here: [c#](#) [vb](#)

This revision (1) was last Modified 2007-12-01T13:54:26 by Tetyana.

# TP Example Code

This topic applies to .NET version only

Let's look at Transparent Persistence example. We will use [SensorPanel](#) class from [Activation](#) example, which represents a simple linked list.

First of all we must configure the database to use TP:

Now, we should explicitly store the objects that we want to be persistent:

In order to test if TP actually works, we will select all the SensorPanel objects from the database, modify them and commit the transaction. If TP took place the objects will be modified in the database:

The code above is ready to accommodate TP, however TP is not possible now as SensorPanel does not implement IActivatable interface. In the [following chapter](#) we will learn how to enable TP for the project classes in the build time.

The whole project code can be downloaded here: [c#](#) [vb](#)

This revision (3) was last Modified 2008-01-20T10:09:50 by Tetyana.

# Example Enhancement

This topic applies to .NET version only

If you've already set up the [TA](#) or [TP](#) example project in the Visual Studio, you can try to compile and run it. In current state TA example produces diagnostic messages about classes that are not Activatable, and TP example does not really save any runtime modifications to SensorPanel objects. Let's fix this.

First of all you will need to add a reference to Db4oTool.MSBuild.dll (from Db4oTool distro) to your project. Note, that this dll in its turn references Db4oTool.exe, Mono.Cecil.dll and Mono.GetOptions.dll, so you must keep them in the same directory.

Open the project file (csproj for c# and vbproj for VB) in any text editor and add the following lines at the end of the file before the closing </Project> tag:

```
<UsingTask AssemblyFile="$(OutputPath)Db4oTool.MSBuild.dll"
  TaskName="Db4oTool.MSBuild.Db4oEnhancerMSBuildTask" />

<ItemGroup>
  <Db4oEnhance Include="$(TargetPath)" />
</ItemGroup>

<Target Name="AfterBuild">
  <Db4oEnhancerMSBuildTask Assemblies="@(\Db4oEnhance)" />
</Target>
```

Save the file and go back to VS (a note should appear asking to update the project file, you must agree to update). Now rebuild and run the application once again. In the build file we did not specify any filters, so all the application classes should be Activatable now and all diagnostic messages should be gone and the modifications should be saved in TP example.

Note, that though the first example does not provide diagnostic information for Native Queries, NQ were also optimized by the updated build script.

This revision (3) was last Modified 2008-01-21T06:17:58 by Tetyana.

# SensorPanel

This topic applies to .NET version only

This revision (3) was last Modified 2007-12-01T14:22:41 by Tetyana.

# Db4oTool

This topic applies to .NET version only.

Db4oTool.exe utility is distributed together with db4o .NET version and can be used for Native Query optimization and Transparent Activation Instrumentation.

The main use-cases for Db4oTool are:

1. [Optimization of NQ](#) at build time. This will improve Native Query performance by cutting of query analyzing time during execution.
2. [Optimizing delegate NQ syntax on CF2.0](#). This optimization can only be done by Db4oTool as CompactFramework 2.0 API does not expose any of the delegate metadata needed for the optimization process.
3. [Transparent Activation](#) instrumentation. This will enable you to use TA without modifying your classes or to use TA on third-party classes.
4. [Transparent Persistence](#) instrumentation. This will enable you to use TP without modifying your classes or to use TP on third-party classes. TP implicitly includes TA instrumentation for the same classes.

If you use Db4oTool for use-cases 1 and 2 you will be able to distribute your application without Db4objects.Db4o.NativeQueries.dll (the assembly where the Native Query runtime optimizer lives).

More Reading:

- [Db4oTool Usage](#)
- [Including Db4oTool In The Build](#)

This revision (8) was last Modified 2008-01-20T08:46:55 by Tetyana.

# Db4oTool Usage

This topic applies to .NET version only.

This topic is under development.

Db4oTool is a command line utility. The general syntax is the following:

Usage: Db4oTool [options] <assembly>

[options] parameter allows to specify a list of options.

<assembly> parameter allows to pass an assembly, which should be optimized.

Both parameters are optional.

Running Db4oTool.exe without any parameters will bring you a short usage hint. This is equivalent to running Db4oTool with -? or -help parameter. Additional help information can be retrieved with -help2 or -usage parameters.

The table below gives an explanation of all Db4oTool options.

-byattribute:PARAM	Filter types to be instrumented by attribute:  Db4oTool -ta -byattribute:Activatable MyAssembly.exe
-byfilter:PARAM	Custom type filter:  Db4oTool -ta -byfilter:IActivatable MyAssembly.exe
-byname:PARAM	Filter types by name (with regular expression syntax):  Db4oTool -ta -byname:MyCompany.MyProduct MyAssembly.exe
-out	Negate the last filter  Db4oTool -ta -byname:Db4oObjects.Db4o -out MyAssembly.exe

-case-sensitive	Specifies if optimized queries should be case-sensitive. This option should be used in conjunction with query optimization option (nq):  Db4oTool -nq -case-sensitive MyAssembly.exe
-fake	Fake operation mode, assembly won't be written. This option can be used for testing before the actual run.  Db4oTool -nq -fake MyAssembly.exe
-? -help	Show standard help list:  Db4oTool -help
-help2	Show an additional help list (development use):  Db4oTool -help2
-instrumentation: PARAM	Use custom instrumentation type.  PARAM is a string with a full class definition, like  Db4oTool.AbstractAssemblyInstrumentation, Db4oTool.exe.  This class must implement Db4oTool.IAssemblyInstrumentation interface. To make the creation of a custom instrumentation class easier db4o provides Db4oTool.AbstractAssemblyInstrumentation class, which can be used as a template. For an example implementation see Db4oTool.TAInstrumentation class.
-nq	Optimize Native Queries  Db4oTool -nq MyAssembly.exe
-ta	Instrument classes to support Transparent Activation:  Db4oTool -ta MyAssembly.exe

-tp	Instrument classes to support Transparent Persistence (Transparent Activation support is included implicitly):  Db4oTool -tp MyAssembly.exe
-usage	Show usage syntax and exit:  Db4oTool -usage
-v -verbose	Verbose operation mode. Should be combined with the other options:  Db4oTool -ta -v MyAssembly.exe
-V -version	Display version and licensing information:  Db4oTool -V
-vv	Pretty verbose operation mode:  Db4oTool -ta -vv MyAssembly.exe

This revision (5) was last Modified 2008-01-20T09:01:29 by Tetyana.

# Including Db4oTool In The Build

This topic applies to .NET version only.

The easiest way to use Db4oTools is to include it directly into the build process. This will enable you to get a processed assembly immediately after a successful build. If you are using Visual Studio Compact Framework emulator, you will get the processed assembly straight into the emulator.

Use the following steps to enable Db4oTool in VS2005 project:

- Make sure that Db4oTool.exe together with the other libraries from the bin folder in the distribution are accessible to your project.
- Open Project Properties page and select "Build Events" tab.
- In the "Post-build event command line:" enter the required Db4oTool command:
  - [path\_to\_Db4oTool\_folder]/Db4oTools.exe [options] \$(TargetPath)  
if Db4oTool is included in your project's references, [path\_to\_Db4oTool\_folder] should be skipped.
- In the "Run the post-build event:" select "On successful build"
- Run the build. The resulting assembly will contain all the modifications made by Db4oTool.

An example of this is included in the distribution in src/Db4oTool/Db4oTool.Example folder.

Possible usages:

1. Native Query optimization:  
[path\_to\_Db4oTool\_folder]/Db4oTools.exe -nq \$(TargetPath)
2. Transparent Activation support for the whole assembly:  
[path\_to\_Db4oTool\_folder]/Db4oTools.exe -ta \$(TargetPath)
3. Transparent Activation support for MyClass objects:  
[path\_to\_Db4oTool\_folder]/Db4oTools.exe -ta -byname:MyClass \$(TargetPath)
4. Transparent Activation and Transparent Persistence support:  
[path\_to\_Db4oTool\_folder]/Db4oTools.exe -tp \$(TargetPath)

For more Db4oTool options please refer to [Db4oTool Usage](#).

This revision (5) was last Modified 2008-01-20T09:07:54 by Tetyana.

# Using Attributes

This topic applies to .NET version only

.NET Attributes provide the means for a developer to add meta-data that describes, or annotates specific elements of code such as classes, methods, properties, etc. At compile time the resulting metadata is placed into the Portable Executable (PE)file along with the Microsoft Intermediate Language (MSIL). Once metadata is in the PE other .NET programs may access it using the .NET Reflection API.

Attributes can be used to document classes at design time, specify runtime information (such as the name of an XML field to be used when serializing information from the class), and even dictate runtime behavior (such as whether the class should automatically participate in a transaction).

You can use attributes with db4o to configure how db4o will process your classes. At present we provide only one attribute:

[ Indexed ]

This attribute can be applied to class fields

and its functionality is equivalent to the db4o configuration setting:

```
Db4o.Configure().ObjectClass(clazz).ObjectField("fieldName").Indexed(true)
```

This revision (6) was last Modified 2006-12-04T07:56:36 by Tetyana.

# Unique Constraints

Unique Constraints feature was first introduced in db4o 6.2.

Unique Constraints allow a user to define a field to be unique across all the objects of a particular Class stored to db4o. This means that you cannot save an object where a previously committed object has the same field value for fields marked as unique.

A Unique Constraint is checked at commit-time and a constraint violation will cause a `UniqueFieldValueConstraintViolationException` to be thrown. This functionality is based on [Commit-Time Callbacks](#) feature.

Multiple constraints can be defined on the same class if required.

More Reading:

- [How To Use Unique Constraints](#)
- [Unique Constraints Example](#)

This revision (3) was last Modified 2007-05-07T10:04:49 by Tetyana.

# How To Use Unique Constraints

In order to work with the unique constraints you will need to remember the following 3 steps.

1. Add an index for a field you wish to be unique:

Java:

```
configuration.objectClass(Item.class).objectField("field").indexed(true);
```

This revision (2) was last Modified 2007-03-25T07:18:49 by Tetyana.

# Unique Constraints Example

Let's look at a simple example with 2 clients adding objects of the same class to the database. First of all, let's create an appropriate configuration:

UniqueConstraintExample.java: configure

```

1  private static Configuration configure() {
2      Configuration configuration = Db4o.newConfiguration();
3      configuration.objectClass(Pilot.class).objectField("name")
4          .indexed(true);
5      configuration.add(new UniqueFieldValueConstraint(Pilot.class,
6          "name"));
7      return configuration;
8  }

```

Configuration returned by `configure` method will be passed to the server.

UniqueConstraintExample.java: storeObjects

```

01  private static void storeObjects() {
02      new File(FILENAME).delete();
03      ObjectServer server = Db4o.openServer(configure(), FILENAME,
04          0);
05      Pilot pilot1 = null;
06      Pilot pilot2 = null;
07      try {
08          ObjectContainer client1 = server.openClient();
09          try {
10              // creating and storing pilot1 to the database
11              pilot1 = new Pilot("Rubens Barrichello", 99);
12              client1.set(pilot1);
13          ObjectContainer client2 = server.openClient();
14          try {

```

```

15 |         // creating and storing pilot2 to the database
16 |         pilot2 = new Pilot("Rubens Barichello", 100);
17 |         client2.set(pilot2);
18 |         // commit the changes
19 |         client2.commit();
20 |     } catch (UniqueFieldValueConstraintViolationException ex) {
21 |         System.out
22 |             .println("Unique constraint violation in client2 saving: "
23 |                     + pilot2);
24 |         client2.rollback();
25 |     } finally {
26 |         client2.close();
27 |     }
28 |     // Pilot Rubens Barichello is already in the database,
29 |     // commit will fail
30 |     client1.commit();
31 | } catch (UniqueFieldValueConstraintViolationException ex) {
32 |     System.out
33 |         .println("Unique constraint violation in client1 saving: "
34 |                     + pilot1);
35 |     client1.rollback();
36 | } finally {
37 |     client1.close();
38 | }
39 | } finally {
40 |     server.close();
41 | }
42 |

```

Running this example you will get an exception trying to commit `client1` as the changes made by `client2` containing a `Pilot` object with the same name are already committed to the database.

This revision (1) was last Modified 2007-03-25T07:20:10 by Tetyana.

# Object Callbacks

Callback methods are automatically called on persistent objects by db4o during certain database events.

For a complete list of the signatures of all available methods see the com.db4o.ext.ObjectCallbacks interface.

You do not have to implement this interface. db4o recognizes the presence of individual methods by their signature, using reflection. You can simply add one or more of the methods to your persistent classes and they will be called.

Returning false to the #objectCanXXXX( ) methods will prevent the current action from being taken.

In a client/server environment callback methods will be called on the client with two exceptions:

objectonDelete( ), objectCanDelete( )

Some possible usecases for callback methods:

- setting default values after refactorings
- checking object integrity before storing objects
- setting transient fields
- restoring connected state (of GUI, files, connections)
- cascading activation
- cascading updates
- creating special indexes

This revision (3) was last Modified 2006-11-13T11:06:05 by Tetyana.

# Callbacks

External Callbacks enable you to add Listeners to an ObjectContainer for the following db4o events

- QueryStarted
- QueryFinished
- Creating (first time an object is about to be saved)
- Created (after the object is saved)
- Activating
- Activated
- Deactivating
- Deactivated
- Updating
- Updated
- Deleting
- Deleted
- Committing
- Committed

QueryStarted and QueryFinished events accept QueryEventArgs as a parameter and can be used to gather query statistics information.

Created, Activated, Deactivated, Updated and Deleted events accept ObjectEventArgs and can be used to gather statistics information or to initiate some special behavior after the action has been taken.

Creating, Activating, Deactivating, Updating and Deleting events accept CancellableObjectEventArgs. Their primary usage is to perform action validity check and to stop the execution if necessary.

Committing event can be used to check some application-specific conditions before commit. For example it can

be used to check unique constraints. Committing event accepts CommitEventArgs as a parameter.

Committed event is raised when the container has completely finished the commit operation. Event subscribers get the notification in a separate thread. Committed event accepts CommitEventArgs.

More Reading:

- [Event Registry API](#)
- [Possible Usecases](#)
- [Benefits](#)
- [Commit-Time Callbacks](#)

This revision (4) was last Modified 2007-04-28T09:33:35 by Tetyana.

# Event Registry API

External callbacks should be registered with db4o EventRegistry. Follow the steps below to start using your own event handlers:

1. Obtain an instance of EventRegistry object for your ObjectContainer

Java:

```
EventRegistry registry = EventRegistryFactory.forObjectContainer(container);
```

2. Register the required event. For "created" event the code is the following:

Java:

```
registry.created().addListener(EventListener4)
```

3. Create your own event handler:

Java:

```
EventListener4 createdEvent = new EventListener4() {
    onEvent(Event4 event, EventArgs args) {
        // handling code
    }
}
```

The action raised the event can be cancelled in Creating, Activating, Deactivating, Updating and Deleting event handlers. These events accept CancellableEventArgs as a parameter. In order to cancel the action use:

Java:

```
cancellableEventArgs.cancel()
```

Here cancellableEventArgs is an event argument of CancellableEventArgs type.

In java cancellableEventArgs should be obtained by explicit casting:

```
EventListener4 listener = new EventListener4() {
    public void onEvent(Event4 e, EventArgs args) {
        CancellableEventArgs cancellableArgs = (CancellableEventArgs)args;
        ....
    }
}
```

[/filter]

4. After the work is done you can unregister the events:

[filter=java]

Java:

```
registry.created().removeListener(createdEvent);
```

EventRegistry features:

- You can register several event handlers for a single event.
- You can get different EventRegistry's for different ObjectContainer instances using EventRegistryFactory.
- In Java, callbacks are implemented as Listeners, .NET uses Native events
- Callbacks only work run in local mode or on the server side in client/server mode.
- Each event applies to all the objects or queries(QueryStarted/QueryFinished events ). In order to distinguish the specific case, to which the handler should be applied, use the event arguments.

For example:

CallbacksExample.java: testCreated

```
01  private static void testCreated(){
02 |     new File(DB40_FILE_NAME).delete();
03 |     ObjectContainer container = Db4o.openFile(DB40_FILE_NAME);
04  |     try {
05 |         EventRegistry registry = EventRegistryFactory.forObjectContainer(container);
06 |         // register an event handler, which will print all the car objects, that have been
07 |         // created
08  |         registry.created().addListener(new EventListener4() {
09 |             public void onEvent(Event4 e, EventArgs args) {
10 |                 ObjectEventArgs queryArgs = ((ObjectEventArgs) args);
11 |                 Object obj = queryArgs.getObject();
12 |                 if (obj instanceof Pilot) {
13 |                     System.out.println(obj.toString());
14 |                 }
15 |             });
16 |
17 |             Car car = new Car("BMW", new Pilot("Rubens Barrichello"));
18 |             container.set(car);
19  |         } finally {
20 |             container.close();
21 |         }
22 }
```

This revision (11) was last Modified 2007-04-28T09:39:20 by Tetyana.

# Possible Usecases

There are many usecases for external callbacks, including:

- cascaded deletes, updates
- referential integrity checks
- gathering statistics
- autoassigned fields
- assigning customary unique IDs for external referencing
- delayed deletion (objects are marked for deletion when `delete(object)` is called and cleaned out of database in a later maintenance operation)
- ensuring object fields uniqueness within the same class etc.

More Reading:

- [Cascaded Behavior](#)
- [Referential Integrity](#)
- [Query Statistics](#)
- [Autoincrement](#)

This revision (8) was last Modified 2007-01-02T16:22:05 by Tetyana.

# Cascaded Behavior

You can use different object events to initiate cascaded behavior on update, activate, delete. The following example shows how to ensure that all the referenced objects are deleted when the parent object is deleted:

CallbacksExample.java: testCascadedDelete

```

01  private static void testCascadedDelete() {
02 |     fillDB();
03 |     final ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
04 |     try {
05 |         // check the contents of the database
06 |         ObjectSet result = container.get(null);
07 |         listResult(result);
08 |
09 |         EventRegistry registry = EventRegistryFactory.forName(container);
10 |         // register an event handler, which will delete the pilot when his car is
deleted
11 |         registry.deleted().addListener(new EventListener4() {
12 |             public void onEvent(Event4 e, EventArgs args) {
13 |                 ObjectEventArgs queryArgs = ((ObjectEventArgs) args);
14 |                 Object obj = queryArgs.object();
15 |                 if (obj instanceof Car){
16 |                     container.delete(((Car) obj).getPilot());
17 |                 }
18 |             }
19 |         });
20 |         // delete all the cars
21 |         result = container.query(Car.class);
22 |         while(result.hasNext()) {
23 |             container.delete(result.next());
24 |         }
25 |         // check if the database is empty
26 |         result = container.get(null);
27 |         listResult(result);
28 |     } finally {
29 |         container.close();
30 |     }
31 }
```

This revision (3) was last Modified 2007-12-02T16:56:25 by Tetyana.

# Referential Integrity

Db4o does not have a built-in referential integrity checking mechanism. Luckily EventRegistry gives you access to all the necessary events to implement it. You will just need to trigger validation on create, update or delete and cancel the action if the integrity is going to be broken.

For example, if Car object is referencing Pilot and the referenced object should exist, this can be ensured with the following handler in deleting() event:

CallbacksExample.java: testIntegrityCheck

```

01  private static void testIntegrityCheck() {
02 |     fillDB();
03 |     final ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
04 |     try {
05 |         EventRegistry registry = EventRegistryFactory.forObjectContainer(container);
06 |         // register an event handler, which will stop deleting a pilot when it is referenced
from a car
07 |         registry.deleting().addListener(new EventListener4() {
08 |             public void onEvent(Event4 e, EventArgs args) {
09 |                 CancelableEventArgs cancelEventArgs = ((CancelableEventArgs) args);
10 |                 Object obj = cancelEventArgs.object();
11 |                 if (obj instanceof Pilot) {
12 |                     Query q = container.query();
13 |                     q.constrain(Car.class);
14 |                     q.descend("pilot").constrain(obj);
15 |                     ObjectSet result = q.execute();
16 |                     if (result.size() > 0) {
17 |                         System.out.println("Object " + (Pilot) obj + " can't be deleted as object
container has references to it");
18 |                         cancelEventArgs.cancel();
19 |                     }
20 |                 }
21 |             }
22 |         });
23 |
24 |         // check the contents of the database
25 |         ObjectSet result = container.get(null);
26 |         listResult(result);
27 |
28 |         // try to delete all the pilots
29 |         result = container.query(Pilot.class);

```

```
30 |     while(result.hasNext()) {  
31 |         container.delete(result.next());  
32 |     }  
33 |     // check if any of the objects were deleted  
34 |     result = container.get(null);  
35 |     listResult(result);  
36 | } finally {  
37 |     container.close();  
38 | }  
39 | }
```

You can also add handlers for creating() and updating() events for a Car object to make sure that the pilot field is not null.  
This revision (2) was last Modified 2007-12-03T06:01:03 by Tetyana.

# Query Statistics

This topic applies to Java version only

QueryStarted and QueryFinished events can be used to gather various statistics about query execution.

For example you can use queryStarted() and queryFinished() events to calculate query time, and activated() event to count objects activated in a query.

Java version provides an example of query events usage in QueryStats class from tools package.

In order to use QueryStats:

1. Connect a QueryStats object to the ObjectContainer

```
QueryStats stats = new QueryStats( );  
  
stats.connect(db);
```

2. Execute your query

```
ObjectSet result = q.execute( );
```

3. Get the metrics from the QueryStats object

```
long executionTime = stats.executionTime( );  
  
int activationCount = stats.activationCount( );
```

4. After the work is done you can disconnect QueryStats from the ObjectContainer

```
stats.disconnect( );
```

This revision (1) was last Modified 2007-01-02T16:21:47 by Tetyana.

# Autoincrement

Db4o does not deliver a field autoincrement feature, which is common in RDBMS. If your application logic requires this feature you can implement it using External Callbacks. One of the possible solutions is presented below.

We will need an object to store the last generated ID and to return a new ID on request:

IncrementedId.java

```
01 /* Copyright (C) 2004 - 2006 db4objects Inc. http://www.db4o.com */
02 /*
03 | * Singleton class used to keep autoincrement information
04 | * and give the next available ID on request
05 | */
06 package com.db4odoc.callbacks;
07
08 import com.db4o.ObjectContainer;
09 import com.db4o.ObjectSet;
10
11 public class IncrementedId {
12 |   private int no;
13 |   private static IncrementedId ref;
14 |
15   private IncrementedId() {
16 |     this.no = 0;
17 | }
18 |
19 | // end IncrementedId
20 |
21   public int getNextID(ObjectContainer db) {
22 |   no++;
23 |   db.set(this);
24 |   return no;
25 | }
26 |
```

```

27 | // end increment
28 |
29 | public static IncrementedId getIdObject( ObjectContainer db) {
30 |     // if ref is not assigned yet:
31 |     if (ref == null) {
32 |         // check if there is a stored instance from the previous
33 |         // session in the database
34 |         ObjectSet os = db.get(IncrementedId.class);
35 |         if (os.size() > 0)
36 |             ref = (IncrementedId) os.next();
37 |     }
38 |
39 |     if (ref == null) {
40 |         // create new instance and store it
41 |         System.out.println("Id object is created");
42 |         ref = new IncrementedId();
43 |         db.set(ref);
44 |     }
45 |     return ref;
46 | }
47 | // end getIdObject
48 |}

```

This object generates the simplest ID, which is an autoincremented integer value. You can add your own algorithm to generate more sophisticated ID sequences, like ABC0001DEF.

When you use external callbacks you are not limited to a single object: a callback can apply to any group of objects. Thus you can create a sequence of classes sharing the same autoincrement. To distinguish the objects, which will have an autoincremented field, we will use an abstract (MustInherit in VB) class:

#### CountedObject.java

```

01 | /* Copyright (C) 2004 - 2006 db4objects Inc. http://www.db4o.com */
02 | /*
03 | * This class is used to mark classes that need to get an autoincremented ID
04 | */
05 | package com.db4odoc.callbacks;

```

```

06
07
08  public abstract class CountedObject {
09 |   int id;
10 |
11  public void setId(int id) {
12 |     this.id = id;
13 | }
14 |
15  public int getId() {
16 |   return id;
17 | }
18 }

```

Each object extending CountedObject will get an autoincremented ID. For example:

TestObject.java

```

01  /* Copyright (C) 2004 - 2006 db4objects Inc. http://www.db4o.com */
02  package com.db4odoc.callbacks;
03
04  public class TestObject extends CountedObject{
05 |   String name;
06 |
07  public TestObject(String name) {
08 |   this.name = name;
09 | }
10 |
11  public String toString() {
12 |   return name+"/"+id;
13 | }
14 |
15 }

```

It is only left to register the callback with the creating() event:

## AutolncExample.java: registerCallback

```

01 public static void registerCallback(final ObjectContainer db) {
02 |     EventRegistry registry = EventRegistryFactory.forObjectContainer(db);
03 |     // register an event handler, which will assign autoincremented IDs to any
04 |     // object extending CountedObject, when the object is created
05 |     registry.create().addListener(new EventListener4() {
06 |         public void onEvent(Event4 e, EventArgs args) {
07 |             ObjectEventArgs queryArgs = ((ObjectEventArgs) args);
08 |             Object obj = queryArgs.object();
09 |             // only for the objects extending the CountedObject
10 |             if (obj instanceof CountedObject) {
11 |                 ((CountedObject) obj).setIdgetNextId(db));
12 |             }
13 |         }
14 |     });
15 }

```

## AutolncExample.java: getNextId

```

1 private static int getNextId(ObjectContainer db) {
2 |     // this function retrieves the next available ID from
3 |     // the IncrementedId object
4 |     IncrementedId r = IncrementedId.getIdForObject(db);
5 |     int nRoll;
6 |     nRoll = r.getNextID(db);
7 |
8 |     return nRoll;
9 }

```

You can test the results with the following code:

## AutolncExample.java: storeObjects

```

1 public static void storeObjects(ObjectContainer db) {
2 |     TestObject test;

```

```
3 |     test = new TestObject("FirstObject");
4 |     db.set(test);
5 |     test = new TestObject("SecondObject");
6 |     db.set(test);
7 |     test = new TestObject("ThirdObject");
8 |     db.set(test);
9 | }
```

## AutoIncExample.java: retrieveObjects

```
1 [+]public static void retrieveObjects(ObjectContainer db) {
2 |     ObjectSet result = db.get(new TestObject(null));
3 |     listResult(result);
4 | }
```

Please, note that the suggested implementation **cannot be used in a multithreaded environment**. In such environment you will have to make sure that the IncrementedId class can only be saved to the database once, and that 2 threads cannot independently and simultaneously increment IncrementedId counter.

This revision (5) was last Modified 2007-05-07T10:09:51 by Tetyana.

# Benefits

External callbacks help you to solve many different problems and customize db4o behavior. Among their benefits:

- With external callbacks you do not have to pollute your object model with persistence code. This is exceptionally valuable when the objects are inherited from external application or library.
- Multiple event handlers can be registered on particular events, keeping your code clean and easily readable.
- You can "plug-in" different modules to perform different tasks. An example can be a module responsible for assigning unique IDs to your objects.

This revision (1) was last Modified 2006-12-05T12:44:07 by Tetyana.

# Commit-Time Callbacks

Commit-time callbacks were introduced in db4o version 6.2.

Commit-time callbacks allow a user to add some specific behavior just before and just after a transaction is committed.

Typical use-cases for commit-time callbacks:

- add constraint-violation checking before commit;
- check application-specific conditions before commit is done;
- start synchronization or backup after commit;
- notify other clients/applications about successful/unsuccessful commit.

Commit-time callbacks can be triggered by the following 2 events:

- **Committing:** event subscribers are notified before the container starts any meaningful commit work and are allowed to cancel the entire operation by throwing an exception; the ObjectContainer instance is completely blocked while subscribers are being notified which is both a blessing because subscribers can count on a stable and safe environment and a curse because it prevents any parallelism with the container;
- **Committed:** event subscribers are notified in a separate thread after the container has completely finished the commit operation; exceptions if any will be ignored.

More Reading:

- [How To Use Commit-Time Callbacks](#)
- [Committing Event Example](#)
- [Committed Event Example](#)
- [Car](#)

This revision (2) was last Modified 2007-05-02T15:36:25 by Tetyana.

# How To Use Commit-Time Callbacks

The general usage of external callbacks is described in [Event Registry API](#). Commit-time event handlers can be defined:

Java:

```
registry.committing().addListener(new EventListener4() { ... });  
registry.committed().addListener(new EventListener4() { ... });
```

Event arguments can be reached through CommitEventArgs parameter. CommitEventArgs class provides the following properties:

added - lists the objects added in the current transaction.

deleted - lists the objects deleted in the current transaction.

updated - lists the objects updated in the current transaction.

transaction - returns the current transaction.

This revision (5) was last Modified 2007-05-07T10:30:56 by Tetyana.

# Committing Event Example

Let's look at an example of committing event usage. In this example committing callback will be used to ensure that only unique objects can be saved to the database. Duplicate objects - objects, which have the same fields - will cause an exception at the commit time.

We will use a simple Item class for an example:

Item.java

```

01  /* Copyright (C) 2007 db4o objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.commitcallbacks;
04
05
06 public class Item {
07     private int _number;
08     private String _word;
09
10    public Item(int number, String word){
11        _number = number;
12        _word = word;
13    }
14
15    public String getWord(){
16        return _word;
17    }
18
19    public int getNumber(){
20        return _number;
21    }
22
23    public String toString(){
24        return _number + "/" + _word;
25    }
26 }
```

The following methods will configure a commit-time callback method for uniqueness check:

CommitCallbackExample.java: configure

```

01 private static void configure() {
02 |     EventRegistry registry = EventRegistryFactory.forObjectContainer(container());
03 |     // register an event handler, which will check object uniqueness on commit
04 registry.committing().addListener(new EventListener4() {
05 |         public void onEvent(Event4 e, EventArgs args) {
06 |             CommitEventArgs commitArgs = ((CommitEventArgs) args);
07 |             // uniqueness should be checked for both added and updated objects
08 |             checkUniqueness(commitArgs.added());
09 |             checkUniqueness(commitArgs.updated());
10 |         }
11 |     });
12 }

```

## CommitCallbackExample.java: checkUniqueness

```

01 private static void checkUniqueness(ObjectInfoCollection collection) {
02 |     Iterator4 iterator = collection.iterator();
03 while (iterator.moveToNext()) {
04 |         ObjectInfo info = (ObjectInfo) iterator.current();
05 |         // only check for Item objects
06 if (info.getObject() instanceof Item) {
07 |             Item item = (Item) info.getObject();
08 |             // search for objects with the same fields in the database
09 |             ObjectSet found = container().get(new Item(item.getNumber(), item.getWord()));
10 if (found.size() > 1) {
11 |                 throw new Db4oException("Object is not unique: " + item);
12 |             }
13 |         }
14 |     }
15 }

```

let's save one initial object `Item(1, "one")`

## CommitCallbackExample.java: storeFirstObject

```

01 private static void storeFirstObject() {
02 |     ObjectContainer container = container();
03 try {
04 |         // creating and storing item1 to the database

```

```

05 |     Item item = new Item(1, "one");
06 |     container.set(item);
07 |     // no problems here
08 |     container.commit();
09 | } catch (Db4oException ex) {
10 |     System.out.println(ex.getMessage());
11 |     container.rollback();
12 | }
13 | }
```

Now we can check the functionality of the committing callback using the following code:

#### CommitCallbackExample.java: storeOtherObjects

```

01 | private static void storeOtherObjects(){
02 |     ObjectContainer container = container();
03 |     // creating and storing similar items to the database
04 |     Item item = new Item(2, "one");
05 |     container.set(item);
06 |     item = new Item(1, "two");
07 |     container.set(item);
08 |     try {
09 |         // commit should work as there were no duplicate objects
10 |         container.commit();
11 |     } catch (Db4oException ex) {
12 |         System.out.println(ex.getMessage());
13 |         container.rollback();
14 |     }
15 |     System.out.println("Commit successful");
16 |
17 |     // trying to save a duplicate object to the database
18 |     item = new Item(1, "one");
19 |     container.set(item);
20 |     try {
21 |         // Commit should fail as duplicates are not allowed
22 |         container.commit();
23 |     } catch (Db4oException ex) {
24 |         System.out.println(ex.getMessage());
25 |         container.rollback();
```

```
26 |     }
27 L }
```

This revision (5) was last Modified 2007-09-15T18:15:56 by Tetyana.

# Committed Event Example

Committed callbacks can be used in various scenarios:

- backup on commit;
- database replication on commit;
- client database synchronization.

In our example we will create an implementation for the last case.

When several clients are working on the same objects it is very possible that the data will be outdated on some of the clients. Before the commit-callbacks feature was introduced the solution was to call `refresh` regularly to get object updates from the server. With the commit-callback this process can be easily automated:

- objects are modified when the commit is done;
- the successful commit triggers committed event on the clients;
- committed event handler updates modified objects on the clients.

Let's open 2 clients, which will work with [Car](#) objects, and register committed event listeners for them.

**PushedUpdatesExample.java: openClient**

```

01 ①②private ObjectContainer openClient() {
02 ③④    try {
03 ⑤        ObjectContainer client = Db4o.openClient("localhost", PORT, USER,
04 ⑥            PASSWORD);
05 ⑦        EventListener4 committedEventListener = createCommittedEventListener(client);
06 ⑧        EventRegistry eventRegistry = EventRegistryFactory
07 ⑨            .forObjectContainer(client);
08 ⑩        eventRegistry.committed().addListener(committedEventListener);
09 ⑪        // save the client-listener pair in a map, so that we can
10 ⑫        // remove the listener later
11 ⑬        clientListeners.put(client, committedEventListener);
12 ⑭        return client;
13 ⑮    } catch (Exception ex) {
14 ⑯        ex.printStackTrace();
15 ⑰    }
16 ⑱    return null;
17 ⑲}

```

**PushedUpdatesExample.java: createCommittedEventListener**

```

01 private EventListener4 createCommittedEventListener(
02     final ObjectContainer objectContainer) {
03     return new EventListener4() {
04         public void onEvent(Event4 e, EventArgs args) {
05             // get all the updated objects
06             ObjectInfoCollection updated = ((CommitEventArgs) args)
07                 .updated();
08             Iterator4 infos = updated.iterator();
09             while (infos.moveToNext()) {
10                 ObjectInfo info = (ObjectInfo) infos.current();
11                 Object obj = info.getObject();
12                 // refresh object on the client
13                 objectContainer.ext().refresh(obj, 2);
14             }
15         }
16     };
17 }
```

Run the following method to see how the 2 clients work concurrently on the same object:

PushedUpdatesExample.java: run

```

01 public void run() throws IOException, DatabaseFileLockedException {
02     new File(DB40_FILE_NAME).delete();
03     ObjectServer server = Db4o.openServer(DB40_FILE_NAME, PORT);
04     try {
05         server.grantAccess(USER, PASSWORD);
06
07         ObjectContainer client1 = openClient();
08         ObjectContainer client2 = openClient();
09
10         if (client1 != null && client2 != null) {
11             try {
12                 // wait for the operations to finish
13                 waitForCompletion();
14
15                 // save pilot with client1
16             } finally {
17                 client1.close();
18                 client2.close();
19             }
20         }
21     } finally {
22         if (server != null) {
23             server.close();
24         }
25     }
26 }
```

```
16 |         Car client1Car = new Car("Ferrari", 2006, new Pilot(  
17 |             "Schumacher"));  
18 |         client1.set(client1Car);  
19 |         client1.commit();  
20 |  
21 |         waitForCompletion();  
22 |  
23 |         // retrieve the same pilot with client2  
24 |         Car client2Car = (Car) client2.query(Car.class).next();  
25 |         System.out.println(client2Car);  
26 |  
27 |         // modify the pilot with client1  
28 |         client1Car.setModel(2007);  
29 |         client1Car.setPilot(new Pilot("Hakkinen"));  
30 |         client1.set(client1Car);  
31 |         client1.commit();  
32 |  
33 |         waitForCompletion();  
34 |  
35 |         // client2Car has been automatically updated in  
36 |         // the committed event handler because of the  
37 |         // modification and the commit by client1  
38 |         System.out.println(client2Car);  
39 |  
40 |         waitForCompletion();  
41 |     } catch (Exception ex) {  
42 |         ex.printStackTrace();  
43 |     } finally {  
44 |         closeClient(client1);  
45 |         closeClient(client2);  
46 |     }  
47 | }  
48 | } catch (Exception ex) {  
49 |     ex.printStackTrace();  
50 | } finally {  
51 |     server.close();  
52 | }  
53 }
```

You should see that client2 picked up the changes committed from the client1 automatically due to the committed event handler.

Working with the committed event you should remember that the listener is called in a separate thread, which needs to be synchronized with the main application thread. This functionality is not implemented in the presented example, instead a simple thread Sleep(1000) method is used (WaitForCompletion method), which is not reliable at all. For a reliable execution use events and notifications from the committed callbacks.

It is a good practice to remove the committed event handlers from the registry before shutting down the clients:

PushedUpdatesExample.java: closeClient

```
01  private void closeClient(ObjectContainer client) {
02 |     // remove listeners before shutting down
03 |     if (clientListeners.get(client) != null) {
04 |         EventRegistry eventRegistry = EventRegistryFactory
05 |             .forObjectContainer(client);
06 |         eventRegistry.committed().removeListener(
07 |             (EventListener4) clientListeners.get(client));
08 |         clientListeners.remove(client);
09 |     }
10 |     client.close();
11 }
```

This revision (3) was last Modified 2007-05-02T13:29:18 by Tetyana.

# Car

Car.java

```
01 package com.db4odoc.commitcallbacks;
02
03 public class Car {
04
05     private String name;
06
07     private int model;
08
09     private Pilot pilot;
10
11     public Car(String name, int model, Pilot pilot) {
12         this.name = name;
13         this.model = model;
14         this.pilot = pilot;
15     }
16
17     public void setModel(int model) {
18         this.model = model;
19     }
20
21     public void setPilot(Pilot pilot) {
22         this.pilot = pilot;
23     }
24
25     public String toString() {
26         return "Car: " + name + " " + model + " Pilot: " + pilot.getName();
27     }
28 }
```

## Pilot.java

```
01 package com.db4odoc.commitcallbacks;
02
03 public class Pilot {
04 |
05 |     private String name;
06 |
07 |     public String getName() {
08 |         return name;
09 |     }
10 |
11 |     public Pilot(String name) {
12 |         this.name = name;
13 |     }
14 }
```

This revision (1) was last Modified 2007-04-28T17:31:26 by Tetyana.

# Translators

The [Object Construction](#) chapter covers the alternative configurations db4o offers for object reinstantiation. What's left to see is how we can store objects of a class that can't be cleanly stored with either of these approaches.

More Reading:

- [Java Example Class](#)
- [.NET Example Class](#)
- [The Translator API](#)
- [.NET Translator Implementation](#)
- [Java Translator Implementation](#)
- [Built-In Translators](#)

This revision (11) was last Modified 2007-05-07T10:33:46 by Tetyana.

# Java Example Class

## Contents

- Bypassing the constructor
- Using the constructor

This topic applies to Java version only

Let's use the following class as an example of a class which can not be stored clearly with db4o.

### NotStorable.java

```
01 /* Copyright (C) 2004 - 2007 db4objects Inc. http://www.db4o.com */
02 package com.db4odoc.translators;
03
04 public class NotStorable {
05     private int id;
06
07     private String name;
08
09     private transient int length;
10
11     public NotStorable(int id, String name) {
12         this.id = id;
13         this.name = name;
14         this.length = name.length();
15     }
16
17     public int getId() {
18         return id;
19     }
20
21     public String getName() {
22         return name;
23     }
```

```

24 |
25 |     public int getLength() {
26 |         return length;
27 |     }
28 |
29 |     public String toString() {
30 |         return id + "/" + name + ":" + length;
31 |     }
32 |

```

We'll be using this code to store and retrieve an instance of this class with different configuration settings:

#### TranslatorExample.java: tryStoreAndRetrieve

```

01 |     private static void tryStoreAndRetrieve(Configuration configuration) {
02 |         ObjectContainer container = Db4o.openFile(configuration, DB40_FILE_NAME);
03 |         try {
04 |             NotStorable notStorable = new NotStorable(42, "Test");
05 |             System.out.println("ORIGINAL: " + notStorable);
06 |             container.set(notStorable);
07 |         } catch (Exception exc) {
08 |             System.out.println(exc.toString());
09 |             return;
10 |         } finally {
11 |             container.close();
12 |         }
13 |         container = Db4o.openFile(DB40_FILE_NAME);
14 |         try {
15 |             ObjectSet result = container.get(NotStorable.class);
16 |             while (result.hasNext()) {
17 |                 NotStorable notStorable = (NotStorable) result.next();
18 |                 System.out.println("RETRIEVED: " + notStorable);
19 |                 container.delete(notStorable);
20 |             }
21 |         } finally {

```

```

22 |         container.close();
23 |     }
24 | }
```

## Bypassing the constructor

TranslatorExample.java: tryStoreWithoutCallConstructors

```

1 [+]private static void tryStoreWithoutCallConstructors() {
2     Configuration configuration = Db4o.newConfiguration();
3     configuration.exceptionsOnNotStorable(false);
4     configuration.objectClass(NotStorable.class)
5         .callConstructor(false);
6     tryStoreAndRetrieve(configuration);
7 }
```

In this case our object seems to be nicely stored and retrieved, however, it has forgotten about its length, since db4o doesn't store transient members and the constructor code that sets it is not executed.

## Using the constructor

TranslatorExample.java: tryStoreWithCallConstructors

```

1 [+]private static void tryStoreWithCallConstructors() {
2     Configuration configuration = Db4o.newConfiguration();
3     configuration.exceptionsOnNotStorable(true);
4     configuration.objectClass(NotStorable.class)
5         .callConstructor(true);
6     tryStoreAndRetrieve(configuration);
7 }
```

At storage time, db4o tests the only available constructor with null arguments and runs into a NullPointerException, so it refuses to accept our object.

(Note that this test only occurs when configured with exceptionsOnNotStorable - otherwise db4o will silently fail when trying to reinstantiate the object.)

This still does not work for our case because the native pointer will definitely be invalid.

In order to solve the problem we will need to use [db4o Translators](#).

This revision (7) was last Modified 2007-08-04T20:00:34 by Tetyana.

# .NET Example Class

## Contents

- [Using The Constructor](#)
- [Bypassing The Constructor](#)

This topic applies to .NET version only.

For this example we'll be using a hypothetical `LocalizedItemList` class which binds together culture information with a list of items.

`System.Globalization.CultureInfo` is particularly interesting because it internally holds a native pointer to a system structure which in turn cannot be cleanly stored by db4o.

We'll be using this code to store and retrieve and instance of this class with different configuration settings:

## Using The Constructor

---

At storage time, db4o tests the only available constructor with null arguments and runs into a `NullPointerException`, so it refuses to accept our object.

(Note that this test only occurs when configured with [exceptionsOnNotStorable](#) - otherwise db4o will silently fail when trying to reinstantiate the object.)

## Bypassing The Constructor

---

This still does not work for our case because the native pointer will definitely be invalid. In fact this example crashes the Common Language Runtime.

In order to solve the problem we will need to use [db4o Translators](#).

This revision (3) was last Modified 2007-08-04T20:14:33 by Tetyana.

# The Translator API

## Contents

- [ObjectTranslator](#)
- [ObjectConstructor](#)

So how do we get our object into the database, now that everything seems to fail? Db4o provides a way to specify a custom way of storing and retrieving objects through the ObjectTranslator and ObjectConstructor interfaces.

## ObjectTranslator

---

The ObjectTranslator API looks like this:

Java:

```
public Object onStore(ObjectContainer container, Object applicationObject);

public void onActivate(ObjectContainer container, Object applicationObject,
Object storedObject);

public Class storedClass()
```

The usage is quite simple: When a translator is configured for a class, db4o will call its onStore method with a reference to the database and the instance to be stored as a parameter and will store the object returned. This object's type has to be primitive from a db4o point of view and it has to match the type specification returned by storedClass().

On retrieval, db4o will create a blank object of the target class (using the configured instantiation method) and then pass it on to onActivate() along with the stored object to be set up accordingly.

## ObjectConstructor

---

However, this will only work if the application object's class provides sufficient setters to recreate its state from the information contained in the stored object, which is not the case for our example class.

For these cases db4o provides an extension to the ObjectTranslator interface, ObjectConstructor, which declares one additional method:

Java:

```
public Object onInstantiate(ObjectContainer container, Object storedObject);
```

If db4o detects a configured translator to be an ObjectConstructor implementation, it will pass the stored class instance to the onInstantiate() method and use the result as a blank application object to be processed by onActivate().

Note that, while in general configured translators are applied to subclasses, too, ObjectConstructor application object instantiation will not be used for subclasses (which wouldn't make much sense, anyway), so ObjectConstructors have to be configured for the concrete classes.

[Java Translator Implementation](#) provides a Translator usage example for Java platform.

[.NET Translator Implementation](#) provides a Translator usage example for .NET platform.

This revision (9) was last Modified 2007-08-05T09:22:11 by Tetyana.

# ObjectConstructor

However, this will only work if the application object's class provides sufficient setters to recreate its state from the information contained in the stored object, which is not the case for our example class.

For these cases db4o provides an extension to the ObjectTranslator interface, ObjectConstructor, which declares one additional method:

Java:

```
public Object onInstantiate(ObjectContainer container, Object storedObject);
```

If db4o detects a configured translator to be an ObjectConstructor implementation, it will pass the stored class instance to the onInstantiate() method and use the result as a blank application object to be processed by onActivate().

Note that, while in general configured translators are applied to subclasses, too, ObjectConstructor application object instantiation will not be used for subclasses (which wouldn't make much sense, anyway), so ObjectConstructors have to be configured for the concrete classes.

This revision (3) was last Modified 2006-11-13T15:59:36 by Tetyana.

# .NET Translator Implementation

This topic applies to .NET version only.

To translate CultureInfo instances, we will store only their name since this is enough to recreate them later. Note that we don't have to do any work in onActivate(), since object reinstantiation is already fully completed in onInstantiate().

Let's try it out:

ObjectTranslators let you reconfigure the state of a 'blank' application object reinstantiated by db4o, ObjectConstructors also take care of instantiating the application object itself. ObjectTranslators and ObjectConstructors can be used for classes that cannot cleanly be stored and retrieved with db4o's standard object instantiation mechanisms.

This revision (6) was last Modified 2007-08-04T20:24:01 by Tetyana.

# Java Translator Implementation

This topic applies to Java version only

To translate NotStorable instances, we will pack their id and name values into an Object array to be stored and retrieve it from there again. Note that we don't have to do any work in onActivate(), since object reinstantiation is already fully completed in onInstantiate().

NotStorableTranslator.java

```
01 /* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */
02 package com.db4odoc.translators;
03
04 import com.db4o.*;
05 import com.db4o.config.*;
06
07 public class NotStorableTranslator implements ObjectConstructor {
08     public Object onStore(ObjectContainer container,
09     Object applicationObject) {
10         System.out.println("onStore for " + applicationObject);
11         NotStorable notStorable = (NotStorable) applicationObject;
12         return new Object[] { new Integer(notStorable.getId()),
13             notStorable.getName() };
14     }
15
16     public Object onInstantiate(ObjectContainer container,
17     Object storedObject) {
18         System.out.println("onInstantiate for " + storedObject);
19         Object[] raw = (Object[]) storedObject;
20         int id = ((Integer) raw[0]).intValue();
21         String name = (String) raw[1];
22         return new NotStorable(id, name);
23     }
24 }
```

```

25 |   public void onActivate(ObjectContainer container,
26 |     Object applicationObject, Object storedObject) {
27 |     System.out.println("onActivate for " + applicationObject
28 |       + " / " + storedObject);
29 |   }
30 |
31 |   public Class storedClass() {
32 |     return Object[].class;
33 |   }
34 |

```

Let's try it out:

TranslatorExample.java: storeWithTranslator

```

1 | private static void storeWithTranslator() {
2 |   Configuration configuration = Db4o.newConfiguration();
3 |   configuration.objectClass(NotStorable.class).translate(
4 |     new NotStorableTranslator());
5 |   tryStoreAndRetrieve(configuration);
6 | }

```

ObjectTranslators let you reconfigure the state of a 'blank' application object reinstantiated by db4o, ObjectConstructors also take care of instantiating the application object itself. ObjectTranslators and ObjectConstructors can be used for classes that cannot cleanly be stored and retrieved with db4o's standard object instantiation mechanisms.

This revision (7) was last Modified 2007-08-04T20:10:16 by Tetyana.

# Built-In Translators

Db4o supplies some build-in translators, which can be used in general cases. Most of them are used internally and are not a part of public API, however they can serve a good example of a translator implementation.

More Reading:

- [TNull Translator](#)
- [Collection Translators](#)
- [TSerializable Translator](#)
- [TTransient Translator](#)
- [TCultureInfo Translator](#)
- [TType Translator](#)
- [TClass Translator](#)

This revision (2) was last Modified 2007-08-13T15:57:52 by Tetyana.

# TNull Translator

TNull translator is used to notify db4o engine that the class data should not be stored. Db4o uses this translator internally for delegates.

Let's look at an example:

## NotStorable.java

```

01 /* Copyright (C) 2007 db4o objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.builtintranslators;
04
05 public class NotStorable {
06     String _name;
07
08     public NotStorable(String name) {
09         _name = name;
10    }
11
12     public String toString() {
13         return _name == null ? "null" : _name;
14    }
15 }
```

## BuiltInTranslatorsExample.java: saveNotStorable

```

01 public static void saveNotStorable()
02 {
03     new File(DB4O_FILE_NAME).delete();
04     Configuration configuration = Db4o.newConfiguration();
05     // Configure NotStorable class with TNull translator to prevent its storage
06     configuration.objectClass(NotStorable.class).translate(new TNull());
07     ObjectContainer container = database(configuration);
08     if (container != null)
09     {
10         try
11         {
```

```

12 |             NotStorable ns = new NotStorable("test1");
13 |             container.set(ns);
14 |             ns = new NotStorable("test2");
15 |             container.set(ns);
16 |
17 |         }
18 |     {
19 |         ex.printStackTrace();
20 |     }
21 |     catch (Exception ex)
22 |     {
23 |         ex.printStackTrace();
24 |     }
25 |     finally
26 |     {
27 |         closeDatabase();
28 |     }
29 |
30 | }

```

## BuiltInTranslatorsExample.java: testTNull

```

01 public static void testTNull()
02 {
03     saveNotStorable();
04     ObjectContainer container = database();
05     if (container != null)
06     {
07         try
08     {
09         // Trying to retrieve
10         ObjectSet result = container.query(NotStorable.class);
11         // As the class is configured with TNull, the data should be null
12         listResult(result);
13     }
14     catch (Db4oException ex)
15     {
16         ex.printStackTrace();

```

```
17      }
18      catch (Exception ex)
19      {
20          ex.printStackTrace();
21      }
22      finally
23      {
24          closeDatabase();
25      }
26  }
```

If you will run this example you will see that though the information about the NotStorable class was saved to the database, the retrieved value is null.

This revision (2) was last Modified 2007-08-13T15:59:38 by Tetyana.

# Collection Translators

In Java version TVector and THashtable classes represent collection translators. Internally they are used to store Vector and Hashtable classes accordingly.

The functionality of these translators is pretty straightforward:

- OnStore the members of the object are moved to an object array (Entry array for TDictionary and THashtable).
- OnActivate a new collection object is constructed from the array values.

In general you will never need to use any of these translators as all the work is done for you under the hood. However you might be interested to have a look at their code if you are planning a totally new custom collection implementation (not CollectionBase derived). The source code for collection translators can be found in Db4objects.Db4o.Config namespace/com.db4o.config package.

This revision (3) was last Modified 2007-08-13T17:28:46 by Tetyana.

# TSerializable Translator

TSerializable translator allows persistence of classes that do not have a constructor acceptable for db4o (For more information see [Translators](#)). Under the hood this translator converts an object to a memory stream on store and restores it upon instantiation. The limitations of this translator:

- if the stored type is refactored, the object value won't be retrievable from the database;
- the translator will be useless if querying for object fields is required (unless each object is fully instantiated before querying).

TSerializable translator should be used only with classes implementing java.io.Serializable interface (Java) or using [Serializable] attribute (.NET).

## Pilot.java

```

01 /* Copyright (C) 2007 db4objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.builtintranslators;
04
05 import java.io.Serializable;
06
07 public class Pilot implements Serializable {
08 |   public String _name;
09 |
10 |   public int _points;
11 |
12     public Pilot() {
13 |   }
14 |
15     public Pilot(String name, int points) {
16 |      _name = name;
17 |      _points = points;
18 |   }
19 |
20     public String getName() {
21 |      return _name;
22 |   }

```

```

23 |
24  public void setName(String name) {
25 |     _name = name;
26 | }
27 |
28  public int getPoints() {
29 |     return _points;
30 | }
31 |
32  public String toString() {
33 |     return _name + "/" + _points;
34 | }
35 |
36 }

```

## BuiltInTranslatorsExample.java: saveSerializable

```

01 public static void saveSerializable()
02 {
03     new File(DB4O_FILE_NAME).delete();
04     Configuration configuration = Db4o.newConfiguration();
05     // configure class as serializable
06     configuration.objectClass(Pilot.class).translate(new TSerializable());
07     ObjectContainer container = database(configuration);
08     if (container != null)
09     {
10         try
11         {
12             Pilot pilot = new Pilot("Test Pilot 1", 99);
13             container.set(pilot);
14             pilot = new Pilot("Test Pilot 2", 100);
15             container.set(pilot);
16         }
17         catch (Db4oException ex)
18         {

```

```

19 |         ex.printStackTrace();
20 |     }
21 |     catch (Exception ex)
22 |     {
23 |         ex.printStackTrace();
24 |     }
25 |     finally
26 |     {
27 |         closeDatabase();
28 |     }
29 | }
30 |
31 | }

```

## BuiltInTranslatorsExample.java: testSerializable

```

01 public static void testSerializable()
02 {
03     saveSerializable();
04     Configuration configuration = Db4o.newConfiguration();
05     // configure class as serializable to retrieve correctly
06     configuration.objectClass(Pilot.class).translate(new TSerializable());
07     ObjectContainer container = database(configuration);
08     if (container != null)
09     {
10         try
11         {
12             System.out.println("Retrieving pilots by name:");
13             Query query = container.query();
14             query.constraint(Pilot.class);
15             query.descend("_name").constraint("Test Pilot 1");
16             ObjectSet resultByName = query.execute();
17             listResult(resultByName);
18
19             System.out.println("Retrieve all pilot objects:");

```

```
20 |         ObjectSet result = container.query(Pilot.class);
21 |         listResult(result);
22 |     }
23 |     catch (Db4oException ex)
24 |     {
25 |         ex.printStackTrace();
26 |     }
27 |     catch (Exception ex)
28 |     {
29 |         ex.printStackTrace();
30 |     }
31 |     finally
32 |     {
33 |         closeDatabase();
34 |     }
35 | }
36 | }
```

This revision (3) was last Modified 2007-08-13T17:32:15 by Tetyana.

# TTransient Translator

TTransient translator should be used with the objects that must not be stored to db4o. These can be classes holding internal references to the runtime or system.

A simple Java example can look as follows.

BuiltInTranslatorsExample.java: saveTransient

```
01 public static void saveTransient()
02 {  
03     new File(DB4O_FILE_NAME).delete();  
04     Configuration configuration = Db4o.newConfiguration();  
05     // Configure NotStorable class with TTransient translator to prevent its storage  
06     configuration.objectClass(NotStorable.class).translate(new TTransient());  
07     ObjectContainer container = database(configuration);  
08     if (container != null)  
09     {  
10         try  
11         {  
12             Item item = new Item("Test1", new NotStorable("test1"));  
13             container.set(item);  
14             item = new Item("Test2", new NotStorable("test2"));  
15             container.set(item);  
16         }  
17         catch (Db4oException ex)  
18         {  
19             ex.printStackTrace();  
20         }  
21         catch (Exception ex)  
22         {  
23             ex.printStackTrace();  
24         }  
25         finally  
26         {  
27             closeDatabase();  
28         }  
29     }  
30 }
```

## BuiltInTranslatorsExample.java: testTransient

```

01 public static void testTransient()
02 {
03     saveTransient();
04     ObjectContainer container = database();
05     if (container != null)
06     {
07         try
08         {
09             // Trying to retrieve
10            ObjectSet result = container.query(Item.class);
11            // NotStorable class is configured transient, so we will
12            // see null values for _ns field.
13            listResult(result);
14        }
15        catch (Db4oException ex)
16        {
17            ex.printStackTrace();
18        }
19        catch (Exception ex)
20        {
21            ex.printStackTrace();
22        }
23        finally
24        {
25            closeDatabase();
26        }
27    }
28 }

```

## Item.java

```

01 /* Copyright (C) 2007 db4objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.builtintranslators;
04
05
06 public class Item {

```

[edit]

```

07 |   String _name;
08 |   NotStorable _ns;
09 |
10 [|] public Item(String name, NotStorable ns) {
11 |     _name = name;
12 |     _ns = ns;
13 | }
14 |
15 [|] public String toString() {
16 |     return _name + "/" + (_ns == null ? "null" : _ns);
17 | }
18 [L]

```

## NotStorable.java

```

01 [|]/* Copyright (C) 2007 db4o objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.built_intranslators;
04
05 [|]public class NotStorable {
06 |   String _name;
07 |
08 [|]   public NotStorable(String name) {
09 |     _name = name;
10 |   }
11 |
12 [|]   public String toString() {
13 |     return _name == null ? "null" : _name;
14 |   }
15 [L]

```

Item class contains NotStorable field. NotStorable object is translated with TTransient translator, which prevents its instances from being stored.

This revision (3) was last Modified 2007-08-13T18:33:39 by Tetyana.

# TCultureInfo Translator

This topic applies to .NET version only.

CultureInfo class cannot be stored cleanly by db4o because it internally holds a native pointer to a system structure. This problem was discussed in detail in [.NET Example Class](#) and a solution suggested in [.NET Translator Implementation](#). In fact built-in TCultureInfo class implements the same solution and can be used immediately without any custom translators:

This revision (2) was last Modified 2007-08-05T14:58:59 by Tetyana.

# TType Translator

This topic applies to .NET version only.

TType translator is used internally to save System.Type objects. Under the hood the object is translated to its string representation when stored, and restored to a Type object from the string when instantiated.

This revision (1) was last Modified 2007-08-05T14:55:49 by Tetyana.

# TClass Translator

This topic applies to Java version only.

TType translator is used internally to save java.lang.Class objects. Under the hood the object is translated to its string representation when stored, and restored to a Class object from the string when instantiated.

This revision (1) was last Modified 2007-08-13T18:29:01 by Tetyana.

# Db4o Reflection API

Reflection API gives your code access to internal information for classes loaded into the VM(Java) / common runtime (.NET). It allows you to work with classes defined in runtime and not in code.

Reflection works with metadata - data that describes other data. In the case of reflection metadata is the description of classes and objects within the JVM or .NET assembly, including their fields, methods and constructors. It allows the programmer to select target classes in runtime, create new objects, call their methods and operate with the fields.

These features make reflection especially useful for creating libraries that work with objects in very general ways. For example, reflection is often used in frameworks that persist objects to databases, XML, or other external formats.

More Reading:

- [GenericReflector](#)
- [Using db4o reflection API](#)
- [Creating your own reflector](#)

This revision (5) was last Modified 2006-11-14T18:56:52 by Eric Falsken.

# GenericReflector

Db4o uses reflection internally for persisting and instantiating user objects. Reflection helps db4o to manage classes in a general way while saving. It also makes possible instantiation of user objects using class name saved in the database file and class information from the JVM or .NET assembly. However db4o reflection API can also work on generic objects when a class information is not available.

Db4o uses GenericReflector as a wrapper around specific reflector (delegate). GenericReflector is set when an ObjectContainer is opened. All subsequent reflector calls are routed through this interface.

GenericReflector keeps list of known classes in memory. When the GenericReflector is called, it first checks its list of known classes. If the class cannot be found, the task is transferred to the delegate reflector. If the delegate fails as well, generic objects are created, which hold simulated "field values" in an array of objects.

Generic reflector makes possible the following usecases:

- running a db4o server without deploying application classes;
- running db4o on Java dialects without reflection (J2ME CLDC, MIDP);
- easier access to stored objects where classes or fields are not available;
- running refactorings in the reflector;
- building interfaces to db4o from any programming language.

One of the live usecases is ObjectManager, which uses GenericReflector to read C# objects from Java.

This revision (4) was last Modified 2006-11-13T19:16:02 by Tetyana.

# Using db4o reflection API

Db4o reflector can be used in your application just like normal java reflector. Let's create a new database with a couple of cars in it:

ReflectorExample.java: setCars

```

01 private static void setCars()
02 {  

03     new File(DB4O_FILE_NAME).delete();  

04     ObjectContainer container=Db4o.openFile(DB4O_FILE_NAME);  

05     try {  

06         Car car1 = new Car("BMW");  

07         container.set(car1);  

08         Car car2 = new Car("Ferrari");  

09         container.set(car2);  

10  

11         System.out.println("Saved: ");  

12         Query query = container.query();  

13         query.constraint(Car.class);  

14         ObjectSet results = query.execute();  

15         listResults(results);  

16     } finally {  

17         container.close();  

18     }  

19 }
```

We can check, what information is available for db4o reflector:

ReflectorExample.java: getReflectorInfo

```

01 private static void getReflectorInfo()
02 {  

03     ObjectContainer container=Db4o.openFile(DB4O_FILE_NAME);  

04     try {  

05         System.out.println("Reflector in use: " + container.ext().reflector());  

06         System.out.println("Reflector delegate" +container.ext().reflector().getDelegate());  

07         ReflectClass[] knownClasses = container.ext().reflector().knownClasses();  

08         int count = knownClasses.length;  

09         System.out.println("Known classes: " + count);  

10     } finally {
```

```

11 |         System.out.println(knownClasses[i]);
12 |     }
13 | } finally {
14 |     container.close();
15 | }
16 | }
```

All the information about Car class can also be retrieved through reflector:

ReflectorExample.java: getCarInfo

```

01 private static void getCarInfo()
02 { ObjectContainer container=Db4o.openFile(DB4O_FILE_NAME);
03 try {
04     GenericReflector reflector = new GenericReflector(null, container.ext().reflector());
05     ReflectClass carClass = reflector.forName(Car.class.getName());
06     System.out.println("Reflected class "+carClass);
07     // public fields
08     System.out.println("FIELDS: ");
09     ReflectField[] fields = carClass.getDeclaredFields();
10    for (int i = 0; i < fields.length; i++)
11        System.out.println(fields[i].getName());
12
13
14     // constructors
15     System.out.println("CONSTRUCTORS: ");
16     ReflectConstructor[] cons = carClass.getDeclaredConstructors();
17     for (int i = 0; i < cons.length; i++)
18         System.out.println(cons[i]);
19
20     // public methods
21     System.out.println("METHODS: ");
22     ReflectMethod method = carClass.getMethod("getPilot", null);
23     System.out.println(method.getDeclaringClass());
24
25 } finally {
26     container.close();
27 }
28 }
```

This revision (7) was last Modified 2006-11-13T19:19:58 by Tetyana.

# Creating your own reflector

By default db4o uses JdkReflector(Java) or NetReflector (.NET) as a GenericReflector delegate.

However, the programmer can instruct db4o to use a specially designed reflection implementation:

Java: Db4o.configure().reflectWith(reflector)

where reflector is one of the available reflectors or your own reflector implementation.

At present db4o comes with SelfReflector, which was designed for environments, which do not have built-in support for reflections (J2ME for example). In this implementation all the classes' information is stored in special registry. User classes should implement self\_get and self\_set methods to be registered individually and become "known" to SelfReflector.

Specific reflectors can be written for special usecases.

Let's look how to create a reflector. Remember that db4o relies on reflector to read the database, so errors in reflector may prevent your database from opening.

To keep things simple we will write a LoggingReflector, its only difference from standard reflector is that information about loaded classes is outputted to console. All reflectors used by db4o should implement com.db4o.reflect.Reflector interface.

## LoggingReflector.java

```

01 /* Copyright (C) 2004 - 2005 db4o objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.reflections;
04
05 import com.db4o.reflect.ReflectArray;
06 import com.db4o.reflect.ReflectClass;
07 import com.db4o.reflect.Reflector;
08 import com.db4o.reflect.jdk.JdkClass;
09
10 public class LoggingReflector implements Reflector {
11     private ReflectArray _array;
12 }
```

```
13 |     private Reflector _parent;
14 |
15 |     public LoggingReflector() {
16 |
17 |     }
18 |
19 |     public ReflectArray array() {
20 |         if (_array == null) {
21 |             _array = new LoggingArray(_parent);
22 |         }
23 |         return _array;
24 |     }
25 |
26 |     public boolean constructorCallsSupported() {
27 |         return true;
28 |     }
29 |
30 |     public ReflectClass forClass(Class clazz) {
31 |         ReflectClass rc = new JdkClass(_parent, clazz);
32 |         System.out.println("forClass: " + clazz + " -> "
33 |             + (rc == null ? "" : rc.getName()));
34 |         return rc;
35 |     }
36 |
37 |     public ReflectClass forName(String className) {
38 |         try {
39 |             Class clazz = Class.forName(className);
40 |             ReflectClass rc = forClass(clazz);
41 |             System.out.println("forName: " + className + " -> "
42 |                 + (rc == null ? "" : rc.getName()));
43 |             return rc;
44 |         } catch (ClassNotFoundException e) {
45 |             return null;
```

```
46 |     }
47 | }
48 |
49 |     public ReflectClass forObject(Object a_object) {
50 |         if (a_object == null) {
51 |             return null;
52 |         }
53 |         ReflectClass rc = _parent.forClass(a_object.getClass());
54 |         System.out.println("forObject: " + a_object + " -> "
55 |             + (rc == null ? "" : rc.getName()));
56 |         return rc;
57 |     }
58 |
59 |     public boolean isCollection(ReflectClass clazz) {
60 |         return false;
61 |     }
62 |
63 |     public void setParent(Reflector reflector) {
64 |         _parent = reflector;
65 |     }
66 |
67 |     public Object deepClone(Object context) {
68 |         return new LoggingReflector();
69 |     }
70 }
```

The output can help you to track all the loaded classes.

Reflection is a powerful tool, which plays a fundamental role in db4o. Understanding reflection will help you to understand the whole db4o functionality in detail.

This revision (10) was last Modified 2006-11-13T19:24:37 by Tetyana.

# IO Adapter

IO system plays an important role for any system working with persistent objects. Db4o allows you to use your own IO system implementation giving you additional power over object storage and retrieval.

More Reading:

- [Pluggable IO Concept](#)
- [MemoryIoAdapter](#)
- [CachedIoAdapter](#)

This revision (4) was last Modified 2006-11-15T11:31:50 by Tetyana.

# Pluggable IO Concept

Db4o input/output implementation comes as a pluggable module. It means that you can write your own IO system based on IoAdapter class.

Pluggable IO Adapter has control over the following:

- place and method of storing a database file;
- the form of the stored file (plain, encrypted);
- additional actions that are taken at read,write,close,open events.

Some of the popular implementations of IoAdapter:

- encryption IoAdapter: database stream is encrypted-decrypted upon read-write operations;
- native IoAdapter: uses native system API for file access (WinAPI for example);
- memory IoAdapter: keeps database in RAM instead of hard drive;
- IoAdapter using several physical locations to store a database;
- debug IoAdapter: outputs various debugging information to the specified output stream.

You can look at the db4o source to see some of these implementations (search for classes inheriting from IoAdapter).

By default db4o uses RandomAccessFileAdapter. You can install another IoAdapter with a single call:

```
Java: Db4o.configure().io(new SpecificIoAdapter())
```

Please, note: this call should be issued before opening of a database file. IoAdapter can be changed only while the database is closed.

For a test, let's try to create debugging IoAdapter, which will log all the information about its actions to the output stream:

**LoggingAdapter.java**

```
001  /* Copyright (C) 2004 - 2007 db4objects Inc. http://www.db4o.com */  
002  
003 package com.db4odoc.ios;  
004  
005 import java.io.File;  
006 import java.io.IOException;  
007 import java.io.PrintStream;  
008 import java.io.RandomAccessFile;  
009  
010 import com.db4o.DTrace;  
011 import com.db4o.ext.DatabaseFileLockedException;  
012 import com.db4o.ext.Db4oIOException;  
013 import com.db4o.internal.Platform4;  
014 import com.db4o.io.IoAdapter;  
015
```

```

016  public class LoggingAdapter extends IoAdapter {
017 |
018 |   @Override
019 |     public IoAdapter open(String path, boolean lockFile, long initialLength, boolean
read0nly) throws Db4oIOException {
020 |       _out.println("Opening file " + path);
021 |       return new LoggingAdapter(path, lockFile, initialLength, read0nly);
022 |   }
023 |
024 |
025 |   private String _path;
026 |   private RandomAccessFile _delegate;
027 |   private PrintStream _out = System.out;
028 |
029 |     public LoggingAdapter() {
030 |   }
031 |
032 |     protected LoggingAdapter(String path, boolean lockFile, long initialLength, boolean
read0nly) throws Db4oIOException {
033 |       try {
034 |         String mode = read0nly ? "r" : "rw";
035 |         _path=new File(path).getCanonicalPath();
036 |         _delegate = new RandomAccessFile(path, mode);
037 |         if(initialLength>0) {
038 |           _delegate.seek(initialLength - 1);
039 |           _delegate.write(new byte[] {0});
040 |         }
041 |         if(lockFile){
042 |           try {
043 |             Platform4.lockFile(_path, _delegate);
044 |           } catch (DatabaseFileLockedException e) {
045 |             _delegate.close();
046 |             throw e;
047 |           }
048 |         }
049 |       } catch (IOException e) {
050 |         throw new Db4oIOException(e);
051 |       }
052 |     }
053 |
054 |     public void setOut(PrintStream out){

```

```
055 |         _out = out;
056 |     }
057 |
058 |     public void close() throws Db4oIOException {
059 |         _out.println("Closing file");
060 |         try {
061 |             Platform4.unlockFile(_path, _delegate);
062 |             _delegate.close();
063 |         } catch (IOException e) {
064 |             throw new Db4oIOException(e);
065 |         } catch (Exception e) {
066 |         }
067 |     }
068 |
069 |
070 |     public void delete(String path) {
071 |         _out.println("Deleting file " + path);
072 |         new File(path).delete();
073 |     }
074 |
075 |     public boolean exists(String path) {
076 |         File existingFile = new File(path);
077 |         return existingFile.exists() && existingFile.length() > 0;
078 |     }
079 |
080 |     public long getLength() throws Db4oIOException {
081 |         long length;
082 |         try {
083 |             _out.println("File length: " + _delegate.length());
084 |             length = _delegate.length();
085 |         } catch (IOException e) {
086 |             throw new Db4oIOException(e);
087 |         }
088 |         return length;
089 |     }
090 |
091 |     public int read(byte[] bytes, int length) throws Db4oIOException {
092 |         int readBytes;
093 |         try {
094 |             _out.println("Reading " + length + " bytes");
095 |             readBytes = _delegate.read(bytes, 0, length);
```

```
096 } catch (I0Exception e) {  
097     throw new Db4oIOException(e);  
098 }  
099 return readBytes;  
100 }  
101  
102  
103 public void seek(long pos) throws Db4oIOException {  
104  
105     if (DTrace.enabled){  
106         DTrace.REGULAR_SEEK.log(pos);  
107     }  
108     try {  
109         _out.println("Setting pointer position to " + pos);  
110         _delegate.seek(pos);  
111     } catch (I0Exception e) {  
112         throw new Db4oIOException(e);  
113     }  
114 }  
115  
116  
117 public void sync() throws Db4oIOException {  
118     _out.println("Synchronizing");  
119     try {  
120         _delegate.getFD().sync();  
121     } catch (I0Exception e) {  
122         throw new Db4oIOException(e);  
123     }  
124 }  
125  
126  
127 public void write(byte[] buffer, int length) throws Db4oIOException {  
128     _out.println("Writing " + length + " bytes");  
129     try {  
130         _delegate.write(buffer, 0, length);  
131     } catch (I0Exception e) {  
132         new Db4oIOException(e);  
133     }  
134 }  
135 }  
136 }
```

137 }

Now let's test how it works:

IOExample.java: testLoggingAdapter

```

01  private static void testLoggingAdapter(){
02 |     Configuration configuration = Db4o.newConfiguration();
03 |     configuration.io(new LoggingAdapter());
04 |     ObjectServer server = Db4o.openServer(configuration, DB40_FILE_NAME, 0xdb40);
05 |     server.grantAccess("y", "y");
06 |     ObjectContainer container = server.openClient();
07 |     //ObjectContainer container = Db4o.openClient(configuration, "localhost", 0xdb40, "y",
08 |     //                                         "y");
08  |     try {
09 |         Pilot pilot = new Pilot("Michael Schumacher");
10 |         container.set(pilot);
11 |         System.out.println("New pilot added");
12 |     } finally {
13 |
14 |         container.close();
15 |         server.close();
16 |     }
17 |
18 |     container = Db4o.openFile(DB40_FILE_NAME);
19  |     try {
20 |         ObjectSet result=container.get(Pilot.class);
21 |         listResult(result);
22 |     } finally {
23 |         container.close();
24 |     }
25 | }
```

The output can be used for debugging purposes.

This revision (8) was last Modified 2006-11-13T15:15:09 by Tetyana.

# MemoryIoAdapter

One of the built-in db4o IoAdapters is MemoryIoAdapter. This adapter keeps the whole database in RAM instead of a hard-drive. Obviously such in-memory database has both advantages and disadvantages:

Pros:

- it is a lot faster, as no disc access is necessary.

Cons:

- it only works, if the database fits into RAM;
- all the data can be lost in case of a system failure.

You can achieve the best results combining the usage of MemoryIoAdapter with the normal file adapter. Let's look, how MemoryIoAdapter should be used in your application.

First of all MemoryIoAdapter should be configured. The API gives you control over memory consumption with

Java: `MemoryIoAdapter#growBy(length)`

method. Length defines the amount of bytes in memory that should be allocated, when no more free slots are found within the allocated file. Large value (100,000) will assure the best performance; small value (100) will keep the memory consumption at the lowest level. The default setting is 10,000.

MemoryIoAdapter supplies methods to exchange data with the database from the filesystem.

Java: `MemoryIoAdapter#put(filename, byte[])`

This method allows you to put the contents of the stored database file (`byte[]`) into the memory database.

The opposite method:

Java: `MemoryIoAdapter#get(filename)`

returns the updated `byte[]` array of the database in memory, which can be saved to disc as a normal database file.

The filename can be used for all further access (`#openFile`, `#openServer` methods) like normal file name. Make sure that you close the database file on the disc after reading its contents into memory and writing it back.

The configured adapter should be assigned as db4o IO system:

Java: Db4o.configure().io(MemoryIoAdapter)

[/filter]

[filter=cs]

C#: Db4o.Configure().Io(MemoryIoAdapter)

Let's create sample database on the disc:

IOExample.java: setObjects

```

01  private static void setObjects(){
02 |     new File(DB4O_FILE_NAME).delete();
03 |     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
04  |     try {
05 |         Pilot pilot = new Pilot("Rubens Barrichello");
06 |         container.set(pilot);
07  |     } finally {
08 |         container.close();
09 |     }
10  }
```

Now we can read the database into the memory file and work in the memory using MemoryIoAdapter:

IOExample.java: getObjectsInMem

```

01  private static void getObjectsInMem() {
02 |     System.out.println("Setting up in-memory database");
03 |     MemoryIoAdapter adapter = new MemoryIoAdapter();
04  |     try {
05 |         RandomAccessFile raf = new RandomAccessFile(DB4O_FILE_NAME, "r");
06 |         adapter.growBy(100);
07 |
08 |         int len = (int) raf.length();
09 |         byte[] b = new byte[len];
10 |         raf.read(b, 0, len);
11 |         adapter.put(DB4O_FILE_NAME, b);
```

```

12 |         raf. close();
13 |     } catch (Exception ex) {
14 |         System.out.println("Exception: " + ex.getMessage());
15 |     }
16 |
17 |     Configuration configuration = Db4o.newConfiguration();
18 |     configuration.io(adapter);
19 |     ObjectContainer container = Db4o.openFile(configuration, DB4O_FILE_NAME);
20 |     try {
21 |         ObjectSet result = container.get(Pilot.class);
22 |         System.out.println("Read stored results through memory file");
23 |         listResult(result);
24 |         Pilot pilotNew = new Pilot("Michael Schumacher");
25 |         container.set(pilotNew);
26 |         System.out.println("New pilot added");
27 |     } finally {
28 |         container.close();
29 |     }
30 |     System.out.println("Writing the database back to disc");
31 |     byte[] dbstream = adapter.get(DB4O_FILE_NAME);
32 |     try {
33 |         RandomAccessFile file = new RandomAccessFile(DB4O_FILE_NAME, "rw");
34 |         file.write(dbstream);
35 |         file.close();
36 |     } catch (IOException ioex) {
37 |         System.out.println("Exception: " + ioex.getMessage());
38 |     }
39 | }

```

Well, the changes are made and written to the disc. Let's check, if all the data are in place. We will need to switch back to the normal RandomAccessFileAdapter:

## IOExample.java: getObjects

```

01 |     private static void getObjects() {
02 |         Configuration configuration = Db4o.newConfiguration();

```

```
03 |     configuration.io(new RandomAccessFileAdapter());  
04 |     ObjectContainer container = Db4o.openFile(configuration, DB4O_FILE_NAME);  
05 |     try {  
06 |         ObjectSet result=container.get(Pilot.class);  
07 |         System.out.println("Read stored results through disc file");  
08 |         listResult(result);  
09 |     } finally {  
10 |         container.close();  
11 |     }  
12 }
```

So, switching between disc and memory database is quite easy. It can be used to increase performance on certain operations. More flexibility can be reached writing your own IoAdapter implementation.

This revision (12) was last Modified 2006-11-13T14:51:09 by Tetyana.

# CachedIoAdapter

CachedIoAdapter is used to cache data for read/write operations on db4o database file. Its functionality is similar to the operating system disk cache.

CachedIoAdapter uses the default RandomAccessFileAdapter as a delegate adapter and creates pages in RAM to cache the data before physical write/read to the disk file.

CachedIoAdapter can significantly increase the performance for bulk operations and in cases when the OS does not provide disk caching.

More Reading:

- [Using CachedIoAdapter](#)
- [CachedIoAdapter Example](#)

This revision (1) was last Modified 2007-02-01T15:28:50 by Tetyana.

# Using CachedIoAdapter

CachedIoAdapter can be set up using the following code:

Java:

```
delegateAdapter = new RandomAccessFileAdapter();  
  
Db4o.configure().io(new CachedIoAdapter(delegateAdapter));
```

This will create 64 pages in memory of 1024 bytes each, resulting in 64 kB of memory available for caching.

You can customize the amount of available memory using:

Java:

```
Db4o.configure().io(new CachedIoAdapter(delegateAdapter, page_size,  
page_count));
```

Optimal cache size depends on many factors (application design, system resources etc) and will show different results for different disc access operations (read/write). The best way to find an optimal solution is to experiment with different page sizes and counts. Too little cache memory will result in decrease of performance, too much of it will unnecessary occupy your system resources. Note that all the caching pages are allocated in the memory on CachedIoAdapter creation and are not released while CachedIoAdapter is in use. Effectively, you can only release this memory closing the database file and switching back to the default RandomAccessFileAdapter.

This revision (4) was last Modified 2007-02-01T20:28:19 by Tetyana.

# CachedIoAdapter Example

Let's look on an example at the benefits of CachedIoAdapter.

We will use the following methods to initiate write and read from a database file:

CachedIOExample.java: setObjects

```

01  public static void setObjects(){
02 |     new File(YAPFILENAME).delete();
03 |     ObjectContainer db = Db4o.openFile(YAPFILENAME);
04  try {
05 |     long t1 = System.currentTimeMillis();
06  for (int i = 0; i < 50000; i++) {
07 |         Pilot pilot = new Pilot("Pilot #" + i);
08 |         db.set(pilot);
09 |
10 |     long t2 = System.currentTimeMillis();
11 |     long timeElapsed = t2 - t1;
12 |     System.out.println("Time elapsed for setting objects =" + timeElapsed + " ms");
13 |     t1 = System.currentTimeMillis();
14 |     db.commit();
15 |     t2 = System.currentTimeMillis();
16 |     timeElapsed = t2 - t1;
17 |     System.out.println("Time elapsed for commit =" + timeElapsed + " ms");
18  } finally {
19 |     db.close();
20 }
21 }
```

CachedIOExample.java: getObjects

```

01  public static void getObjects(){
02 |
03 |     ObjectContainer db = Db4o.openFile(YAPFILENAME);
04  try {
05 |     long t1 = System.currentTimeMillis();
06 |     ObjectSet result = db.get(null);
07 |     long t2 = System.currentTimeMillis();
08 |     long timeElapsed = t2 - t1;
09 |     System.out.println("Time elapsed for the query =" + timeElapsed + " ms");
```

```
10 |     System.out.println("Objects in the database: " + result.size());  
11 | } finally {  
12 |     db.close();  
13 | }  
14 | }
```

Try to execute the code with the default settings and write down the results. Then configure CachedIoAdapter using the code below and test the performance again:

CachedIOExample.java: configureCache

```
1 public static void configureCache(){  
2     System.out.println("Setting up cached io adapter");  
3     // new cached IO adapter with 256 pages 1024 bytes each  
4     CachedIoAdapter adapter = new CachedIoAdapter(new RandomAccessFileAdapter(), 1024, 256);  
5     Db4o.configure().io(adapter);  
6 }
```

The performance delta will be more significant for more objects and bigger cache memory.

This revision (8) was last Modified 2007-02-01T20:46:04 by Tetyana.

# Db4o meta-information

Db4o meta information API provides an access to the actual structure of db4o database file. Its primary use is [refactoring](#).

More Reading:

- [Accessing db4o meta-information](#)
- [StoredClass and StoredField interfaces](#)

This revision (7) was last Modified 2007-01-01T09:30:07 by Tetyana.

# Accessing db4o meta-information

Db4o provides an access to the database meta-information through its extended object container interface (ExtObjectContainer(Java)/IExtObjectContainer(.NET)).

Within the object database meta-schema is represented by classes and their fields. To access their meta-information db4o provides special interfaces:

- StoredClass(Java)/IStoredClass(.NET)
- StoredField(Java)/IStoredField(.NET)

The following ExtObjectContainer methods give you access to the StoredClass.

Java: ExtObjectContainer#storedClass(Foo.class)

returns StoredClass for the specified clazz, which can be specified as:

- a fully qualified classname;
- a Class/Type object;
- any object to be used as a template.

Java: ExtObjectContainer#storedClasses()

returns an array of all StoredClass meta-information objects.

**MetalInfoExample.java: setObjects**

```

01 ①private static void setObjects() {
02 |     new File(DB4O_FILE_NAME).delete();
03 |     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
04 ②try {
05 |     Car car = new Car("BMW", new Pilot("Rubens Barrichello"));
06 |     container.set(car);
07 |     car = new Car("Ferrari", new Pilot("Michael Schumacher"));
08 |     container.set(car);
09 ③} finally {
10 |     container.close();
11 | }
12 }
```

## MetalInfoExample.java: getMetaObjects

```
01 ①②private static void getMetaObjects() {  
02 |     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);  
03 ③④    try {  
04 |         System.out  
05 |             .println("Retrieve meta information for class: ");  
06 |         StoredClass sc = container.ext().storedClass(  
07 |             Car.class.getName());  
08 |         System.out.println("Stored class: " + sc.toString());  
09 |  
10 |         System.out  
11 |             .println("Retrieve meta information for all classes in database: ");  
12 |         StoredClass sclasses[] = container.ext().storedClasses();  
13 ⑤⑥        for (int i = 0; i < sclasses.length; i++) {  
14 |            System.out.println(sclasses[i].getName());  
15 |        }  
16 ⑦⑧    } finally {  
17 |        container.close();  
18 |    }  
19 }
```

This revision (9) was last Modified 2007-12-26T12:59:04 by Tetyana.

# StoredClass and StoredField interfaces

(IStoredClass and IStoredField in .NET)

Let's look closer at the class meta-information interfaces.

They look quite similar to reflection API, but unlike reflection there is no information about methods and constructors.

You can only use StoredClass to get the class's fields:

Java: `StoredClass#getStoredFields()`

returns all stored fields of this stored class.

Java: `StoredClass#storedField(name, type)`

returns an existing stored field of this stored class.

You can also use this interface to explore classes hierarchy.

Java: `StoredClass#getParentStoredClass`

returns the parent of the class.

StoredField interface gives you access to various meta-field information, such as field name, field type. It also provides some helpful methods for manipulating fields accepting their object as a variable (see db4o API for more information).

Code attachment not found: /Resources/Reference/Implementation\_Strategies/Db4o\_Meta-Information/MetainfJava.Zip

This revision (9) was last Modified 2007-12-26T13:08:22 by Tetyana.

# LINQ Collection

## Contents

- [Simple Select](#)
- [Database Object Clone](#)
- [Select Any Type](#)
- [Join Query](#)
- [Sorting](#)
- [Grouping Results](#)
- [Modified Result](#)
- [Selecting A Result](#)
- [Aggregate Functions](#)

This topic applies to .NET version only

This chapter provides a collection of LINQ examples with db4o, which you can use to find different ways to construct your queries. The information below covers most of the database querying operations and should be enough for the majority of cases. However, it is still advisable to study [MSDN](#) resources, which explain in detail how LINQ and Extension methods work. For the introduction to LINQ queries for db4o please refer to [LINQ](#) chapter.

## Simple Select

---

Let's fill up the database with some data:

In order to select objects of only Pilot class we can issue the following query:

Note, that where clause is optional: if it is not specified all objects of Pilot class will be returned.

We can make querying even easier by using anonymous types:

## Database Object Clone

---

You can easily use LINQ to create database objects clones:

Retrieved objects are not bound to the object container, but they duplicate the values from the database. You can use them for an "object-readonly" mode.

## Select Any Type

---

With the selection we are not bound to only one type of objects - actually we can select everything that is currently in the database:

We can further use this broad selection too. As the query result implements `IQueryable` interface, we can re-use it to retrieve more specific objects:

In this example we use an all objects selection to find a range of pilots and then cars with pilots within the range. Remember, however, that the objects in `allObjects` variable are not actually retrieved from the database until they are browsed or used. If you want to get them all into the memory for future use, utilize `ToList` or `ToArray` methods.

## Join Query

---

Above we've already discussed how to select objects of one type based on the selection of the other type. However, that way is rather cumbersome and most probably you won't need it: you can use join operator instead:

This simple syntax allows you to join any amount of classes using any possible relationship between them. You can also make use of [Join](#) extension method syntax to get the same result.

## Sorting

---

Sorting is added with `orderby` operator and can be ascending(default) or descending.

We can also use subsequent sorts. For example sort by name descending and by points ascending:

The same effect can be achieved by using `OrderBy`/`OrderByDescending` and `ThenBy`/`ThenByDescending` extension methods.

## Grouping Results

---

Grouping results can be often useful in different reports. For example, we can group the data from the previous example by pilot's name:

Here we use `GroupBy` extension method. The parameters specify the grouping property (`Points`) and the display property (`Name`).

## Modified Result

---

Sometimes we need to apply some calculation on each result - with LINQ we can do that directly in the

query:

In the example above the query will bring as the calculated percentage of maximum for each pilot's points.

## Selecting A Result

---

Though a query usually returns all the candidates matching the selected criteria, we can specify which of the results we want: First, ElementAt, Any, All.

The query above will return true if there are any objects in the result set and false otherwise.

## Aggregate Functions

---

Often we are not interested in each and every result, but need some statistics about it: sum, aggregate, average, max etc. This can be achieved with extension methods.

This method returns only the average value of pilot points.

In this case we use Aggregate extension method to return all the names and a semicolon-separated string. In Aggregate function first parameter specify the initial return value, second parameter is a function that appends each new value to the initial value.

This revision (3) was last Modified 2008-02-25T16:06:20 by Tetyana.

# Native Query Collection

This collection of Native Queries examples is intended to show different implementations of the querying interface and help users with the practical use cases.

Persistent classes used in the examples are defined in [Persistent Classes](#).

More Reading:

- [Simple Selection](#)
- [Using Selection Criterias](#)
- [Selecting Ranges](#)
- [Sorting](#)
- [Result Representation](#)
- [Calculation Examples](#)
- [Combined Result Sets](#)
- [Parameterized NQ](#)
- [Store Pilots](#)
- [Store Persons](#)
- [Store Cars](#)
- [Persistent Classes](#)

This revision (2) was last Modified 2007-04-23T18:51:13 by Tetyana.

# Simple Selection

## Contents

- [SelectAllPilots](#)
- [SelectAllPilotsNonGeneric](#)

The following examples show how to use NQ to select all objects of the specified type from a database. [Store Pilots](#) function is used to fill in the database.

## SelectAllPilots

For languages with generics support (Java5-6; .NET2.0-3.0):

### SimpleExamples.java: selectAllPilots

```

01 ①②private static void selectAllPilots() {
02 |    ObjectContainer container = database();
03 |    if (container != null) {
04 |        try {
05 |            List<Pilot> result = container.query(new Predicate<Pilot>() {
06 |                public boolean match(Pilot pilot) {
07 |                    // each Pilot is included in the result
08 |                    return true;
09 |                }
10 |            });
11 |            listResult(result);
12 |        } catch (Exception ex) {
13 |            System.out.println("System Exception: " + ex.getMessage());
14 |        } finally {
15 |            closeDatabase();
16 |        }
17 |    }
18 |

```

# SelectAllPilotsNonGeneric

For languages without generics support (Java1.1-1.4; .NET1.0):

SimpleExamples.java: selectAllPilotsNonGeneric

```
01 | private static void selectAllPilotsNonGeneric() {
02 |     ObjectContainer container = database();
03 |     if (container != null) {
04 |         try {
05 |             List result = container.query(new Predicate() {
06 |                 public boolean match(Object object) {
07 |                     // each Pilot is included in the result
08 |                     if (object instanceof Pilot) {
09 |                         return true;
10 |                     }
11 |                     return false;
12 |                 }
13 |             });
14 |             listResult(result);
15 |         } catch (Exception ex) {
16 |             System.out.println("System Exception: " + ex.getMessage());
17 |         } finally {
18 |             closeDatabase();
19 |         }
20 |     }
21 | }
```

This revision (3) was last Modified 2007-04-23T19:28:12 by Tetyana.

# Using Selection Criterias

## Contents

- [SelectPilot5Points](#)
- [SelectTestPilots](#)
- [SelectPilotsNumberX6](#)

The following examples show how to select objects matching a specific criteria. [Store Pilots](#) function is used to fill in the database.

## SelectPilot5Points

Select only those pilots, which have 5 points.

`SimpleExamples.java: selectPilot5Points`

```

01 | private static void selectPilot5Points() {
02 |     ObjectContainer container = database();
03 |     if (container != null) {
04 |         try {
05 |             List<Pilot> result = container.query(new Predicate<Pilot>() {
06 |                 public boolean match(Pilot pilot) {
07 |                     // pilots with 5 points are included in the
08 |                     // result
09 |                     return pilot.getPoints() == 5;
10 |                 }
11 |             });
12 |             listResult(result);
13 |         } catch (Exception ex) {
14 |             System.out.println("System Exception: " + ex.getMessage());
15 |         } finally {
16 |             closeDatabase();
17 |         }
18 |     }

```

19 L }

## SelectTestPilots

Select all pilots, whose name includes "Test".

SimpleExamples.java: selectTestPilots

```

01 ①private static void selectTestPilots() {
02 |     ObjectContainer container = database();
03 |     if (container != null) {
04 |         try {
05 |             List<Pilot> result = container.query(new Predicate<Pilot>() {
06 |                 public boolean match(Pilot pilot) {
07 |                     // all Pilots containing "Test" in the name
08 |                     // are included in the result
09 |                     return pilot.getName().indexOf("Test") >= 0;
10 |                 }
11 |             });
12 |             listResult(result);
13 |         } catch (Exception ex) {
14 |             System.out.println("System Exception: " + ex.getMessage());
15 |         } finally {
16 |             closeDatabase();
17 |         }
18 |     }
19 L }
```

## SelectPilotsNumberX6

Select those pilots, whose name (number) ends with 6.

SimpleExamples.java: selectPilotsNumberX6

```
01 ①②private static void selectPilotsNumberX6() {  
02 |     ObjectContainer container = database();  
03 ③④    if (container != null) {  
04 ⑤⑥        try {  
05 ⑦            List<Pilot> result = container.query(new Predicate<Pilot>() {  
06 ⑧                public boolean match(Pilot pilot) {  
07                    // all Pilots with the name ending with 6 will  
08                    // be included  
09                    return pilot.getName().endsWith("6");  
10                }  
11            });  
12            listResult(result);  
13 ⑩        } catch (Exception ex) {  
14            System.out.println("System Exception: " + ex.getMessage());  
15 ⑪        } finally {  
16            closeDatabase();  
17        }  
18    }  
19 }
```

This revision (3) was last Modified 2007-09-16T16:50:19 by Tetyana.

# Selecting Ranges

## Contents

- [SelectTestPilots6PointsMore](#)
- [SelectPilots6To12Points](#)
- [SelectPilotsRandom](#)
- [SelectPilotsEven](#)
- [SelectAnyOnePilot](#)
- [SelectDistinctPilots](#)

This group of examples shows how to select ranges of objects. [Store Pilots](#) function is used to fill in the database.

## SelectTestPilots6PointsMore

Select "Test" pilots with the points range of more than 6.

**SimpleExamples.java: selectTestPilots6PointsMore**

```

01  private static void selectTestPilots6PointsMore() {
02 |     ObjectContainer container = database();
03 |     if (container != null) {
04 |         try {
05 |             List<Pilot> result = container.query(new Predicate<Pilot>() {
06 |                 public boolean match(Pilot pilot) {
07 |                     // all Pilots containing "Test" in the name
08 |                     // and 6 point are included in the result
09 |                     boolean b1 = pilot.getName().indexOf("Test") >= 0;
10 |                     boolean b2 = pilot.getPoints() > 6;
11 |                     return b1 && b2;
12 |                 }
13 |             });
14 |             listResult(result);
15 |         } catch (Exception ex) {
16 |             System.out.println("System Exception: " + ex.getMessage());
17 |         } finally {

```

```

18 |         closeDatabase();
19 |     }
20 | }
21 | }
```

## SelectPilots6To12Points

Select all pilots, who have points in [6,12] range.

SimpleExamples.java: selectPilots6To12Points

```

01 private static void selectPilots6To12Points() {
02 |     ObjectContainer container = database();
03 |     if (container != null) {
04 |         try {
05 |             List<Pilot> result = container.query(new Predicate<Pilot>() {
06 |                 public boolean match(Pilot pilot) {
07 |                     // all Pilots having 6 to 12 point are
08 |                     // included in the result
09 |                     return ((pilot.getPoints() >= 6) && (pilot.getPoints() <= 12));
10 |                 }
11 |             });
12 |             listResult(result);
13 |         } catch (Exception ex) {
14 |             System.out.println("System Exception: " + ex.getMessage());
15 |         } finally {
16 |             closeDatabase();
17 |         }
18 |     }
19 | }
```

## SelectPilotsRandom

Select pilots randomly: random array of point values is generated, those pilots who have points values within this array are included in the result set.

## SimpleExamples.java: selectPilotsRandom

```
01  private static void selectPilotsRandom() {  
02 |     ObjectContainer container = database();  
03 |     if (container != null) {  
04 |         try {  
05 |             List<Pilot> result = container.query(new Predicate<Pilot>() {  
06 |                 private ArrayList randomArray = null;  
07 |  
08 |                 private List getRandomArray() {  
09 |                     if (randomArray == null) {  
10 |                         randomArray = new ArrayList();  
11 |                         for (int i = 0; i < 10; i++) {  
12 |                             randomArray.add((int) (Math.random() * 10));  
13 |                             System.out.println(randomArray.get(i));  
14 |                         }  
15 |                     }  
16 |                     return randomArray;  
17 |                 }  
18 |  
19 |                 public boolean match(Pilot pilot) {  
20 |                     // all Pilots having points in the values of  
21 |                     // the randomArray  
22 |                     return getRandomArray().contains(pilot.getPoints());  
23 |                 }  
24 |             });  
25 |             listResult(result);  
26 |         } catch (Exception ex) {  
27 |             System.out.println("System Exception: " + ex.getMessage());  
28 |         } finally {  
29 |             closeDatabase();  
30 |         }  
31 |     }  
32 | }
```

32 L }

## SelectPilotsEven

Select pilots with even points.

SimpleExamples.java: selectPilotsEven

```

01  private static void selectPilotsEven() {
02 |     ObjectContainer container = database();
03 |     if (container != null) {
04 |         try {
05 |             List<Pilot> result = container.query(new Predicate<Pilot>() {
06 |                 public boolean match(Pilot pilot) {
07 |                     // all Pilots having even points
08 |                     return pilot.getPoints() % 2 == 0;
09 |                 }
10 |             });
11 |             listResult(result);
12 |         } catch (Exception ex) {
13 |             System.out.println("System Exception: " + ex.getMessage());
14 |         } finally {
15 |             closeDatabase();
16 |         }
17 |     }
18 | }
```

## SelectAnyOnePilot

Select one pilot and quit.

SimpleExamples.java: selectAnyOnePilot

```

01  private static void selectAnyOnePilot() {
02 |     ObjectContainer container = database();
```

```

03  if (container != null) {
04      try {
05          List<Pilot> result = container.query(new Predicate<Pilot>() {
06              boolean selected = false;
07
08              public boolean match(Pilot pilot) {
09                  // return only first result (first result can
10                  // be any value from the resultset)
11                  if (!selected) {
12                      selected = true;
13                      return selected;
14                  } else {
15                      return !selected;
16                  }
17              }
18          });
19          listResult(result);
20      } catch (Exception ex) {
21          System.out.println("System Exception: " + ex.getMessage());
22      } finally {
23          closeDatabase();
24      }
25  }
26 }
```

## SelectDistinctPilots

This example shows how to select only unique results from the non-unique contents in the database.

SimpleExamples.java: storeDuplicates

```

01 private static void storeDuplicates() {
02     new File(DB40_FILE_NAME).delete();
03     ObjectContainer container = database();
04     if (container != null) {
```

```

05 |     try {
06 |         Pilot pilot;
07 |         for (int i = 0; i < OBJECT_COUNT; i++) {
08 |             pilot = new Pilot("Test Pilot #" + i, i);
09 |             container.set(pilot);
10 |         }
11 |         for (int i = 0; i < OBJECT_COUNT; i++) {
12 |             pilot = new Pilot("Test Pilot #" + i, i);
13 |             container.set(pilot);
14 |         }
15 |         container.commit();
16 |     } catch (Db4oException ex) {
17 |         System.out.println("Db4o Exception: " + ex.getMessage());
18 |     } catch (Exception ex) {
19 |         System.out.println("System Exception: " + ex.getMessage());
20 |     } finally {
21 |         closeDatabase();
22 |     }
23 | }
24 |

```

## SimpleExamples.java: selectDistinctPilots

```

01 |     private static void selectDistinctPilots() {
02 |         ObjectContainer container = database();
03 |         if (container != null) {
04 |             try {
05 |                 DistinctPilotsPredicate predicate = new DistinctPilotsPredicate();
06 |                 List<Pilot> result = container.query(predicate);
07 |                 listResult(predicate.uniqueResult());
08 |             } catch (Exception ex) {
09 |                 System.out.println("System Exception: " + ex.getMessage());
10 |             } finally {
11 |                 closeDatabase();

```

```
12 |      }
13 |      }
14 L }
```

SimpleExamples.java: DistinctPilotsPredicate

```
1
```

This revision (4) was last Modified 2007-09-16T16:58:36 by Tetyana.

# Sorting

## Contents

- [GetSortedPilots](#)
- [GetPilotsSortByNameAndPoints](#)
- [GetPilotsSortWithComparator](#)

The following examples represent NQ sorting techniques. [Store Pilots](#) function is used to fill in the database.

## GetSortedPilots

Select all the pilots from the database and sort descending by points.

### SimpleExamples.java: getSortedPilots

```

01  public static void getSortedPilots() {
02 |     ObjectContainer container = database();
03 |     try {
04 |         List result = container.query(new Predicate<Pilot>() {
05 |             public boolean match(Pilot pilot) {
06 |                 return true;
07 |             }
08 |         }, new QueryComparator<Pilot>() {
09 |             // sort by points
10 |             public int compare(Pilot p1, Pilot p2) {
11 |                 return p2.getPoints() - p1.getPoints();
12 |             }
13 |         });
14 |         listResult(result);
15 |     } finally {
16 |         closeDatabase();
17 |     }
18 }
```

## GetPilotsSortByNameAndPoints

Select all pilots, sort descending by name and by points.

SimpleExamples.java: getPilotsSortByNameAndPoints

```
01 public static void getPilotsSortByNameAndPoints() {
02     ObjectContainer container = database();
03     try {
04         List result = container.query(new Predicate<Pilot>() {
05             public boolean match(Pilot pilot) {
06                 return true;
07             }
08         }, new QueryComparator<Pilot>() {
09             // sort by name then by points: descending
10             public int compare(Pilot p1, Pilot p2) {
11                 int result = p1.getName().compareTo(p2.getName());
12                 if (result == 0) {
13                     return p1.getPoints() - p2.getPoints();
14                 } else {
15                     return -result;
16                 }
17             }
18         });
19         listResult(result);
20     } finally {
21         closeDatabase();
22     }
23 }
```

## GetPilotsSortWithComparator

Sort by points using pre-defined comparator.

#### SimpleExamples.java: getPilotsSortWithComparator

```

01  public static void getPilotsSortWithComparator() {
02 |     ObjectContainer container = database();
03 |     try {
04 |         List result = container.query(new Predicate<Pilot>() {
05 |             public boolean match(Pilot pilot) {
06 |                 return true;
07 |             }
08 |         }, new PilotComparator());
09 |         listResult(result);
10 |     } finally {
11 |         closeDatabase();
12 |     }
13 }
```

#### SimpleExamples.java: PilotComparator

```

01  public static class PilotComparator implements Comparator<Pilot> {
02 |     public int compare(Pilot p1, Pilot p2) {
03 |         int result = p1.getName().compareTo(p2.getName());
04 |         if (result == 0) {
05 |             return p1.getPoints() - p2.getPoints();
06 |         } else {
07 |             return -result;
08 |         }
09 |     }
10 }
```

This revision (2) was last Modified 2007-04-23T19:46:09 by Tetyana.

# Result Representation

## Contents

- [SelectAndChangePilots](#)

This example shows how to modify the query output without effecting the objects in the database. [Store Pilots](#) function is used to fill in the database.

## SelectAndChangePilots

SimpleExamples.java: selectAndChangePilots

```

01 ①②private static void selectAndChangePilots() {
02 |     ObjectContainer container = database();
03 |     if (container != null) {
04 |         try {
05 |             List<Pilot> result = container.query(new Predicate<Pilot>() {
06 |                 public boolean match(Pilot pilot) {
07 |                     // Add ranking to the pilots during the query.
08 |                     // Note: pilot records in the database won't
09 |                     // be changed!!!
10 |                     if (pilot.getPoints() <= 5) {
11 |                         pilot.setName(pilot.getName() + ": weak");
12 |                     } else if (pilot.getPoints() > 5
13 |                               && pilot.getPoints() <= 15) {
14 |                         pilot.setName(pilot.getName() + ": average");
15 |                     } else if (pilot.getPoints() > 15) {
16 |                         pilot.setName(pilot.getName() + ": strong");
17 |                     }
18 |                     return true;
19 |                 }
20 |             });

```

```
21 |         listResult(result);  
22 |     } catch (Exception ex) {  
23 |         System.out.println("System Exception: " + ex.getMessage());  
24 |     } finally {  
25 |         closeDatabase();  
26 |     }  
27 | }  
28 }
```

This revision (3) was last Modified 2007-09-16T17:01:14 by Tetyana.

# Calculation Examples

## Contents

- [SumPilotPoints](#)
- [SelectMinPointsPilot](#)
- [AveragePilotPoints](#)
- [CountSubGroups](#)

This set of examples shows how to use Native Queries to perform different calculations on the objects in the database. [Store Pilots](#) function is used to fill in the database.

## SumPilotPoints

Calculate the sum of the points of all the pilots in the database.

**CalculationExamples.java: sumPilotPoints**

```

01 ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰
02 ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰
03 ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰
04 ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰
05 ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰
06 ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰
07 ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰
08 ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰
09 ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰
10 ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰
11 ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰
12 ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰
13 ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰
14 ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰
15 ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰
16 ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰

```

## CalculationExamples.java: SumPredicate

```

1  private static class SumPredicate extends Predicate<Pilot> {
2      private int sum = 0;
3
4      public boolean match(Pilot pilot) {
5          // return all pilots
6          sum += pilot.getPoints();
7          return true;
8      }
9  }

```

## SelectMinPointsPilot

Find a pilot having the minimum points.

## CalculationExamples.java: selectMinPointsPilot

```

01  private static void selectMinPointsPilot() {
02      ObjectContainer container = database();
03      if (container != null) {
04          try {
05              List<Pilot> result = container.query(new Predicate<Pilot>() {
06                  public boolean match(Pilot pilot) {
07                      // return all pilots
08                      return true;
09                  }
10              }, new QueryComparator<Pilot>() {
11                  // sort by points then by name
12                  public int compare(Pilot p1, Pilot p2) {
13                      return p1.getPoints() - p2.getPoints();
14                  }
15              });
16              if (result.size() > 0) {

```

```

17 |         System.out.println("The min points result is: "
18 |             + result.get(0));
19 |
20 } catch (Exception ex) {
21 |     System.out.println("System Exception: " + ex.getMessage());
22 } finally {
23 |     closeDatabase();
24 |
25 }
26 }

```

## AveragePilotPoints

---

Calculate what is the average amount of points for all the pilots in the database.

CalculationExamples.java: averagePilotPoints

```

01 private static void averagePilotPoints() {
02     ObjectContainer container = database();
03
04     if (container != null) {
05         try {
06             AveragePredicate averagePredicate = new AveragePredicate();
07             List<Pilot> result = container.query(averagePredicate);
08             if (averagePredicate.count > 0) {
09                 System.out
10                     .println("Average points for professional pilots: "
11                         + averagePredicate.sum
12                         / averagePredicate.count);
13             } else {
14                 System.out.println("No results");
15             }
16 } catch (Exception ex) {
17     System.out.println("System Exception: " + ex.getMessage());

```

## Calculation Examples

```
18 } finally {  
19     closeDatabase();  
20 }  
21 }  
22 }
```

### CalculationExamples.java: AveragePredicate

```
01 private static class AveragePredicate extends Predicate<Pilot> {  
02     private int sum = 0;  
03  
04     private int count = 0;  
05  
06     public boolean match(Pilot pilot) {  
07         // return professional pilots  
08         if (pilot.getName().startsWith("Professional")) {  
09             sum += pilot.getPoints();  
10             count++;  
11             return true;  
12         }  
13         return false;  
14     }  
15 }
```

## CountSubGroups

Calculate how many pilots are in each group ("Test", "Professional").

### CalculationExamples.java: countSubGroups

```
01 private static void countSubGroups() {  
02     ObjectContainer container = database();  
03     if (container != null) {
```

```

04|    try {
05|        CountPredicate predicate = new CountPredicate();
06|        List<Pilot> result = container.query(predicate);
07|        listResult(result);
08|        Iterator keyIterator = predicate.countMap.keySet().iterator();
09|        while (keyIterator.hasNext()) {
10|            String key = keyIterator.next().toString();
11|            System.out
12|                .println(key + ": " + predicate.countMap.get(key));
13|        }
14|    } catch (Exception ex) {
15|        System.out.println("System Exception: " + ex.getMessage());
16|    } finally {
17|        closeDatabase();
18|    }
19|
20}

```

## CalculationExamples.java: CountPredicate

```

01| private static class CountPredicate extends Predicate<Pilot> {
02|
03|     private HashMap countMap = new HashMap();
04|
05|     public boolean match(Pilot pilot) {
06|         // return all Professional and Test pilots and count in
07|         // each category
08|         String[] keywords = { "Professional", "Test" };
09|         for (int i = 0; i < keywords.length; i++) {
10|             if (pilot.getName().startsWith(keywords[i])) {
11|                 if (countMap.containsKey(keywords[i])) {
12|                     countMap.put(keywords[i], ((Integer) countMap
13|                         .get(keywords[i])) + 1);

```

```
14 }  
15 |         countMap. put( keywords[ i ], 1 );  
16 |     }  
17 |     return true;  
18 | }  
19 | }  
20 | return false;  
21 | }  
22 }
```

This revision (2) was last Modified 2007-04-23T19:51:36 by Tetyana.

# Combined Result Sets

## Contents

- [SelectPilotsAndTrainees](#)
- [SelectPilotsInRange](#)

This set of examples shows how to use NQ to select objects from more than one result sets. [Store Cars](#) and [Store Persons](#) functions are used to fill in the database.

## SelectPilotsAndTrainees

Selects all pilots and trainees using Person superclass.

MultiExamples.java: selectPilotsAndTrainees

```

01 ①private static void selectPilotsAndTrainees() {
02 |    ObjectContainer container = database();
03 |    if (container != null) {
04 |        try {
05 |            List<Person> result = container.query(new Predicate<Person>() {
06 |                public boolean match(Person person) {
07 |                    // all persons
08 |                    return true;
09 |                }
10 |            });
11 |            listResult(result);
12 |        } catch (Exception ex) {
13 |            System.out.println("System Exception: " + ex.getMessage());
14 |        } finally {
15 |            closeDatabase();
16 |        }
17 |    }
18 |
}

```

# SelectPilotsInRange

Selects all cars that have pilot field in the preselected Pilot array.

MultiExamples.java: selectPilotsInRange

```
01 ①②private static void selectPilotsInRange() {
02 |     ObjectContainer container = database();
03 |     if (container != null) {
04 |         try {
05 |             List<Car> result = container.query(new Predicate<Car>() {
06 |                 private List<Pilot> pilots = null;
07 |
08 |                 private List getPilotsList() {
09 |                     if (pilots == null) {
10 |                         pilots = database().query(new Predicate<Pilot>() {
11 |                             public boolean match(Pilot pilot) {
12 |                                 return pilot.getName().startsWith("Test");
13 |                             }
14 |                         });
15 |                     }
16 |                     return pilots;
17 |                 }
18 |
19 |                 public boolean match(Car car) {
20 |                     // all Cars that have pilot field in the
21 |                     // Pilots array
22 |                     return getPilotsList().contains(car.getPilot());
23 |                 }
24 |             });
25 |             listResult(result);
26 |         } catch (Exception ex) {
27 |             System.out.println("System Exception: " + ex.getMessage());
```

```
28 } finally {  
29 |     closeDatabase();  
30 | }  
31 | }  
32 }
```

This revision (4) was last Modified 2007-09-16T17:07:57 by Tetyana.

# Parameterized NQ

## Contents

- [GetTestPilots](#)
- [GetProfessionalPilots](#)
- [Querying Class Hierarchy](#)

The following examples show how to pass parameters to the Native Query predicate. [Store Pilots](#) function is used to fill in the database.

## GetTestPilots

Using predicate constructor to specify the querying parameter.

### ParameterizedExamples.java: getTestPilots

```

01 ①②private static void getTestPilots() {
02 |     ObjectContainer container = database();
03 |     if (container != null) {
04 |         try {
05 |             List<Pilot> result = container.query(new PilotNamePredicate(
06 |                 "Test"));
07 |             listResult(result);
08 |         } catch (Exception ex) {
09 |             System.out.println("System Exception: " + ex.getMessage());
10 |         } finally {
11 |             closeDatabase();
12 |         }
13 |     }
14 | }
```

### ParameterizedExamples.java: PilotNamePredicate

```

01 ①②private static class PilotNamePredicate extends Predicate<Pilot> {
```

```

02 |     private String startsWith;
03 |
04 |     public PilotNamePredicate(String startsWith) {
05 |         this.startsWith = startsWith;
06 |     }
07 |
08 |     public boolean match(Pilot pilot) {
09 |         return pilot.getName().startsWith(startsWith);
10 |     }
11 |

```

## GetProfessionalPilots

---

Using a function to process the NQ.

### ParameterizedExamples.java: getProfessionalPilots

```

01 |     private static void getProfessionalPilots() {
02 |         ObjectContainer container = database();
03 |         if (container != null) {
04 |             try {
05 |                 List<Pilot> result = byNameBeginning("Professional");
06 |                 listResult(result);
07 |             } catch (Exception ex) {
08 |                 System.out.println("System Exception: " + ex.getMessage());
09 |             } finally {
10 |                 closeDatabase();
11 |             }
12 |         }
13 |

```

### ParameterizedExamples.java: byNameBeginning

```

1 |     private static List<Pilot> byNameBeginning(final String startsWith) {

```

```

2  return database().query(new Predicate<Pilot>() {
3      public boolean match(Pilot pilot) {
4          return pilot.getName().startsWith(startsWith);
5      }
6  });
7 }

```

## Querying Class Hierarchy

The following example shows how to correctly implement parameterized NQ predicate for querying derived classes.

Let's use the class hierarchy as defined in [Persistent Classes](#).

PersonNamePredicate will be used for querying classes derived from Person:

ComplexParameterizedExample.java: PersonNamePredicate

```

01 private static class PersonNamePredicate<T extends Person> extends
02  Predicate<T> {
03      private String startsWith;
04
05      public PersonNamePredicate(String startsWith) {
06          this.startsWith = startsWith;
07      }
08
09      public PersonNamePredicate(Class<T> clazz, String startsWith) {
10          super(clazz);
11          this.startsWith = startsWith;
12      }
13
14      public boolean match(T candidate) {
15          return candidate.getName().startsWith(startsWith);
16      }
17 }

```

Let's save some Pilot and Trainee objects:

#### ComplexParameterizedExample.java: storePersons

```

01  private static void storePersons() {
02 |     new File(DB40_FILE_NAME).delete();
03 |     ObjectContainer container = database();
04 |     if (container != null) {
05 |         try {
06 |             Trainee trainee;
07 |             // store OBJECT_COUNT pilots and trainees
08 |             for (int i = 0; i < OBJECT_COUNT; i++) {
09 |                 trainee = new Trainee("Trainee #" + i, new Pilot(
10 |                     "Professional Pilot #" + i, i));
11 |                 container.set(trainee);
12 |             }
13 |             // store a new trainee with a "Training" pilot
14 |             trainee = new Trainee("Trainee #1", new Pilot(
15 |                 "Training Pilot #1", 20));
16 |             container.set(trainee);
17 |             container.commit();
18 |         } catch (Db4oException ex) {
19 |             System.out.println("Db4o Exception: " + ex.getMessage());
20 |         } catch (Exception ex) {
21 |             System.out.println("System Exception: " + ex.getMessage());
22 |         } finally {
23 |             closeDatabase();
24 |         }
25 |     }
26 |

```

Now we have both Pilot and Trainee objects starting from "Train". What should we do to retrieve only Trainee objects?

## ComplexParameterizedExample.java: getTrainees

```

01  private static void getTrainees() {
02 |     ObjectContainer container = database();
03 |     if (container != null) {
04 |         try {
05 |             // query for Trainee objects starting with "Train".
06 |             // Wrongly created predicate mixes Trainee and Pilot
07 |             // objects and creates a resultset based on only "Tr"
08 |             // criteria (class of an object is not considered)
09 |
10 |             testQuery(container, createPredicateWrong(Trainee.class,
11 |                                                 "Train"));
12 |             // Correctly created result set returns only objects
13 |             // of the requested class
14 |             testQuery(container, createPredicateCorrect(Trainee.class,
15 |                                                 "Train"));
16 |         } catch (Exception ex) {
17 |             System.out.println("System Exception: " + ex.getMessage());
18 |         } finally {
19 |             closeDatabase();
20 |         }
21 |     }
22 | }
```

## ComplexParameterizedExample.java: testQuery

```

01  private static void testQuery(ObjectContainer container,
02 |                                 Predicate<Trainee> predicate) {
03 |     List<Trainee> result = container.query(predicate);
04 |     System.out.println(result.size());
05 |     try {
06 |         for (Trainee trainee : result) {
```

```

07 |         System.out.println(trai nee);
08 |     }
09 | } catch (Exception ex) {
10 |     System.out.println(ex.toString());
11 | }
12 | }
```

## ComplexParameterizedExample.java: createPredicateWrong

```

1 private static <T extends Person> Predicate<T> createPredicateWrong(
2     Class<T> clazz, String startsWith) {
3     return new PersonNamePredicate<T>(startsWith);
4 }
```

## ComplexParameterizedExample.java: createPredicateCorrect

```

1 private static <T extends Person> Predicate<T> createPredicateCorrect(
2     Class<T> clazz, String startsWith) {
3     return new PersonNamePredicate<T>(clazz, startsWith);
4 }
```

You can see that in order to get the correct results we need to supply our predicate class with the queried class, otherwise both Pilot and Trainee objects will be included in the result set.

This revision (6) was last Modified 2007-09-16T17:10:48 by Tetyana.

# Store Pilots

SimpleExamples.java: storePilots

```
01  private static void storePilots() {  
02 |     new File(DB40_FILE_NAME).delete();  
03 |     ObjectContainer container = database();  
04 |     if (container != null) {  
05 |         try {  
06 |             Pilot pilot;  
07 |             for (int i = 0; i < OBJECT_COUNT; i++) {  
08 |                 pilot = new Pilot("Test Pilot #" + i, i);  
09 |                 container.set(pilot);  
10 |             }  
11 |             for (int i = 0; i < OBJECT_COUNT; i++) {  
12 |                 pilot = new Pilot("Professional Pilot #" + (i + 10), i + 10);  
13 |                 container.set(pilot);  
14 |             }  
15 |             container.commit();  
16 |         } catch (Db4oException ex) {  
17 |             System.out.println("Db4o Exception: " + ex.getMessage());  
18 |         } catch (Exception ex) {  
19 |             System.out.println("System Exception: " + ex.getMessage());  
20 |         } finally {  
21 |             closeDatabase();  
22 |         }  
23 |     }  
24 }
```

This revision (1) was last Modified 2007-04-23T19:17:47 by Tetyana.

# Store Persons

MultiExamples.java: storePilotsAndTrainees

```
01  private static void storePilotsAndTrainees() {  
02 |     new File(DB40_FILE_NAME).delete();  
03 |     ObjectContainer container = database();  
04 |     if (container != null) {  
05 |         try {  
06 |             Pilot pilot;  
07 |             Trainee trainee;  
08 |             for (int i = 0; i < OBJECT_COUNT; i++) {  
09 |                 pilot = new Pilot("Professional Pilot #" + i, i);  
10 |                 trainee = new Trainee("Trainee #" + i, pilot);  
11 |                 container.set(trainee);  
12 |             }  
13 |             container.commit();  
14 |         } catch (Db4oException ex) {  
15 |             System.out.println("Db4o Exception: " + ex.getMessage());  
16 |         } catch (Exception ex) {  
17 |             System.out.println("System Exception: " + ex.getMessage());  
18 |         } finally {  
19 |             closeDatabase();  
20 |         }  
21 |     }  
22 }
```

This revision (1) was last Modified 2007-04-23T19:54:51 by Tetyana.

# Store Cars

MultiExamples.java: storeCars

```
01  private static void storeCars() {  
02 |     new File(DB40_FILE_NAME).delete();  
03 |     ObjectContainer container = database();  
04 |     if (container != null) {  
05 |         try {  
06 |             Car car;  
07 |             for (int i = 0; i < OBJECT_COUNT; i++) {  
08 |                 car = new Car("BMW", new Pilot("Test Pilot #" + i, i));  
09 |                 container.set(car);  
10 |             }  
11 |             for (int i = 0; i < OBJECT_COUNT; i++) {  
12 |                 car = new Car("Ferrari", new Pilot("Professional Pilot #"  
13 |                     + (i + 10), (i + 10)));  
14 |                 container.set(car);  
15 |             }  
16 |             container.commit();  
17 |         } catch (Db4oException ex) {  
18 |             System.out.println("Db4o Exception: " + ex.getMessage());  
19 |         } catch (Exception ex) {  
20 |             System.out.println("System Exception: " + ex.getMessage());  
21 |         } finally {  
22 |             closeDatabase();  
23 |         }  
24 |     }  
25 | }
```

This revision (1) was last Modified 2007-04-23T19:58:03 by Tetyana.

# Persistent Classes

Person.java

```
01 /* Copyright (C) 2007 db4objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.nqcollection;
04
05
06 public interface Person {
07 |
08 |   public String getName();
09 |
10 }
```

Pilot.java

```
01 /* Copyright (C) 2004 - 2007 db4objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.nqcollection;
04
05 public class Pilot implements Person {
06 |   private String name;
07 |
08 |   private int points;
09 |
10 public Pilot(String name, int points) {
11 |   this.name = name;
12 |   this.points = points;
13 | }
14 |
15 public String getName() {
```

```

16 |     return name;
17 | }
18 |
19 |⊕⊕ public void setName(String name) {
20 |     this.name = name;
21 | }
22 |
23 |⊕⊕ public int getPoints() {
24 |     return points;
25 | }
26 |
27 |⊕⊕ public boolean equals(Object obj) {
28 |⊕⊕     if (obj instanceof Pilot) {
29 |         return (((Pilot) obj).getName().equals(name) &&
30 |                 ((Pilot) obj).getPoints() == points);
31 |     }
32 |     return false;
33 | }
34 |
35 |⊕⊕ public String toString() {
36 |     return name + "/" + points;
37 | }
38 |
39 |⊕⊕ public int hashCode() {
40 |     return name.hashCode() + points;
41 | }
42 |}

```

## Trainee.java

```

01 |⊕⊕/* Copyright (C) 2007 db4objects Inc. http://www.db4o.com */
02 |
03 package com.db4odoc.nqcollection;

```

```
04
05
06 public class Trainee implements Person {
07 |
08 |     private String name;
09 |     private Pilot instructor;
10 |
11 public Trainee(String name, Pilot pilot) {
12 |         this.name = name;
13 |         this.instructor = pilot;
14 }
15 |
16 public String getName() {
17 |     return name;
18 }
19 |
20 public Pilot getInstructor() {
21 |     return instructor;
22 }
23 |
24 public String toString() {
25 |     return name + "(" + instructor + ")";
26 }
27 }
```

This revision (1) was last Modified 2007-04-23T18:23:09 by Tetyana.

# Data Binding

This topic applies to .NET version only

One common question we get from our users is:

**Can I still take advantage of data aware control mechanisms with my objects?**

The answer is quite simply yes. .NET data binding also works with plain objects.

This is also to say that using db4o is completely orthogonal to the use of data binding.

The usual pattern would be something like the following:

- the code asks db4o to retrieve the objects that must be presented to the user
- the UI controls are bound to the objects (no interaction with db4o here)
- the user interacts with the objects through the controls (no interaction with db4o here)
- when the user is done interacting with the objects or by his request, the application will ask db4o to persist his changes

Let's take a very simple example that illustrates the points above.

Our business class:

This revision (6) was last Modified 2007-05-07T12:25:53 by Tetyana.

# Refactoring and Schema Evolution

Application design is a volatile thing: it changes from version to version, from one customer implementation to another. The database (if used) changes together with the application. For relational databases this process is called Schema Evolution, for object databases the term Refactoring is used as more appropriate.

Object database refactoring changes the shape of classes stored on the disk. The main challenge here is to preserve old object information and make it usable with the new classes' design.

More Reading:

- [Automatic refactoring](#)
- [Refactoring API](#)
- [Field type change](#)
- [Refactoring Class Hierarchy](#)

This revision (5) was last Modified 2006-11-14T18:57:20 by Eric Falsken.

# Automatic refactoring

In simplest cases db4o handles schema changes automatically:

- If you **add** a new field, db4o automatically starts storing the new data. Older instances of your stored class (from before the field was added) are still loaded, but the new field is set to its default value, or null.
- If you **remove** a field, db4o ignores the stored value when activating instances of your class. The stored value is not removed from the database until the next Defragment, and is still accessible via the StoredClass/StoredField API.
- If you **add an interface** to be implemented by your stored classes, db4o automatically starts using it and you are able to retrieve your saved data using new interface.

This revision (3) was last Modified 2006-11-13T19:26:27 by Tetyana.

# Refactoring API

Db4o provides special API which can help you to move classes between packages (Java)/namespaces(.NET), rename classes or fields:

Java:

```
Db4o.configure().objectClass("package.class").rename("newPackage.newClass")
Db4o.configure().objectClass("package.class").objectField("oldField").rename("newField")
```

The safe order of actions for rename calls is:

1. Backup your database and application
2. Close all open objectContainers if any
3. Rename classes or fields or copy classes to the new package/namespace in your application. (Do not remove old classes yet).
4. Issue ObjectClass#rename and objectField#rename calls without having an ObjectContainer open.
5. Open database file and close it again without actually working with it.
6. Remove old classes (if applicable).

After that you will only see the new classes/fields in ObjectManager, the old ones will be gone.

Let's look how it works on an example. We will use initial class Pilot:

Pilot.java

```
01 /* Copyright (C) 2004 - 2007 db4objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.refactoring;
04
05 public class Pilot {
06     private String name;
07
08     public Pilot(String name) {
09         this.name = name;
10     }
11
12     public String getName() {
13         return name;
14     }
15
16     public String toString() {
17         return name;
18     }
19 }
```

and change it to the new class PilotNew renaming field and changing package/namespace at the same time:

## PilotNew.java

```

01 /* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.refactoring;
04
05 public class PilotNew {
06     private String identity;
07
08     private int points;
09
10    public PilotNew(String name, int points) {
11        this.identity = name;
12        this.points = points;
13    }
14
15    public String getIdentity() {
16        return identity;
17    }
18
19    public String toString() {
20        return identity + "/" + points;
21    }
22 }

```

First let's create a database and fill it with some values:

## RefactoringExample.java: setObjects

```

01 private static void setObjects() {
02     new File(DB4O_FILE_NAME).delete();
03     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
04    try {
05        Pilot pilot = new Pilot("Rubens Barrichello");
06        container.set(pilot);
07        pilot = new Pilot("Michael Schumacher");
08        container.set(pilot);
09    } finally {

```

```

10 |     container.close();
11 | }
12 | }
```

## RefactoringExample.java: checkDB

```

1▣▣private static void checkDB() {
2|     ObjectContainer container = Db4o.openFile(DB40_FILE_NAME);
3▣▣    try {
4|         ObjectSet result = container.get(new Object());
5|         listResult(result);
6▣▣    } finally {
7|        container.close();
8|    }
9| }
```

We already have PilotNew class so we can go on with renaming:

## RefactoringExample.java: changeClass

```

1▣▣private static void changeClass() {
2|     Configuration configuration = Db4o.newConfiguration();
3|     configuration.objectClass(Pilot.class).rename("com.db4odoc.f1.refactoring.PilotNew");
4|     configuration.objectClass(PilotNew.class).objectField("name").rename("identity");
5|     ObjectContainer container = Db4o.openFile(configuration, DB40_FILE_NAME);
6|     container.close();
7| }
```

Now the data for the old Pilot class should be transferred to the new PilotNew class, and "name" field data should be stored in "identity" field.

To make our check more complicated let's add some data for our new class:

## RefactoringExample.java: setNewObjects

```

01▣▣private static void setNewObjects() {
02|     ObjectContainer container = Db4o.openFile(DB40_FILE_NAME);
03▣▣    try {
04|        PilotNew pilot = new PilotNew("Rubens Barichello", 99);
05|        container.set(pilot);
06|        pilot = new PilotNew("Michael Schumacher", 100);
07|        container.set(pilot);
```

```

08 } finally {
09 |     container.close();
10 | }
11 }

```

We can check what is stored in the database now:

RefactoringExample.java: retrievePilotNew

```

1 private static void retrievePilotNew() {
2     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
3     try {
4         ObjectSet result = container.query(PilotNew.class);
5         listResult(result);
6     } finally {
7         container.close();
8     }
9 }

```

There is one thing to remember. The rename feature is intended to rename a class from one name to the other. Internally this will rename the meta-information. If you will try to rename class to the name that is already stored in the database, the renaming will fail, because the name is reserved. In our example it will happen if setNewObjects method will be called before changeClass.

This revision (11) was last Modified 2008-03-09T06:40:25 by Tetyana.

# Field type change

The reviewed refactoring types are fairly easy. It gets more difficult when you need to change a field's type.

If you modify a field's type, db4o internally creates a new field of the same name, but with the new type. The values of the old typed field are still present, but hidden. If you will change the type back to the old type the old values will still be there.

You can access the values of the previous field data using StoredField API.

Java:

```
StoredClass#storedField(name, type)
```

gives you access to the field, which type was changed.

Java:

```
StoredField#get(Object)
```

allows you to get the old field value for the specified object.

To see how it works on example, let's change Pilot's field name from type string to type Identity:

**Identity.java**

```
01 /* Copyright (C) 2004 - 2007 db4objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.refactoring.newclasses;
04
05 public class Identity {
06 |   private String name;
07 |
08 |   private String id;
09 |
10     public Identity(String name, String id) {
11 |      this.name = name;
12 |      this.id = id;
13 |    }
14 |
15     public void setName(String name) {
16 |      this.name = name;
17 |    }
18 |
19     public void setId(String id) {
```

```

20 |     this.id = id;
21 | }
22 |
23 | public String toString() {
24 |     return name + "[" + id + "]";
25 | }
26 |
27 |}

```

Now to access old "name" values and transfer them to the new "name" we can use the following procedure:

RefactoringExample.java: transferValues

```

01 | private static void transferValues() {
02 |     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
03 |     try {
04 |         StoredClass sc = container.ext().storedClass("com.db4odoc.f1.refactoring.oldclasses.Pilot");
05 |         System.out.println("Stored class: " + sc.toString());
06 |         StoredField sfOld = sc.storedField("name", String.class);
07 |         System.out.println("Old field: " + sfOld.toString() + "; " + sfOld.getStoredType());
08 |         Query q = container.query();
09 |         q.constrain(Pilot.class);
10 |         ObjectSet result = q.execute();
11 |         for (int i = 0; i < result.size(); i++) {
12 |             Pilot pilot = (Pilot) result.get(i);
13 |             System.out.println("Pilot=" + pilot);
14 |             pilot.setName(new Identity(sfOld.get(pilot).toString(), ""));
15 |             System.out.println("Pilot=" + pilot);
16 |             container.set(pilot);
17 |         }
18 |
19 |     } finally {
20 |         container.close();
21 |     }
22 | }

```

These are the basic refactoring types, which can help with any changes you will need to make.

This revision (9) was last Modified 2007-08-15T20:48:35 by Eric Falsken.

# Refactoring Class Hierarchy

db4o does not directly support the following two refactorings:

- Inserting classes into an inheritance hierarchy.
- Removing class from inheritance hierarchies.

For example:

```
class A
```

```
class B : A
```

```
class C : B
```

1. A new Class D can not be introduced above C.
2. Classes A and B can not be removed.

The only current possible solution for the above refactorings is a workaround:

1. Create the new hierarchy with different names, preferably in a new package.
2. Copy all values from the old classes to the new classes.
3. Redirect all links from existing objects to the new classes.

More Reading:

- [Removing Class From A Hierarchy](#)
- [Inserting Class Into A Hierarchy](#)
- [A](#)
- [B](#)
- [C](#)

This revision (2) was last Modified 2007-09-30T18:22:10 by Tetyana.

# Removing Class From A Hierarchy

Let's use [A](#), [B](#) and [C](#) classes and remove B class, copying its values to the updated C class.

First of all let's store some class objects to the database:

refactoringExample.java: storeData

```

01  public static void storeData() {
02 |     new File(DB40_FILE_NAME).delete();
03 |     ObjectContainer container = Db4o.openFile(DB40_FILE_NAME);
04 |     try {
05 |         A a = new A();
06 |         a.name = "A class";
07 |         container.set(a);
08 |
09 |         B b = new B();
10 |         b.name = "B class";
11 |         b.number = 1;
12 |         container.set(b);
13 |
14 |         C c = new C();
15 |         c.name = "C class";
16 |         c.number = 2;
17 |         container.set(c);
18 |     } finally {
19 |         container.close();
20 |     }
21 }
```

refactoringExample.java: readData

```

01  public static void readData() {
```

```

02 |     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
03 |     try {
04 |         ObjectSet result = container.get(new A());
05 |         System.out.println("A class: ");
06 |         listResult(result);
07 |
08 |         result = container.get(new B());
09 |         System.out.println();
10 |         System.out.println("B class: ");
11 |         listResult(result);
12 |
13 |         result = container.get(new C());
14 |         System.out.println();
15 |         System.out.println("C class: ");
16 |         listResult(result);
17 |     } finally {
18 |         container.close();
19 |     }
20 }

```

If we will remove B class and update C class to inherit from A, we won't be able to read C data from the database anymore (exception). In order to preserve C data we will need to transfer it to another class:

## D.java

```

01 /* Copyright (C) 2007 db4objects Inc. http://www.db4o.com */
02 package com.db4odoc.refactoring.refactored;
03
04 import com.db4odoc.refactoring.initial.A;
05
06 public class D extends A {
07     public int number;
08
09     public String toString() {

```

```

10 |     return name + "/" + number;
11 | }
12 | }
```

We can also transfer B data into this class.

Once D class is created we can run the data transfer:

refactoringUtil.java: moveValues

```

01 public static void moveValues() {
02 |     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
03 |     try {
04 |         // querying for B will bring back B and C values
05 |         ObjectSet result = container.get(new B());
06 |         while (result.hasNext()) {
07 |             B b = (B) result.next();
08 |             D d = new D();
09 |             d.name = b.name;
10 |             d.number = b.number;
11 |             container.delete(b);
12 |             container.set(d);
13 |         }
14 |
15 |     } finally {
16 |         container.close();
17 |         System.out.println("Done");
18 |     }
19 | }
```

Now B and C classes can be safely removed from the project and all the references to them updated to D. We can check that all the values are in place:

RefactoringExample.java: readData

```
01 public static void readData() {  
02     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);  
03     try {  
04         ObjectSet result = container.get(new D());  
05         System.out.println();  
06         System.out.println("D class: ");  
07         listResult(result);  
08     } finally {  
09         container.close();  
10    }  
11 }
```

When performing refactoring on your working application do not forget to make a copy of the code and data before making any changes!

This revision (1) was last Modified 2007-09-30T18:40:12 by Tetyana.

# Inserting Class Into A Hierarchy

We will use the same [A](#), [B](#) and [C](#) classes as in the [previous example](#)

. The goal is to insert a new class with additional members between B and C class in the hierarchy.

Let's store some class objects first:

refactoringExample.java: storeData

```

01  public static void storeData() {
02 |     new File(DB40_FILE_NAME).delete();
03 |     ObjectContainer container = Db4o.openFile(DB40_FILE_NAME);
04  |     try {
05 |         A a = new A();
06 |         a.name = "A class";
07 |         container.set(a);
08 |
09 |         B b = new B();
10 |         b.name = "B class";
11 |         b.number = 1;
12 |         container.set(b);
13 |
14 |         C c = new C();
15 |         c.name = "C class";
16 |         c.number = 2;
17 |         container.set(c);
18  |     } finally {
19 |         container.close();
20 |     }
21 }
```

refactoringExample.java: readData

```

01 public static void readData() {
02 |     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
03 |     try {
04 |         ObjectSet result = container.get(new A());
05 |         System.out.println("A class: ");
06 |         listResult(result);
07 |
08 |         result = container.get(new B());
09 |         System.out.println();
10 |         System.out.println("B class: ");
11 |         listResult(result);
12 |
13 |         result = container.get(new C());
14 |         System.out.println();
15 |         System.out.println("C class: ");
16 |         listResult(result);
17 |     } finally {
18 |         container.close();
19 |     }
20 }

```

The following class will be inserted:

```

D.java

01 /* Copyright (C) 2007 db4o objects Inc. http://www.db4o.com */
02 package com.db4odoc.refactoring.refactored;
03
04 import java.util.Date;
05
06 import com.db4odoc.refactoring.initial.B;
07
08 public class D extends B {
09 |     public Date storedDate;

```

```

10 |
11 public String toString() {
12 |     return name + "/" + number + ":" + storedDate;
13 | }
14 }

```

Now C class must inherit from D. We can't change C class itself, because its data will be lost. Therefore we will create a new E class to hold C data:

## E.java

```

1 /* Copyright (C) 2007 db4objects Inc. http://www.db4o.com */
2 package com.db4odoc.refactoring.refactored;
3
4 public class E extends D {
5 |
6 }

```

When all the necessary classes are created we can copy C data into E class:

## refactoringUtil.java: moveValues

```

01 public static void moveValues() {
02 |     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
03 |     try {
04 |         ObjectSet result = container.get(new C());
05 |         while (result.hasNext()) {
06 |             C c = (C) result.next();
07 |             E e = new E();
08 |             e.name = c.name;
09 |             e.number = c.number;
10 |             container.delete(c);
11 |             container.set(e);
12 |         }

```

```

13 |
14     } finally {
15         container.close();
16         System.out.println("Done");
17     }
18 }

```

Now C classes can be safely removed from the project and all the references to it updated to E(or D). We can check that all the values are in place:

## RefactoringExample.java: readData

```

01 public static void readData() {
02     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
03     try {
04         ObjectSet result = container.get(new D());
05         System.out.println();
06         System.out.println("D class: ");
07         listResult(result);
08
09         result = container.get(new E());
10         System.out.println();
11         System.out.println("E class: ");
12         listResult(result);
13     } finally {
14         container.close();
15     }
16 }

```

When performing refactoring on your working application do not forget to make a copy of the code and data before making any changes!

This revision (2) was last Modified 2007-09-30T18:47:14 by Tetyana.

# A

A.java

```
01 /* Copyright (C) 2007 db4o objects Inc. http://www.db4o.com */  
02 package com.db4odoc.refactoring.initial;  
03  
04 public class A {  
05 |  
06 |   public String name;  
07 |  
08 |   public String toString(){  
09 |     return name;  
10 |   }  
11 }
```

This revision (2) was last Modified 2007-09-30T18:35:27 by Tetyana.

# B

B.java

```
01 /* Copyright (C) 2007 db4o objects Inc. http://www.db4o.com */  
02 package com.db4odoc.refactoring.initial;  
03  
04 public class B extends A {  
05     public int number;  
06  
07     public String toString(){  
08         return name + "/" + number;  
09     }  
10 }
```

This revision (1) was last Modified 2007-09-30T18:37:01 by Tetyana.

# C

C.java

```
1 /* Copyright (C) 2007 db4objects Inc. http://www.db4o.com */
2 package com.db4odoc.refactoring.initial;
3
4 public class C extends B {
5
6 }
```

This revision (1) was last Modified 2007-09-30T18:39:02 by Tetyana.

# Aliases

Db4o Alias/IAlias interface gives you an opportunity to have different names for your persistent classes in the runtime and in the database. The functionality is pretty simple: before class is saved to/retrieved from the database an alias collection is checked for the presence of an alias for the specified class name. If the alias exists, it is used for saving or retrieving instead of the original name.

Db4o provides 2 types of aliases:

- *TypeAlias* is used to alias specific class name to another class name
- *WildcardAlias* allows to alias a package, namespace or multiple similar named classes.

Aliases should be added or removed before a database file is opened.

Java:

```
Configuration. addAlias(alias)  
Configuration. removeAlias(alias)
```

More Reading:

- [TypeAlias](#)
- [WildcardAlias](#)
- [Cross-Platform Aliasing](#)
- [Cross-Platform Aliasing From .NET To Java](#)

This revision (4) was last Modified 2006-12-14T11:18:46 by Tetyana.

# TypeAlias

TypeAlias constructor accepts 2 parameters:

- stored class name - storedType
- runtime class name - runtimeType

Note that the runtimeType class should exist in your application when you configure the alias.

The alias matches are found by comparing full names of the stored and runtime classes

Let's declare a new TypeAlias

Java:

```
private static TypeAlias tAlias;
```

The following method will initialize tAlias and configure db4o to use it:

**AliasExample.java: configureClassAlias**

```
01 | private static Configuration configureClassAlias() {
02 |     // create a new alias
03 |     tAlias = new TypeAlias("com.db4odoc.alises.Pilot", "com.db4odoc.alises.Driver");
04 |     // add the alias to the db4o configuration
05 |     Configuration configuration = Db4o.newConfiguration();
06 |     configuration.addAlias(tAlias);
07 |     // check how does the alias resolve
08 |     System.out.println("Stored name for com.db4odoc.alises.Driver: "+tAlias.
resolveRuntimeName("com.db4odoc.alises.Driver"));
09 |     System.out.println("Runtime name for com.db4odoc.alises.Pilot: "+tAlias.
resolveStoredName("com.db4odoc.alises.Pilot"));
10 |     return configuration;
11 | }
```

We can always check the results of adding an alias using resolveRuntimeName and resolveStoredName as you see in the example. Basically the same methods are used internally by db4o to resolve aliased names.

**AliasExample.java: saveDrivers**

```
01 | private static void saveDrivers(Configuration configuration) {
02 |     new File(DB40_FILE_NAME).delete();
```

```

03 |     ObjectContainer container = Db4o.openFile(configuration, DB4O_FILE_NAME);
04 |     try {
05 |         Driver driver = new Driver("David Barri chello", 99);
06 |         container.set(driver);
07 |         driver = new Driver("Kimi RAIKKONEN", 100);
08 |         container.set(driver);
09 |     } finally {
10 |         container.close();
11 |     }
12 |

```

Due to the alias configured the database will have Pilot classes saved. You can check it using ObjectManager or you can remove alias and read it from the database:

#### AliasExample.java: removeClassAlias

```

1 |     private static void removeClassAlias(Configuration configuration) {
2 |         configuration.removeAlias(tAlias);
3 |

```

#### AliasExample.java: getPilots

```

1 |     private static void getPilots(Configuration configuration) {
2 |         ObjectContainer container = Db4o.openFile(configuration, DB4O_FILE_NAME);
3 |         try {
4 |             ObjectSet result = container.query(com.db4odoc.alises.Pilot.class);
5 |             listResult(result);
6 |         } finally {
7 |             container.close();
8 |         }
9 |

```

Obviously you can install the same alias again and read the stored objects using Driver class.

This revision (3) was last Modified 2008-01-12T12:15:54 by Tetyana.

# WildcardAlias

WildcardAlias allows creating aliases for packages, namespaces or multiple similar classes. WilcardAlias constructor accepts 2 parameters:

- storedPattern
- runtimePattern

\* symbol is used to specify the place where multiple matches are allowed (you can use only one \* per pattern).

Let's look how to alias all classes within one package/namespace.

AliasExample.java: savePilots

```

01  private static void savePilots(Configuration configuration) {
02 |     new File(DB40_FILE_NAME).delete();
03 |     ObjectContainer container = Db4o.openFile(configuration, DB40_FILE_NAME);
04  |     try {
05 |         Pilot pilot = new Pilot("David Barichello", 99);
06 |         container.set(pilot);
07 |         pilot = new Pilot("Kimi Räikkönen", 100);
08 |         container.set(pilot);
09  |     } finally {
10 |         container.close();
11 |     }
12 }
```

AliasExample.java: configureAlias

```

01  private static Configuration configureAlias() {
02 |     // com.db4odoc.aliases.* - package for the classes saved in the database
03 |     // com.db4odoc.aliases.newalias.* - runtime package
04 |     WildcardAlias wAlias = new WildcardAlias("com.db4odoc.aliases.*", "com.db4odoc.aliases."
05 |     newalias.*");
06 |     // add the alias to the configuration
07 |     Configuration configuration = Db4o.newConfiguration();
08 |     configuration.addAlias(wAlias);
09 |     System.out.println("Stored name for com.db4odoc.aliases.newalias.Pilot: "+wAlias.
10 |     resolveRuntimeName("com.db4odoc.aliases.newalias.Pilot"));
11 |     System.out.println("Runtime name for com.db4odoc.aliases.Pilot: "+wAlias.
12 |     resolveStoredName("com.db4odoc.aliases.Pilot"));
13 |     return configuration;
14 }
```

**11** L }

You can check the matches for the concrete classes using resolveRuntimeName and resolveStoredName.

In order to add your own aliasing logic implement Alias interface (for example you may want to use more sophisticated pattern logic). Your own resolving logic implementation should reside in resolveRuntimeName and resolveStoredName methods.

This revision (1) was last Modified 2006-12-14T12:01:25 by Tetyana.

# Cross-Platform Aliasing

One of the most valuable aliases usecases is working with persistent Java classes from a .NET application and vice versa. You can use both TypeAlias and WildcardAlias depending on how many classes you need to access.

Below you will get an example of a system where classes were saved to the database from a Java application and read and modified later from a .NET application. A vice versa example is reviewed in [Cross-Platform Aliasing From .NET To Java](#).

For example, Pilot objects are saved to a database from a Java application:

InterLanguageExample.java: saveObjects

```

01  private static void saveObjects(){
02 |     new File(DB4O_FILE_NAME).delete();
03 |     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
04  |     try {
05 |         Pilot pilot = new Pilot("David Barrichello", 99);
06 |         container.set(pilot);
07 |         pilot = new Pilot("Michael Schumacher", 100);
08 |         container.set(pilot);
09  |     } finally {
10 |         container.close();
11 |     }
12 }
```

In order to read the saved objects successfully from a .NET application we will need to define an alias for persistent classes and an alias for the Db4oDatabase class. We will use a wildcard alias for all the persistent classes:

InterLanguageExample.cs: ConfigureAlias

```

1  private static IConfiguration ConfigureAliases()
2  {
3      IConfiguration configuration = Db4oFactory.NewConfiguration();
4      configuration.AddAlias(new WildcardAlias("com.db4odoc.aliases.*", "Db4objектs.
Db4odoc.Aliases.*", Db4objектs.Db4odoc"));
5      configuration.AddAlias(new TypeAlias("com.db4o.ext.Db4oDatabase", "Db4objектs.
Db4o.Ext.Db4oDatabase", Db4objектs.Db4o));
6      return configuration;
7 }
```

InterLanguageExample.vb: ConfigureAlias

```

1 Public Shared Function ConfigureAliases() As IConfiguration
2     Dim configuration As IConfiguration = Db4oFactory.NewConfiguration()
3     configuration.AddAlias(New WildcardAlias("com.db4odoc_aliases.*", "Db4obj ects.
Db4odoc.Aliases.*", Db4obj ects.Db4odoc"))
4     configuration.AddAlias(New TypeAlias("com.db4o.ext.Db4oDatabase", "Db4obj ects.
Db4o. Ext. Db4oDatabase", Db4obj ects.Db4o"))
5     Return configuration
6 End Function

```

Now the objects are accessible from the .NET application:

InterLanguageExample.cs: GetObjects

```

01 private static void GetObjects(IConfiguration configuration)
02 {
03     IObjectContainer db = Db4oFactory.OpenFile(configuration, Db4oFileName);
04     try
05     {
06         IList<Pilot> result = db.Query<Pilot>(delegate(Pilot pilot) {
07             return pilot.Points % 2 == 0;
08         });
09         for (int i = 0; i < result.Count; i++) {
10             Pilot pilot = result[i];
11             pilot.Name = "Modified " + pilot.Name;
12             db.Set(pilot);
13         }
14         ListResult(result);
15     }
16     finally
17     {
18         db.Close();
19     }
20 }

```

InterLanguageExample.vb: GetObjects

```

1 Public Shared Sub GetObjects(ByVal configuration As IConfiguration)
2     Dim db As IObjectContainer = Db4oFactory.OpenFile(configuration, Db4oFileName)
3     Try
4         Dim result As ObjectSet = db.Query(GetType(Pilot))
5         ListResult(result)
6     Finally

```

```
7 |             db. Close()
8 |         End Try
9 L     End Sub
```

One thing to remember: field names in class definitions in Java and .NET should be exactly the same.

We can read the database from the initial Java application again. Note, that no alias is required as the class definitions were created from Java:

InterLanguageExample.java: readObjects

```
1▣■private static void readObjects(){
2|    ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
3▣■    try {
4|        ObjectSet result = container.get(new Pilot(null, 0));
5|        listResult(result);
6▣■    } finally {
7|        container.close();
8|    }
9L }
```

This revision (7) was last Modified 2007-08-12T19:44:55 by Tetyana.

# Cross-Platform Aliasing From .NET To Java

The following example shows a cross-platform system where classes were saved to the db4o database from a .NET application and read and modified later from a .Java application. A vice versa example is reviewed in [Cross-Platform Aliasing](#).

Pilot objects are saved to a database from a .NET application:

## InterLanguageExample2.cs: SaveObjects

```

01 private static void SaveObjects()
02 {
03     File.Delete(Db4oFileName);
04     IObjectContainer container = Db4oFactory.OpenFile(Db4oFileName);
05     try
06     {
07         Pilot pilot = new Pilot("David Barri chello", 99);
08         container.Set(pilot);
09         pilot = new Pilot("Michael Schumacher", 100);
10         container.Set(pilot);
11     }
12     finally
13     {
14         container.Close();
15     }
16 }
```

## InterLanguageExample2.vb: SaveObjects

```

01 Private Shared Sub SaveObjects()
02     File.Delete(Db4oFileName)
03     Dim container As IObjectContainer = Db4oFactory.OpenFile(Db4oFileName)
04     Try
05         Dim pilot As New Pilot("David Barri chello", 99)
06         container.[Set](pilot)
07         pilot = New Pilot("Michael Schumacher", 100)
08         container.[Set](pilot)
09     Finally
```

```

10 |           container.Close()
11 |       End Try
12 L   End Sub

```

In order to read the saved objects successfully from a java application we will need to define an alias for persistent classes and an alias for the Db4oDatabase class. We will use a wildcard alias for all the persistent classes:

InterLanguageExample2.java: configureAlias

```

01 private static Configuration configureAlias() {
02 |     Configuration configuration = Db4o.newConfiguration();
03 |     configuration.addAlias(new WildcardAlias(
04 |         "Db4obj ects. Db4odoc. Al i ases. *, Db4obj ects. Db4odoc",
05 |         "com. db4odoc. al i ases. *"));
06 |     configuration.addAlias(new TypeAl i as(
07 |         "Db4obj ects. Db4o. Ext. Db4oDatabase, Db4obj ects. Db4o",
08 |         "com. db4o. ext. Db4oDatabase"));
09 |     return configuration;
10 L }

```

Now the objects are accessible from the Java application:

InterLanguageExample2.java: getObjects

```

01 private static void getObjects(Configuration configuration) {
02 |     ObjectContainer db = Db4o.openFile(configuration, DB4O_FILE_NAME);
03 try {
04 |     List<Pilot> result = db.query(new Predicate<Pilot>() {
05 |         public boolean match(Pilot pilot) {
06 |             return true;
07 |         }
08 |     });
09 for (int i = 0; i < result.size(); i++) {
10 |     Pilot pilot = result.get(i);
11 |     pilot.setName("Modified " + pilot.getName());
12 |     db.set(pilot);
13 | }
14 |     listResult(result);
15 } finally {

```

```

16 |         db. close();
17 |     }
18 L }
```

One thing to remember: field names in class definitions in Java and .NET should be exactly the same.

We can read the database from the initial .NET application again. Note, that no alias is required as the class definitions were created in this application :

#### InterLanguageExample2.cs: ReadObjects

```

01 private static void ReadObjects()
02 {+
03     IObjectContainer container = Db4oFactory.OpenFile(Db4oFileName);
04     try
05     {
06         IList<Pilot> result = container.Query<Pilot>();
07         ListResult(result);
08     }
09     finally
10    {
11        container.Close();
12    }
13 }
```

#### InterLanguageExample2.vb: ReadObjects

```

1 Private Shared Sub ReadObjects()
2     Dim container As IObjectContainer = Db4oFactory.OpenFile(Db4oFileName)
3     Try
4         Dim result As IList(Of Pilot) = container.Query(Of Pilot)()
5         ListResult(result)
6     Finally
7         container.Close()
8     End Try
9 End Sub
```

This revision (1) was last Modified 2007-08-12T19:50:53 by Tetyana.

# Encryption

db4o provides simple built-in encryption functionality. This feature is easy to turn on or off, and must be configured before opening a database file.

In addition, db4o provides the ability for you to plug in your own encrypting IO Adapters.

More Reading:

- [Built-In Simple Encryption](#)
- [Custom Encryption Adapters](#)

This revision (6) was last Modified 2007-03-16T15:57:25 by Patrick Roemer.

# Built-In Simple Encryption

(This functionality is deprecated)

The other encryption methods built-into db4o, is called simple encryption. To use it, the following two methods have to be called, before a database file is created:

Java:

```
Db4o.configure().encrypt(true); Db4o.configure().password  
("yourEncryptionPasswordHere");
```

The security standard of the built-in encryption functionality is not very high, not much more advanced than "subtract 5 from every byte". This is great for systems with limited resources, or where the encryption needs to be done as quickly as possible.

This revision (7) was last Modified 2007-07-05T03:18:33 by German Viscuso.

# Custom Encryption Adapters

db4o still provides a solution for high-security encryption by allowing any user to choose his own encryption mechanism that he thinks he needs. The db4o file IO mechanism is pluggable and any fixed-length encryption mechanism can be added. All that needs to be done is to write an IoAdapter plugin for db4o file IO.

This is a lot easier than it sounds. Simply:

- take the sources of com.db4o.io.RandomAccessFileAdapter as an example
- write your own IoAdapter implementation that delegates raw file access to another adapter using the GoF decorator pattern.
- Implement the #read() and #write() methods to encrypt and decrypt when bytes are being exchanged with the file
- plug your adapter into db4o with the following method:

```
Java: Db4o.configure().io(new MyEncryptionAdapter());
```

However, you'll have to keep in mind that db4o will write partial updates. For example, it may write a full object and then only modify one field entry later on. Therefore it is not sufficient to en-/decrypt each access in isolation. You'll rather have to make up a tiling structure that defines the data chunks that have to be en-/decrypted together.

A community project containing an XTEA encryption IoAdapter implementation can be found here:

<http://developer.db4o.com/ProjectSpaces/view.aspx/XTEA>

Another method to inject encryption capabilities into db4o for instances of specific classes only is to implement and configure an en-/decrypting translator.

This revision (9) was last Modified 2007-03-19T11:16:24 by Tetyana.

# Class Mapping

[This functionality is deprecated]

db4o provides you with a possibility to create a mapping from a class in the database to a runtime class.

Java:

```
ObjectClass.readAs(Object clazz)
```

clazz parameter specifies a runtime class, which will be used to instantiate objects from the database.

The use-case is the following:

- objects of class A are stored to the database;
- the objects should be retrieved from the database and instantiated as objects of class B.

Java:

```
Db4o.configure().objectClass(A.class).readAs(B.class)
```

This configuration should be set before opening a database file.

The mapping functionality is similar to [Aliases](#), but more limited.

Let's look at an example.

We will use 2 identical classes [Pilot](#) and [PilotReplacement](#).

Objects of Pilot class will be saved to the database:

MappingExample.java: storeObjects

```
01 | private static void storeObjects() {
02 |     new File(FILENAME).delete();
03 |     ObjectContainer container = Db4o.openFile(FILENAME);
04 |     try {
05 |         Pilot pilot = new Pilot("Michael Schumacher", 100);
06 |         container.set(pilot);
```

```

07 |     pilot = new Pilot("Rubens Barichello", 99);
08 |     container.set(pilot);
09 | } finally {
10 |     container.close();
11 | }
12 | }
```

Let's try to retrieve the persisted objects using PilotReplacement class:

MappingExample.java: retrieveObjects

```

01 | private static void retrieveObjects() {
02 |     Configuration configuration = Db4o.newConfiguration();
03 |     configuration.objectClass(Pilot.class).readAs(PilotReplacement.class);
04 |     ObjectContainer container = Db4o.openFile(configuration, FILENAME);
05 | } try {
06 |     Query query = container.query();
07 |     query.constrain(PilotReplacement.class);
08 |     ObjectSet result = query.execute();
09 |     listResult(result);
10 | } finally {
11 |     container.close();
12 | }
13 | }
```

If meta information for this mapping class has been stored before to the database file, `readAs` method will have no effect.

This revision (5) was last Modified 2007-05-07T12:36:22 by Tetyana.

# Pilot

Pilot.java

```
01 /* Copyright (C) 2007 db4objects Inc. http://www.db4o.com */
02 package com.db4odoc.classmapping;
03
04 public class Pilot {
05     private String name;
06     private int points;
07
08     public Pilot(String name, int points) {
09         this.name=name;
10         this.points =points;
11     }
12
13     public String getName() {
14         return name;
15     }
16
17     public void addPoints(int points) {
18         this.points += points;
19     }
20
21     public String toString() {
22         return name + "/" + points;
23     }
24 }
```

This revision (1) was last Modified 2007-03-20T15:29:47 by Tetyana.

# PilotReplacement

PilotReplacement.java

```
01 /* Copyright (C) 2007 db4objects Inc. http://www.db4o.com */
02 package com.db4odoc.classmapping;
03
04 public class PilotReplacement {
05     private String name;
06     private int points;
07
08     public PilotReplacement(String name, int points) {
09         this.name=name;
10         this.points =points;
11     }
12
13     public String getName() {
14         return name;
15     }
16
17     public void addPoints(int points) {
18         this.points += points;
19     }
20
21     public String toString() {
22         return name + "/" + points;
23     }
24 }
```

This revision (1) was last Modified 2007-03-20T15:31:06 by Tetyana.

# IDs and UUIDs

The db4o team recommends not to use object IDs where it is not necessary. db4o keeps track of object identities in a transparent way, by identifying "known" objects on updates. The reference system also makes sure that every persistent object is instantiated only once, when a graph of objects is retrieved from the database, no matter which access path is chosen. If an object is accessed by multiple queries or by multiple navigation access paths, db4o will always return the one single object, helping you to put your object graph together exactly the same way as it was when it was stored, without having to use IDs.

The use of IDs does make sense when object and database are disconnected, for instance in stateless applications.

More Reading:

- [Internal IDs](#)
- [Unique Universal IDs](#)

This revision (7) was last Modified 2006-11-14T18:48:51 by Eric Falsken.

# Internal IDs

The internal db4o ID is a physical pointer into the database with only one indirection in the file to the actual object so it is the fastest external access to an object db4o provides. The internal ID of an object is available with

Java: `ObjectContainer.ext().getID(object)`

To get an object for an internal ID use

Java: `ObjectContainer.ext().getByID(id)`

Note that `#getByID()` does not activate objects. If you want to work with objects that you get with `#getByID()`, your code would have to make sure the object is [activated](#) by calling

Java: `ObjectContainer.activate(object, depth)`

db4o assigns internal IDs to any stored first class object. These internal IDs are guaranteed to be unique within one ObjectContainer/ObjectServer and they will stay the same for every object when an ObjectContainer/ObjectServer is closed and reopened. Internal IDs **will change** when an object is moved from one ObjectContainer to another, as it happens during [Defragment](#).

This revision (8) was last Modified 2007-05-07T12:40:29 by Tetyana.

# Unique Universal IDs

For long-term external references and to identify an object even after it has been copied or moved to another ObjectContainer, db4o supplies Unique Universal IDs (UUIDs).

Every newly created db4o database generates a signature object. It is stored as an instance of Db4oDatabase in your database file.

This signature is linked to all newly created objects (if UUID generation is enabled) as the "signature part" of the UUID.

Further to that db4o creates a timestamp for every new object and uses an internal counter to make sure that timestamps are unique. This is called the "long part" of the UUID.

The long part is indexed to make the search fast. If two objects with an identical long parts are found, the signature parts are compared also.

The long part of the UUID can also be used to find out when an object was created. You can use

Java:

```
com.db4o.foundation.TimeStampIdGenerator#idToMilliseconds()
```

to get object creation time in milliseconds.

UUIDs are guaranteed to be unique, if the signature of your db4o database is unique.

Normally any database has a unique signature unless its file is copied. The original and copied database files are identical, so they have the same signatures. If such files are used in replication, the process will end up with exceptions. What is the solution then?

Signature of a database file can be changed using

Java:

```
LocalObjectContainer#generateNewIdentity
```

method.

**UUIDExample.java: testChangeIdentity**

```
01 private static void testChangeIdentity() {
02 |     new File(DB4O_FILE_NAME).delete();
03 |     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
```

```
04 |     Db4oDatabase db;
05 |     byte[] oldSignature;
06 |     byte[] newSignature;
07 |     try {
08 |         db = container.ext().identity();
09 |         oldSignature = db.getSignature();
10 |         System.out.println("oldSignature: "
11 |             + printSignature(oldSignature));
12 |         ((LocalObjectContainer) container).generateNewIdentity();
13 |     } finally {
14 |         container.close();
15 |     }
16 |     container = Db4o.openFile(DB4O_FILE_NAME);
17 |     try {
18 |         db = container.ext().identity();
19 |         newSignature = db.getSignature();
20 |         System.out.println("newSignature: "
21 |             + printSignature(newSignature));
22 |     } finally {
23 |         container.close();
24 |     }
25 |
26 |     boolean same = true;
27 |
28 |     for (int i = 0; i < oldSignature.length; i++) {
29 |         if (oldSignature[i] != newSignature[i]) {
30 |             same = false;
31 |         }
32 |     }
33 |
34 |     if (same) {
35 |         System.out.println("Database signatures are identical");
36 |     } else {
37 |         System.out.println("Database signatures are different");
38 |     }
```

39 L }

UUIDs are not generated by default, since they occupy extra space in the database file and produce performance overhead for maintaining their index. UUIDs can be turned on globally or for individual classes:

Java: `configuration.generateUUIDs(ConfigScope.GLOBALLY)`

- turns on UUID generation for all classes in a database.

Java: `configuration.objectClass(Foo.class).generateUUIDs(true)`

- turns on UUID generation for a specific class.

You can get the UUID value for an object using the following methods:

Java:

`ExtObjectContainer#getObjectInfo(Object)`  
`ObjectInfo#getUUID()`

To get the object from the database, knowing its UUID, use:

Java:

`ExtObjectContainer#getByUUID(Db4oUUID)`

The following example shows the usage of UUID:

UUIDExample.java: `setObjects`

```

01  private static void setObjects() {
02 |     new File(DB4O_FILE_NAME).delete();
03 |     Configuration configuration = Db4o.newConfiguration();
04 |     configuration.objectClass(Pilot.class).generateUUIDs(true);
05 |     ObjectContainer container = Db4o.openFile(configuration, DB4O_FILE_NAME);
06  ↘    try {
07 |     Car car = new Car("BMW", new Pilot("Rubens Barrichello"));
08 |     container.set(car);
09  ↘    } finally {
10 |     container.close();
11  ↘    }

```

12 }

## UUIDExample.java: testGenerateUUID

```
01|private static void testGenerateUUID() {
02|    ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
03|    try {
04|        Query query = container.query();
05|        query.constrain(Car.class);
06|        ObjectSet result = query.execute();
07|        Car car = (Car) result.get(0);
08|        ObjectInfo carInfo = container.ext().getObjectContext(car);
09|        Db4oUUID carUUID = carInfo.getUUID();
10|        System.out
11|            .println("UUID for Car class are not generated:");
12|        System.out.println("Car UUID: " + carUUID);
13|
14|        Pilot pilot = car.getPilot();
15|        ObjectInfo pilotInfo = container.ext().getObjectContext(
16|            pilot);
17|        Db4oUUID pilotUUID = pilotInfo.getUUID();
18|        System.out.println("UUID for Pilot:");
19|        System.out.println("Pilot UUID: " + pilotUUID);
20|        System.out.println("long part: "
21|            + pilotUUID.getLongPart() + "; signature: "
22|            + printSignature(pilotUUID.getSignaturePart()));
23|        long ms = TimeStampIdGenerator.idToMilliseconds(pilotUUID
24|            .getLongPart());
25|        System.out.println("Pilot object was created: "
26|            + (new Date(ms)).toString());
27|        Pilot pilotReturned = (Pilot) container.ext().getByUUID(
28|            pilotUUID);
29|        System.out.println("Pilot from UUID: " + pilotReturned);
30|    } finally {
31|        container.close();
```

32 | }  
33 L }

Sometimes you can find out that you need UUIDs only when the database is already created and has some data in it. What can you do in that case?

Fortunately enabling replication for existing data files is a very simple process:

Java:

```
configuration.objectClass(Task.class).enableReplication(true)
```

After that you will just need to use the old defragment tool from tools package supplied with the distribution before version 6.0 (source code only) to enable replication.

You can use UUID for replication and as a reference to a specific object instance from an external application or data store.

This revision (17) was last Modified 2008-02-05T12:05:58 by dlouwers.

# Freespace Management System

Db4o Freespace manager is a special system that is responsible for allocating, discarding, merging of space for db4o usage and keeping database size at minimum

Db4o organizes its files into variable-sized slots with one degree of indirection (pointer slots). The physical address of a pointer slot corresponds to the internal ID of the respective object. On the lowest level the database file is seen as a uniform pool of bytes, from which slots of variable size can be allocated.

Freespace manager keeps track of the portions of the database file that haven't been allocated for use and gets notified when slots are 'freed', i.e. returned back to the pool.

That is how it works:

- All objects are written to the database file immediately, when they are stored or updated.
- Updated objects get a new or previously freed place in the database file.
- New or modified pointers to new or modified objects are stored in RAM.
- Old and deleted objects are marked as 'freespace'.

In general case each persisted object occupies one slot. This slot contains metadata, values of direct primitive members and references to the slots of non-primitive members. What actually goes directly into a slot and what is external is not fixed and differs from version to version. However, there is one level of indirection: a reference to a non-primitive member will not refer to the member's slot directly, but rather to the 'pointer', which is an 8 byte slot containing the address in the file and the data length. The address of this pointer slot is the object's internal ID that is used for indexing, etc.

For a simple example, assume we have

```
class Car { String manufacturer; }

class Driver { String name; Car car; }
```

and an object graph like this

```
Driver: name='Barrichello', car={Car: manufacturer='BMW'}
```

This will translate into four slots

```
1234: [4711, length(Driver)]
4711: [Driver,'Barrichello',0815]
0815: [4321, length(Car)]
4321: [Car,'BMW']
```

with 1234 being the db4o ID of the driver and 0815 the ID of the car. Whenever the driver or the car object is updated, its actual slot may be stored somewhere else, but the pointer slot (the ID) will remain the same and keep track of the slot's address. (Please, note that this is a simplified example: actual implementation uses more slots and more sophisticated processing).

More Reading:

- [Two Freespace Systems](#)
- [How To Use FreeSpacemanager](#)
- [Defragmentation Role](#)

This revision (7) was last Modified 2007-10-10T15:46:35 by Tetyana.

# Two Freespace Systems

db4o comes with three freespace systems:

- RAM-based: the information about freespace is held in RAM;
- b-Tree-based: the information about freespace is written to disk, b-Trees are used to manage this information;
- index-based: similar to b-Trees, but existing index functionality to store freespace information.

You can configure db4o to use either of these by calling

Java: `configuration.freespace().useRamSystem()`

or

Java: `configuration.freespace().useBTreeSystem()`

or

Java: `configuration.freespace().useIndexSystem()`

This call should be made before you open the database for the first time

By default db4o uses **RAM freespace management system**. The information about free slots is loaded into memory on opening a database file and discarded on closing it. This system is quite fast, but it has its downside:

1. Higher RAM usage during operation.
2. Loss of freespace upon abnormal termination. That is done for security reasons and freespace can be reclaimed using defragmentation.

RAM-based freespace management is a good performance solution, but it can be insufficient for the systems with limited RAM resources and high probability of abnormal system termination (power failure on mobile devices).

In order to meet the requirements of such environments you can use new **b-Tree-based freespace management system**. It solves the problems of RAM-based system:

1. RAM usage is kept at the minimum.
2. No freespace is lost on abnormal system termination (database file won't grow unnecessarily).

How it works?:

- The system uses b-Trees to keep information about available freespace
- b-Trees operate against the file, and only uses memory for caching
- For every new write to the database file the system tries to find a freed slot, which is at least the size needed or greater, traversing freespace index
- When an object is updated or deleted, its 'old' slot is added to the freespace b-Tree entry
- This b-Tree system is ACID (no information is lost upon abnormal system termination)

b-Tree-based freespace system can show poorer performance compared to RAM-based system, as it needs to access the file to write updated freespace information.

However, b-Tree-based freespace system is fast enough, especially for mobile devices, where file access is not much slower than RAM-access, and ACID transactions together with low memory consumption are most valuable factors.

Index-based freespace system has similar to b-Tree characteristics, but poorer performance and is used for legacy reasons.

This revision (9) was last Modified 2007-10-21T18:34:18 by Tetyana.

# How To Use FreeSpacemanager

There are several configuration options that can help you to tune up your freespacemanager to achieve the best performance and reliability of your system. All methods should be called before opening database files.

Public interface FreespaceConfiguration provides methods to select freespace system (*useIndexSystem()*, *useRamSystem()*) as described before. See API documentation for more information.

Another FreespaceConfiguration method

Java:

```
void discardSmallerThan(int byteCount)
```

configures the minimum size of free space slots in the database file that are to be reused. FreespaceManager keeps 2 lists of all 'freed' space that can be reused (sorted by address and by size). In some cases (numerous updates, deletes) these lists can grow large, causing extra RAM consumption and performance loss for maintenance. With this method you can specify an upper bound for the byte slot size to discard from Freespace manager list. It is not recommended to specify a value of byteCount > 100 as freespace re-usage will become less efficient and the database file will grow faster. However, if defragment can be run frequently, it will also reclaim lost space and decrease the database file to the minimum size. Therefore byteCount may be set to bigger value.

By default byteCount = 0, which means that all 'freed' space is reused.

Another configuration setting that can be used with frequently defragmented systems

Java:

```
configuration.automaticShutDown(false);
```

Detailed description of this method can be found in [Tuning](#) part of the Reference documentation.

This revision (15) was last Modified 2007-05-07T12:44:22 by Tetyana.

# Defragmentation Role

So we have a reliable freespace manager, which will keep the database file size to the minimum. But do we still need to bother about [defragmentation](#)?

Yes, we do for several reasons:

- When the object is deleted, its space in the database is marked as 'freed'. But 8 bytes of its internal ID stay behind. Defragment cleans all this up by writing all objects to a completely new database file. The resulting file will be smaller and faster.
- Within the database file quite a lot of space is used for transactional processing. Objects are always written to a new slot when they are modified, while their 'old' space is marked as free. Adjacent free slots will be merged on the fly, but free and used slots won't be clustered together unless defragmentation is run.

This revision (6) was last Modified 2007-05-07T12:47:01 by Tetyana.

# Reporting

db4o has been tested with several enterprise reporting systems:

.NET:

- [Microsoft Visual Studio 2005 - ReportViewer](#)

Java:

- [Actuate BIRT](#)
- [Elixir Report](#)
- [Jaspersoft JasperReports](#)
- [Jinfonet JReport](#)

More Reading:

- [Reporting With JasperReports](#)
- [Reporting With BIRT](#)
- [.NET Reporting](#)

This revision (2) was last Modified 2007-02-16T18:20:45 by Tetyana.

# Reporting With JasperReports

[JasperReports](#) is an open-source java-based reporting system. JasperReports can be easily integrated with any Java desktop application and has the ability to deliver rich content onto the screen, to the printer or into PDF, HTML, XLS, CSV and XML files.

More information about JasperReports including tutorial, reference and examples, can be obtained from its [official web-site](#).

This chapter will help you to start using JasperReports with your db4o database.

Before you proceed, you will need to [download](#) JasperReports. Check the full list of system requirements [here](#). For the examples discussed in this chapter you will only need to ensure that:

- you have JRE5 or higher installed;
- you have the following apache jars: commons-digester.jar, commons-collections.jar, commons-logging.jar, commons-beanutils.jar;
- you have iText library version 1.3 or higher.

It is recommended to use [iReport](#) designer with JasperReports for an easy visual report design, however it is not required for this tutorial.

More Reading:

- [Report Structure](#)
- [Data Source](#)
- [Data Preparation](#)
- [Report Design](#)
- [Report Generation](#)
- [ObjectDataSource](#)

This revision (1) was last Modified 2007-04-27T16:33:50 by Tetyana.

# Report Structure

Report design is defined in an xml file with a conventional extension \*.jrxml. The xml structure is defined in a DTD file and can be downloaded from the [sourceforge](#).

A simplest table report definition can look like this:

simple\_report.xml

```

01 <?xml version="1.0"?>
02 <!DOCTYPE jasperReport
03 PUBLIC "-//JasperReports//DTD Report Design//EN"
04 "http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
05
06 <jasperReport name="Simple_Report">
07 <field name="Name" class="java.lang.String"/>
08 <detail>
09 <band height="20">
10 <textField bookmarkLevel="2">
11 <reportElement x="0" y="0" width="100" height="15"/>
12 <textFieldExpression class="java.lang.String">$F{Name}</textFieldExpression>
13 </textField>
14 </band>
15 </detail>
16 </jasperReport>
```

For a simple example, we will use the following Jasper xml elements:

**parameter** - represents the definition of a report parameter.

In jrxml file:

```
<parameter name="Title" class="java.lang.String"> </parameter>
```

In \*.java file:

```
Map parameters = new HashMap();
```

```
parameters.put("Title", "The Pilot Report");
```

parameters map is further passed to the [report filling function](#).

**field** - represents the definition of a data field that will store values retrieved from the data source of the report.

In \*.jrxml file:

```
<field name="Name" class="java.lang.String"/>
```

In \*.java file fields are handled by classes implementing [JRDataSource](#) interface.

For the full description of JasperReports xml elements please refer to the [Reference](#).

This revision (2) was last Modified 2007-04-27T16:37:49 by Tetyana.

# Data Source

JasperReports can be built on any data having visual representation. In order to make the data "understandable" to the report object, it should be supplied through a JRDataSource interface. The JasperReports package supplies several implementations, which can be used with a RDBMS, xml, csv and object data sources.

In order to represent db4o objects in the most convenient way, we will build a special JRDataSource implementation - ObjectDataSource - using reflection to obtain object field values.

ObjectDataSource will accept data as a list of objects, because this is the way it is returned from a db4o query:

```
List <Pilot> pilots = objectContainer.query(pilotPredicate);
```

ObjectDataSource.java: ObjectDataSource

```
01 /**
02 | * ObjectDataSource class is used to extract object field values for the report.
03 | * <br><br>
04 | * usage: <br>
05 | * List pilots = ...<br>
06 | * ObjectDataSource dataSource = new ObjectDataSource(pilots);<br>
07 | * In the report (*.jrxml) you will need to define fields. For example: <br>
08 | *   <field name="Name" class="java.lang.String"/><br>
09 | * where field name should correspond to your getter method:<br>
10 | *   "Name" - for getName()<br>
11 | *   "Id" - for getId()<br>
12 | *
13 L */
14 public class ObjectDataSource implements JRDataSource {
15 |
16 |   private Iterator iterator;
17 |
18 |   private Object currentValue;
19 |
20 public ObjectDataSource(List list) {
21 |     this.iterator = list.iterator();
22 L }
```

ObjectDataSource must implement 2 methods:

```
public boolean next()
```

and

```
public Object getFieldValue(JRField field)
```

The `next()` implementation is very simple: it just moves the current pointer to the next object in the list:

ObjectDataSource.java: `next`

```
1 | public boolean next() throws JRException {  
2 |     currentValue = iterator.hasNext() ? iterator.next() : null;  
3 |     return (currentValue != null);  
4 | }
```

`getFieldValue` method should return the value for the specified field. The field is defined in [\\*.jrxml](#) file and is passed to the `JRDataSource` as a `JRField`. In the case of an object list datasource the objective is to correspond field names to the object field values. One of the ways to do this is to correspond the name of the field in the report to the name of the getter method in the object class. For example:

```
<field name="Name" class="java.lang.String"/>

class Pilot

{

    ...

    public String getName(){

        return name;

    }

}
```

The method name is "get" + `JRField#getName()` or "get" + "Name". Knowing the method name, we can invoke it using reflection and obtain the value of the object field:

ObjectDataSource.java: `getFieldValue`

```
01 | public Object getFieldValue(JRField field) throws JRException {  
02 |     Object value = null;  
03 |     try {  
04 |         // getter method signature is assembled from "get" + field name  
05 |         // as specified in the report  
06 |         Method fieldAccessor = currentValue.getClass().getMethod("get" + field.getName(),  
07 |             null);  
07 |         value = fieldAccessor.invoke(currentValue, null);
```

```
08 } catch (IllegalAccessException iae) {  
09 |     iae.printStackTrace();  
10 } catch (InvocationTargetException ite) {  
11 |     ite.printStackTrace();  
12 } catch (NoSuchMethodException nsme) {  
13 |     nsme.printStackTrace();  
14 }  
15 |     return value;  
16 }
```

The full code of the class can be downloaded from [ObjectDataSource](#).

This revision (1) was last Modified 2007-04-27T16:40:03 by Tetyana.

# Data Preparation

Before the report can be generated, we will need to store some data to our database. We will use the following Pilot class:

Pilot.java

```
01 public class Pilot {  
02     private String name;  
03  
04     private int points;  
05  
06     public Pilot(String name, int points) {  
07         this.name = name;  
08         this.points = points;  
09     }  
10  
11     public String getName() {  
12         return name;  
13     }  
14  
15     public void setName(String name) {  
16         this.name = name;  
17     }  
18  
19     public int getPoints() {  
20         return points;  
21     }  
22  
23     public boolean equals(Object obj) {  
24         if (obj instanceof Pilot) {  
25             return (((Pilot) obj).getName().equals(name) &&  
26                     ((Pilot) obj).getPoints() == points);  
27     }  
28 }
```

```

27 |     }
28 |     return false;
29 | }
30 |
31 |     public String toString() {
32 |         return name + "/" + points;
33 |     }
34 |
35 |     public int hashCode() {
36 |         return name.hashCode() + points;
37 |     }
38 | }

```

Pilot class has `name` and `points` fields, which can be obtained through `getName()` and `getPoints()` methods.

Let's store some pilots to the database:

#### JasperReportsExample.java: storePilots

```

01 |     private static void storePilots() {
02 |         new File(DB40_FILE_NAME).delete();
03 |         ObjectContainer container = database();
04 |         if (container != null) {
05 |             try {
06 |                 Pilot pilot;
07 |                 for (int i = 0; i < OBJECT_COUNT; i++) {
08 |                     pilot = new Pilot("Test Pilot #" + i, i);
09 |                     container.set(pilot);
10 |                 }
11 |                 for (int i = 0; i < OBJECT_COUNT; i++) {
12 |                     pilot = new Pilot("Professional Pilot #" + (i + 10), i + 10);
13 |                     container.set(pilot);
14 |                 }

```

```
15 |         container.commit();  
16 |     } catch (Db4oException ex) {  
17 |         System.out.println("Db4o Exception: " + ex.getMessage());  
18 |     } catch (Exception ex) {  
19 |         System.out.println("System Exception: " + ex.getMessage());  
20 |     } finally {  
21 |         closeDatabase();  
22 |     }  
23 | }  
24 }
```

This revision (2) was last Modified 2007-04-27T16:44:38 by Tetyana.

# Report Design

Let's create a report representing pilots in a table. The table will have 2 fields: "Name" and "Points". The fields should be defined like this:

```
<field name="Name" class="java.lang.String"/>

<field name="Points" class="java.lang.Integer"/>
```

The field values - \${Name} and \${Points} - can be used in the table part of the report:

```
<detail>

<band height="15">

    <textField bookmarkLevel="2">

        <reportElement x="150" y="0" width="175" height="15"/>

        <box leftBorder="Thin" bottomBorder="Thin" leftPadding="10"
rightPadding="10"/>

        <textElement textAlign="Left" />

        <textFieldExpression class="java.lang.String">${Name}</
textFieldExpression>

        <anchorNameExpression>${Name} + " (" + ${Points} + ")"</
anchorNameExpression>

    </textField>

    <textField isStretchWithOverflow="true">

        <reportElement positionType="Float" x="325" y="0" width="50"
height="15"/>

        <box leftBorder="Thin" bottomBorder="Thin" rightBorder="Thin"
leftPadding="10" rightPadding="10"/>

        <textElement textAlign="Right" />

    </textField>
</band>
</detail>
```

## Report Design

```
<textFieldExpression class="java.lang.Integer">$F{Points}</
textFieldExpression>

</textField>

</band>

</detail>
```

The full report design can be downloaded from [the-pilot-report.jrxml](#).  
This revision (2) was last Modified 2007-04-27T16:49:22 by Tetyana.

# Report Generation

Now, when the [objects](#), [data source class](#) and the [report design](#) are ready, it is very easy to generate the report and save it to pdf, html or display on the screen:

JasperReportsExample.java: pilotsReport

```
01  public static void pilotsReport() {  
02 |  
03 |     // obtain a list of objects for the report  
04 |     List<Pilot> pilots = database().query(new Predicate<Pilot>() {  
05 |         public boolean match(Pilot pilot) {  
06 |             // each Pilot is included in the result  
07 |             return true;  
08 |         }  
09 |     });  
10 |  
11 |     // pass parameters to the report  
12 |     Map parameters = new HashMap();  
13 |     parameters.put("Title", "The Pilot Report");  
14 |  
15 |     try {  
16 |         // compile report design  
17 |         JasperReport jasperReport = JasperCompileManager  
18 |             .compileReport("reports/the-pilot-report.jrxml");  
19 |  
20 |         // create an object datasource from the pilots list  
21 |         ObjectDataSource dataSource = new ObjectDataSource(pilots);  
22 |  
23 |         // fill the report  
24 |         JasperPrint jasperPrint = JasperFillManager.fillReport(  
25 |             jasperReport, parameters, dataSource);  
26 |     }
```

```
27 |         // export result to the *.pdf
28 |         JasperExportManager.exportReportToPdfFile(jasperPrint,
29 |             "reports/the-pilot-report.pdf");
30 |
31 |         // or export to *.html
32 |         JasperExportManager.exportReportToHtmlFile(jasperPrint,
33 |             "reports/the-pilot-report.html");
34 |
35 |         // or view it immediately in the Jasper Viewer
36 |         JasperViewer.viewReport(jasperPrint);
37 |     } catch (JRException e) {
38 |         e.printStackTrace();
39 |     }
40 | }
```

This revision (1) was last Modified 2007-04-27T16:52:49 by Tetyana.

# ObjectDataSource

ObjectDataSource.java

```
01 /* Copyright (C) 2007 db4objects Inc. http://www.db4o.com */  
02 import java.lang.reflect.InvocationTargetException;  
03 import java.lang.reflect.Method;  
04 import java.util.Iterator;  
05 import java.util.List;  
06  
07 import net.sf.jasperreports.engine.JRDataSource;  
08 import net.sf.jasperreports.engine.JRException;  
09 import net.sf.jasperreports.engine.JRFIELD;  
10  
11 /**  
12 * ObjectDataSource class is used to extract object field values for the report.  
13 * <br><br>  
14 * usage: <br>  
15 * List pilots = ... <br>  
16 * ObjectDataSource dataSource = new ObjectDataSource(pilots); <br>  
17 * In the report (*.jrxml) you will need to define fields. For example: <br>  
18 * <field name="Name" class="java.lang.String"/><br>  
19 * where field name should correspond to your getter method: <br>  
20 * "Name" - for getName()<br>  
21 * "Id" - for getId()<br>  
22 *  
23 */  
24 public class ObjectDataSource implements JRDataSource {  
25 |  
26 | private Iterator iterator;  
27 |  
28 | private Object currentValue;  
29 |  
30 public ObjectDataSource(List list) {  
31 | this.iterator = list.iterator();  
32 |}  
33 // end ObjectDataSource  
34 |  
35 public Object getFieldValue(JRFIELD field) throws JRException {
```

```
36 |     Object value = null;
37 |     try {
38 |         // getter method signature is assembled from "get" + field name
39 |         // as specified in the report
40 |         Method fieldAccessor = currentValue.getClass().getMethod("get" + field.getName(),
41 |                                         null);
42 |         value = fieldAccessor.invoke(currentValue, null);
43 |     } catch (IllegalAccessException iae) {
44 |         iae.printStackTrace();
45 |     } catch (InvocationTargetException ite) {
46 |         ite.printStackTrace();
47 |     } catch (NoSuchMethodException nsme) {
48 |         nsme.printStackTrace();
49 |     }
50 |     return value;
51 | }
52 |
53 |     public boolean next() throws JRException {
54 |         currentValue = iterator.hasNext() ? iterator.next() : null;
55 |         return (currentValue != null);
56 |     }
57 | }
58 |
59 }
```

This revision (1) was last Modified 2007-04-27T16:41:11 by Tetyana.

# Reporting With BIRT

BIRT is an Eclipse-based open source reporting system. It can be used for web and desktop applications. BIRT can be integrated with db4o by using script DataSource.

The following topics present the most basic example of db4o-BIRT integration.

Before you proceed you will need to install [BIRT engine](#) and [Eclipse BIRT designer plugin](#).

For more information on the BIRT functionality and java integration API refer to the [BIRT official site](#) and [BIRT Integration Reference](#).

More Reading:

- [Data Preparation](#)
- [Report Design](#)
- [Scripted Data Source](#)
- [Application-Report Integration](#)

Full source code: [ReportsBIRT.zip](#)

This revision (3) was last Modified 2007-05-07T12:55:41 by Tetyana.

# Data Preparation

First of all we will need to prepare some data for the report. Let's use a simple Pilot class:

## Pilot.java

```
01 public class Pilot {  
02     private String name;  
03     private int points;  
04  
05     public Pilot(String name, int points) {  
06         this.name = name;  
07         this.points = points;  
08     }  
09  
10    public String getName() {  
11        return name;  
12    }  
13  
14    public int getPoints() {  
15        return points;  
16    }  
17  
18 }
```

The following method should be called to fill up the database with some sample data:

## Db4oModule.java: storeData

```
01 public void storeData() {  
02     new File(DB_FILE).delete();  
03     ObjectContainer container = Db4o.openFile(DB_FILE);  
04     try {
```

```

05 |     Pilot pilot = new Pilot("Michael Schumacher", 100);
06 |     container.set(pilot);
07 |     pilot = new Pilot("Rubens Barrichello", 99);
08 |     container.set(pilot);
09 |     pilot = new Pilot("Kimi Raikkonen", 100);
10 |     container.set(pilot);
11 | } finally {
12 |     container.close();
13 | }
14 |

```

For the table representation we will need a list of values:

#### Db4oModule.java: readData

```

01 | public List readData() {
02 |     ObjectContainer container = Db4o.openFile(DB_FILE);
03 |     List result = new ArrayList();
04 |     try {
05 |         ObjectSet pilots = container.query(Pilot.class);
06 |         while (pilots.hasNext()) {
07 |             Pilot pilot = (Pilot) pilots.next();
08 |             result.add(new String[] { pilot.getName(),
09 |                 String.format("%3d", pilot.getPoints()) });
10 |         }
11 |     } finally {
12 |         container.close();
13 |     }
14 |     return (result);
15 |

```

This revision (1) was last Modified 2007-02-16T18:33:16 by Tetyana.

# Report Design

Open Report Design perspective (Window/Open Perspective/Report Design). Create a new report (select the project in the Navigator window, right-click, select New/Report). In the New Report window change the default filename to Pilots.rptdesign and press "Finish".

Report designer layout window opens.

Open Palette view and drag "table" icon onto the report layout. In the "Insert Table" window change number of columns to 2 and click "OK".

On the next step we will need to specify a DataSource for the report. Open Data Explorer view, right-click "Data Sources", select "New Data Source". In order to deal with the object data we will need to use a "Scripted Data Source". This data source does not accept any other parameters, so you can just click "Finish".

Now we can define a data set for our table. Select "Data Sets"/"New Data Set" from the drop-down menu. Change the data set name to "Pilot". In the next window define 2 columns: "Name" and "Points", and press "Finish".

Now we have data that can be used for the layout. Open report layout window again and drag Pilot.Name and Pilot.Points columns from the Data Explorer/Data Sets onto "Detail Row" on the table.

This revision (1) was last Modified 2007-02-16T18:36:25 by Tetyana.

# Scripted Data Source

At this point we have created the report layout and we've filled in the data. Now we need to "teach" the report to understand our data source. We can do this using a script.

Open Pilots.rptdesign view and select "Script" tag.

In the Data Explorer window select "Pilot" dataset. Select "open" script from the drop-down menu in the script window. Add the following script:

```
dataClass = new Packages.Db4oModule();

pilots = dataClass.readData( );

totalrows = pilots.size( );

currentrow = 0;
```

This script will prepare the data when the report is opened. Save the changes.

Select "fetch" script from the drop-down menu. Add the following script:

```
row["Pilot"] = currentrow;

row["Points"] = totalrows;

if (currentrow >= totalrows){

    return(false);

}

var pilotsRow = pilots.get(currentrow);

var Pilot = pilotsRow[0];

var Points = pilotsRow[1];

row["Pilot"] = Pilot;

row["Points"] = Points;
```

## Scripted Data Source

```
currentrow = currentrow + 1;  
  
return(true);
```

This script fetches the data for the rows in the table.

This revision (1) was last Modified 2007-02-16T18:37:46 by Tetyana.

# Application-Report Integration

The last step is to integrate the existing report into our java application. This is done with the help of [Report Engine API](#).

Add the libraries from BIRT\_HOME/lib to the project. Create the following method in your main application class:

ReportMain.java: main

```

01 public static void main(String[] args) throws BirtException {
02     // create engine configuration
03     EngineConfig config = new EngineConfig();
04     // specify the path to the BIRT installation
05     config.setEngineHome(BIRT_HOME);
06     // set the output directory for the logging and set the logging level
07     config.setLogConfig(OUTPUT_DIR, Level.FINE);
08
09     // start the Platform to load the plugins
10    Platform startup(config);
11    // create a factory, which will supply ReportEngine instance
12    IReportEngineFactory factory = (IReportEngineFactory) Platform
13        .createFactoryObject(IReportEngineFactory.EXTENSION_REPORT_ENGINE_FACTORY);
14    // there is a significant cost associated with creating a new engine
15    // instance, therefore the application should use only one engine instance
16    // for its reports
17    engine = factory.createReportEngine(config);
18    engine.changeLogLevel(Level.WARNING);
19
20    IReportRunnable design;
21    // Open the report design
22    design = engine.openReportDesign(REPORT_DESIGN);
23    // Create task to run and render the report
24    IRunAndRenderTask task = engine.createRunAndRenderTask(design);
25
26    // Set rendering options - such as file or stream output,
27    // output format, whether it is embeddable, etc
28    HTMLRenderOption options = new HTMLRenderOption();
29    options.setOutputFileName(OUTPUT_DIR + "//" + REPORT_OUTPUT);
30    // output format can be either html or pdf

```

```
31 |     options.setOutputFormat("html");
32 |     task.setRenderOption(options);
33 |
34 |     // run the report
35 |     task.run();
36 |     task.close();
37 |
38 |     // shutdown the engine and the platform after use
39 |     // to clean up and unload the extensions
40 |     engine.shutdown();
41 |     Platform.shutdown();
42 | }
```

Now you can run the report and check the results in the OUTPUT\_DIR folder.

This revision (1) was last Modified 2007-02-16T18:40:55 by Tetyana.

# .NET Reporting

Microsoft Visual Studio 2005 provides report design functionality and ReportViewer controls allowing you to add full-featured reports to your WinForms and Web applications. Microsoft reports support object data sources making it easy to integrate reporting functionality with db4o.

The following example presents the simplest report based on db4o persistent objects. Please, note, that the suggested implementation is provided merely to show a way of using Microsoft.Reporting for db4o-based reports and is not supposed to be a complete reporting reference.

The example is based on a WinForms application, however you can use the same instructions to add reports in your db4o Web application.

More Reading:

- [Persistent Class](#)
- [Adding A Report](#)
- [Db4o Integration](#)
- [Report Integration](#)
- [Db4o Manager](#)

Full source code: [ReportsCS.zip](#), [ReportsVB.zip](#)

This revision (4) was last Modified 2007-05-07T13:00:20 by Tetyana.

# Persistent Class

This topic applies to .NET version only

First of all you will need to define an object class, which will be used for the report. Windows.Reporting uses object attributes as field values. Let's use the following class for an example:

The Pilot class has Name and Points attributes. These attributes can be used as report fields. If you need more fields to be present in your report you can specify additional attributes in your class or create a custom query class, which can combine information from several objects.

The Pilot class should be built before it will become accessible for reporting.

This revision (3) was last Modified 2007-05-23T18:16:38 by Tetyana.

# Adding A Report

This topic applies to .NET version only

Add a new report to your project by selecting Project/Add New Item/Report. Name it PilotReport.

You will need to specify DataSources for the report. Go to Data/Add New Data Source menu. This will launch Data Configuration Wizard. Select "Object" as an object type, click "Next" and select the Pilot class.

Open PilotReport in design mode and drag table element from the toolbox onto the report. Open Data Sources window and drag "Name" and "Points" attributes of the Pilot data source into the "Detail" row of the table.

Delete the third unused column from the table.

You can use the designer view to improve the report appearance: add a title, customize fonts, colors, borders etc.

This revision (2) was last Modified 2007-05-23T18:17:38 by Tetyana.

# Db4o Integration

This topic applies to .NET version only

We will use a separate module to maintain a database connection and perform db4o operations: [Db4o Manager](#).

The following function will add several Pilot objects to the database:

Report's data source can accept a single object (Textbox control) or a list of objects (Table, List, Chart etc.). In this example we use a table, so a list of objects is needed:

This function will return a list of pilot object IDs. Please, note, that the actual objects will be instantiated only as they are being retrieved from the list, which means that db4o connection should be kept open while the report is being created.

This revision (3) was last Modified 2007-05-23T18:15:07 by Tetyana.

# Report Integration

This topic applies to .NET version only

Now the existing report should be integrated into the application.

Select the application form (Form1) in design view and drag ReportViewer icon from the toolbox onto the form.

Select the ReportViewer control, and open the smart tags panel by clicking the triangle on the top right corner. Click the "Choose Report" drop-down list and select the PilotReport. A BindingSource (PilotBindingSource) is automatically created corresponding to the Pilot data source used in the report.

Open the code view for the Form1. Find Form1\_Load function and add the following code:

Db4oManager takes care of opening db4o connection on the first request. However the connection should be closed manually on the application termination:

This revision (3) was last Modified 2007-05-23T18:14:12 by Tetyana.

# Db4o Manager

This topic applies to .NET version only

This revision (3) was last Modified 2007-05-23T18:12:56 by Tetyana.

# Exceptions

Db4o versions prior to 6.2 were designed to silently "swallow" most of the exceptions. This mode proved to be efficient in the situations, when db4o is supposed to work without interruptions under any circumstances.

However there are many situations when it is vitally important for the user to know what went wrong and to be able to react accordingly. Some of the examples:

- callbacks;
- opening database file or client connection;
- unsupported schema changes;
- invalid ID format;
- corrupted database file, etc

The new approach introduced in the db4o 6.2 allows exceptions to "bubble up" to the user level.

More Reading:

- [Exception Types](#)
- [How To Work With Db4o Exceptions](#)

This revision (1) was last Modified 2007-03-27T13:34:36 by Tetyana.

# Exception Types

Using db4o you will have to deal with db4o-specific exceptions and system exceptions thrown directly out of db4o (like OutOfMemory error in Java or System.Exception in .NET).

Db4o-specific exceptions are Unchecked exceptions, which all inherit from a single root class Db4oException.

In Java Unchecked exceptions are inherited from RuntimeExceptions class, while in .NET all exceptions are unchecked.

Db4o exceptions are chained; you can get the cause of the exception using:

Java:

```
db4oException.getCause();
```

In order to see all db4o-specific exceptions you can examine the hierarchy of Db4oException class. Currently the following exceptions are available:

**Db4oException** - db4o exception wrapper: exceptions occurring during internal processing will be proliferated to the client calling code encapsulated in an exception of this type.

**BackupInProgressException** - an exception to be thrown when another process is already busy with the backup.

**ConstraintViolationException** - base class for all constraint exceptions.

**UniqueFieldValueConstraintViolationException** - an exception to be used to determine constraint violation on commit.

**DatabaseClosedException** - an exception to be thrown when the database was closed or failed to open.

**DatabaseFileLockedException** - this exception is thrown during any of db4o open calls if the database file is locked by another process.

**DatabaseMaximumSizeReachedException** - this exception is thrown if the database size is bigger than possible.

**DatabaseReadOnlyException** - an exception to be thrown when a write operation was attempted on a database in read-only mode.

**GlobalOnlyConfigException** - this exception is thrown when a global-only configuration setting is attempted for the local configuration.

**IncompatibleFormatException** - an exception to be thrown when an open operation is attempted on a file(database), which format is incompatible with the current version of db4o.

**InvalidIDException** - an exception to be thrown when an ID format supplied to #bind or #getById methods is incorrect.

**InvalidPasswordException** - this exception is thrown when the password provided to access an encrypted database is not correct.

**OldFormatException** - an exception to be thrown when an old file format was detected and the file could not be open.

**ReflectException** - an exception to be thrown when a class can not be stored or instantiated by current db4o reflector.

**ReplicationConflictException** - an exception to be thrown when a conflict occurs and no ReplicationEventListener is specified.

This revision (4) was last Modified 2007-05-11T09:20:14 by Tetyana.

# How To Work With Db4o Exceptions

Appropriate exception handling will help you to create easy to support systems, saving your time and efforts in the future. The following hints identify important places for exception handling.

1. Opening a database file can throw a DatabaseFileLockedException:

ExceptionExample.java: openDatabase

```

01  private static ObjectContainer openDatabase() {
02 |     ObjectContainer container = null;
03 |     try {
04 |         container = Db4o.openFile(DB4O_FILE_NAME);
05 |     } catch(DatabaseFileLockedException ex) {
06 |         // System.out.println(ex.getMessage());
07 |         // ask the user for a new filename, print
08 |         // or log the exception message
09 |         // and close the application,
10 |         // find and fix the reason
11 |         // and try again
12 |     }
13 |     return container;
14 |

```

2. Opening a client connection can throw IOException:

ExceptionExample.java: openClient

```

01  private static ObjectContainer openClient() {
02 |     ObjectContainer container = null;
03 |     try {
04 |         container = Db4o.openClient("host", 0xdb40, "user", "password");
05 |     } catch(Db4oIOException ex) {
06 |         //System.out.println(ex.getMessage());
07 |         // ask the user for new connection details, print
08 |         // or log the exception message
09 |         // and close the application,

```

```

10 |         // find and fix the reason
11 |         // and try again
12 |     } catch (OldFormatException ex) {
13 |         // see above
14 |     } catch (InvalidPasswordException ex) {
15 |         // see above
16 |     }
17 |     return container;
18 | }

```

3. Working with db4o and committing a transaction can throw various exceptions; the best practice is to surround your db4o interaction with try-catch block.

#### ExceptionExample.java: work

```

01 | private static void work() {
02 |     ObjectContainer container = openDatabase();
03 |     try {
04 |         // do some work with db4o
05 |         container.commit();
06 |     } catch (Db4oException ex) {
07 |         // handle exception ....
08 |     } catch (RuntimeException ex) {
09 |         // handle exception ....
10 |     } finally {
11 |         container.close();
12 |     }
13 | }

```

This revision (2) was last Modified 2007-04-23T08:14:24 by Tetyana.

# Platform Specific Issues

Db4o can be run in a variety of environments, which have Java virtual machine or .NET CLR. We use a common core code base, which allows automatic production of db4o builds for the following platforms:

- Java JDK 1.1
- Java JDK 1.2
- Java JDK 1.4
- Java JDK 5
- .NET 1.1
- .NET 1.1 CompactFramework
- .NET 2.0
- .NET 2.0 CompactFramework
- Mono

Db4o has a small database footprint and requires minimum processing resources thus being an excellent choice for embedded use in smartphones, photocopiers, car electronics, and packaged software (including real-time monitoring systems). It also shows good performance and reliability in web and desktop applications.

You can use db4o on desktop with:

- Windows (Java, .Net)
- Linux (Java, Mono).

On mobile and embedded devices with:

- Symbian (PersonalJava)
- Savaje(J2ME CDC)
- Zaurus(Personal Java or Java Personal Profile)
- Windows CE or Windows Mobile (.NET Compact Framework 1.1 and 2.0)

db4o provides the same API for all platforms, however each platform has its own features, which should be taken into consideration in software development process. These features will be discussed in the following chapters.

More Reading:

- [db4o on Java Platforms](#)
- [Security Requirements On Java Platform](#)
- [db4o on .NET Platforms](#)
- [Security Requirements On .NET Platform](#)

- [Deployment Instructions](#)
- [Db4o On Mono](#)
- [Cross-Platform Applications](#)
- [Symbian OS](#)
- [Servlets](#)
- [ASP.NET](#)
- [Xml Import-Export In .NET](#)
- [Xml Import-Export In Java](#)
- [Classloader issues](#)
- [Database For OSGi](#)
- [Isolated Storage](#)
- [Android](#)

This revision (6) was last Modified 2006-11-18T14:03:02 by Tetyana.

# db4o on Java Platforms

## Contents

- [All Java](#)
- [JDK1.1](#)
- [JDK1.2 - JDK 1.3](#)
- [JDK 1.4](#)
- [JDK 1.5](#)
- [Mobile Java editions](#)
- [Which db4o Java version to use?](#)

This topic applies to Java version only

## All Java

---

- root package is com.db4o

## JDK1.1

---

The major limitations of db4o for JDK1.1:

- no support for storing private fields (reflection in JDK 1.1 can only work on public fields)
- no support for JDK collections (java.util.list since JDK1.2)
- no support for weak references

db4o for Java 1.1 also goes without support for Native Query Optimization. NQ optimization uses bytecode optimizer library Bloat (<http://www.cs.purdue.edu/s3/projects/bloat/>), which is not JDK1.1 compatible.

## JDK1.2 - JDK 1.3

---

Java JDKs after version 1.1 are free of JDK1.1 limitations mentioned above. You can also use Native Query Optimization since JDK1.2.

The main limitation of JDK1.2 - 1.3 is the lack of file locking functionality. As it is necessary to lock the database file in use, db4o simulates locking files by using a timer thread that writes access time to the file. This can be quite slow.

## JDK 1.4

---

File locking functionality is available since JDK1.4, so that expensive database locking simulation is not necessary anymore and is advised to be switched off:

```
Db4o.configure().lockDatabaseFile(false)
```

Db4o can bypass the constructors declared for the class using platform-specific mechanisms. (For Java, this option is only available on JREs >= 1.4.) This mode allows reinstantiating objects even when their class doesn't provide a suitable constructor. For more information see [Constructors chapter](#)

If this option is available in the current runtime environment, it will be the default setting.

## JDK 1.5

---

- Generics support introduced in JDK1.5 makes db4o Native Query syntax much simpler:  

```
List <Pilot> pilots = db.query(new Predicate <Pilot> () {
    public boolean match(Pilot pilot) {
        return pilot.getPoints() == 100; }});
```
- following JDK5 annotations db4o introduces its own annotations.
- you can use built-in enums
- db4o for JDK5 also has replication support.

## Mobile Java editions

---

Currently db4o runs on J2ME dialects that support reflection, such as Personal Java, J2ME CDC and J2ME PersonalProfile. PersonalJava is closely equivalent to Java 1.1.8 regarding the libraries and features it contains (see JDK1.1 for the list of limitations). Use db4o-x.x-java1.1.jar with PersonalJava.

J2ME CDC and J2ME PersonalProfile are based on subsets of JDK1.3 or JDK1.4 depending on the version used.

J2ME CLDC and MIDP are not yet supported. Their support requires:

- replacing reflection (not available in CLDC) with a build-time preprocessor
- providing RMS RecordStore based I/O

## Which db4o Java version to use?

---

Db4o comes with several jars supporting different java versions. Together with the advanced features of higher Java versions db4o provides valuable improvements to its functionality (see the comparison above).

To get the best functionality you must use the highest db4o java version that your virtual machine supports:

Java versions	Recommended db4o jar	
JRE1.1	db4o-x.x-java1.1.jar	When you use JRE1.4 you should use db4o-x-x-java1.2.jar with database locking simulation disabled:  Db4o.configure().lockDatabaseFile(false)
JRE1.2, JRE1.3, JRE1.4	db4o-x.x-java1.2.jar	
JRE5, JRE6	db4o-x.x-java5.jar	

This will improve the performance dramatically, because JRE1.4 provides a built-in functionality to lock database files and the costly db4o database locking simulation is not needed.

This revision (21) was last Modified 2007-05-07T13:44:49 by Tetyana.

# Security Requirements On Java Platform

This topic applies to Java version only

Java Security Manager can be used to specify Java application security permissions. It is usually provided by web-browsers for applet execution, however any Java application can make use of a security manager. For example, to use the default security manager you will only need to pass `-Djava.security.manager` option to JVM command line. Custom security managers can be created and utilized as well (please refer to Java documentation for more information).

Unfortunately, current db4o implementation can't work under a default security manager. If this feature is important for you, please, vote and watch db4o task [COR-830](#).

This revision (1) was last Modified 2007-09-01T14:01:08 by Tetyana.

# db4o on .NET Platforms

## Contents

- [All .NET](#)
- [.NET 2.0](#)
- [.NET CF 2.0](#)

This topic applies to .NET version only

## All .NET

---

- .NET version of db4o uses Pascal case for method names
- Root namespace is Db4objects.Db4o (since version 6.0)
- All namespaces start with upper case letter (since version 6.0)
- Interface names have an I prefix (since version 6.0)
- .NET Reflection mechanism adds assembly name to class definition. If you use db4o database with 2 applications (client and server) you'll have to move all persistent class definitions into a shared .dll. Identical classes compiled into different executables/libraries will be treated as different.
- .NET attributes can be used for db4o configuration
- Enumerations are treated as integer types

Please, refer to [Security Requirements On .NET Platform](#) for further information.

## .NET 2.0

---

- You can enjoy simplified syntax for Native Queries with Generics support introduced in .NET2.0.

```
 IList <Pilot> pilots = db.Query <Pilot>(delegate(Pilot pilot) {  
    return pilot.Points == 100;  
});
```
- db4o for .NET2.0 supports replication.

## .NET CF 2.0

---

Due to some platform limitations CompactFramework 2.0 users, which use the more convenient delegate based Native Query syntax and want their queries to be optimized, are required to run the Db4oTool.exe command line utility on their assemblies prior to deploying them.

This revision (10) was last Modified 2007-11-27T18:15:44 by Tetyana.

# Security Requirements On .NET Platform

This topic applies to .NET version only.

db4o requires certain security permissions to be granted for successfull execution. It is important to know these permission requirements if the environment where db4o will be used is not fully trusted. .NET security model is out of scope of this article, to find out more about it use internet search on ".NET security permissions".

Security permissions of an assembly can be calculated with the help of PermCalc tool, which can be found in VS2005 installation:

[Visual Studio Home]\SDK\v2.0\Bin

The following command line will calculate the minimum security permissions for Db4objects.Db4o.dll and will save them in xml format Sandbox.Permcalc.xml document:

```
PermCalc.exe -sandbox Db4objects.Db4o.dll
```

The output should look like this:

```
Db4o.Xml

01 <?xml version="1.0"?>
02 <Sandbox>
03   <PermissionSet version="1" class="System.Security.PermissionSet">
04     <IPermission version="1" class="System.Security.Permissions.FileIOPermission,
mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
Read=*AllFiles* PathDiscovery=*AllFiles* />
05     <IPermission version="1" class="System.Security.Permissions.ReflectionPermission,
mscorlib, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089" Flags="MemberAccess" />
06     <IPermission version="1" class="System.Security.Permissions.SecurityPermission,
mscorlib, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089" Flags="UnmanagedCode,
Execution, ControlEvidence, SerializatonFormatter, ControlAppDomain" />
07     <IPermission Window="SafeSubWindows" Clipboard="OwnClipboard" version="1"
```

```

class="System.Security.Permissions.UIPermission, mscorelib, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089" />
08    <IPermission version="1" class="System.Security.Permissions.
KeyContainerPermission, mscorelib, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089" Unrestricted="true" />
09  </PermissionSet>
10 </Sandbox>

```

(UIPermission is not required).

The table below contains short explanation of each permission requirement. For the complete list of method calls requiring special security permissions, please run permcalc tool with -Stacks parameter. (More information on PermCalc can be found on [MSDN](#) site).

Permission name	Functionality
FileIOPermission	File read, write and create permissions are required for the corresponding operations on the database file. db4o does not restrict the location of a database file, therefore these permissions and browsing permission is required for all files in the system.
ReflectionPermission	db4o ability to create runtime objects from the database data is based on reflection. Reflection should be allowed.
SecurityPermission:	
<i>UnmanagedCode Execution</i>	Unmanaged code is used internally for file access and socket operations. Permission for the code to run. Without this permission, managed code will not be executed.
<i>ControlEvidence</i>	Is required internally to make use of some .NET functionality
<i>SerializationFormatter</i>	Used to utilize serialization services (formatters)
<i>ControlAppDomain</i>	Utilized with AppDomain functions.
KeyContainerPermission	Is used in .NET Socket operations.

When you deploy an assembly, you must take into consideration all the assemblies that can be referenced from the original assembly. db4o can be deployed with the following additional assemblies:

- Db4objects.Db4o.NativeQueries.dll : for runtime [NQ](#) optimization;
- Db4objects.Db4o.Instrumentation.dll : bytecode instrumentation library (required for [NQ](#) optimization).

The following table lists the permission requirements of these auxiliary assemblies:

Assembly	Permission requirements
----------	-------------------------

Db4objects.Db4o.NativeQueries

- No specific permissions

Db4objects.Db4o.Instrumentation

- SecurityPermission(UnmanagedCode)
- UIPermission

This revision (4) was last Modified 2007-11-27T19:09:49 by Tetyana.

# Deployment Instructions

## Contents

- Java deployment

This topic will help you to select correct db4o libraries for deployment with your db4o-based application.

## Java deployment

For the basic db4o functionality you only need to deploy db4o jar itself:

Application	Required libraries
Standalone or client/server db4o application for Java 1.1	db4o-x.x-java1.1.jar
Standalone or client/server db4o application for Java 1.2	db4o-x.x-java1.2.jar
Standalone or client/server db4o application for Java 5	db4o-x.x-java5.jar

The following table shows, which features can be added to the basic functionality and the libraries that should be deployed in addition to the basic library to support these features:

Application features	Required libraries
Native Queries optimization (for Java > 1.1).  Note, that the application can be <a href="#">NQ optimized in compile-time</a> , in which case NQ optimization libraries are not required to be deployed.	bloat-1.0.jar  db4o-x.x-nqopt.jar
Transparent Activation support	db4o-x.x-taj.jar  bloat.jar  db4o-x.x-instrumentation.jar

OSGI	db4o-x.x-osgi.jar
------	-------------------

This revision (3) was last Modified 2007-11-27T19:24:13 by Tetyana.

# Db4o On Mono

This topic applies to .NET version only

Steps to build Mono distribution.

As a native .NET database db4o can be used on [Mono](#), open source development platform based on the .NET framework. db4o Mono support is a community project, hosted at db4o Project Spaces.

db4o can be built from db4o sources which are available at [db4o svn](#).

In order to build db4o library you can use:

1. gnu make: Makefile is provided in db4o.net/Db4objects.Db4o folder
2. nant task: an example default.build (using csc) is provided in db4o.net folder.

Please, bear in mind that db4o for Mono is a community supported project and its correct functionality is not guaranteed. If you encounter compile errors during db4o build you can log the problem at [Mono bugzilla](#). You are also welcome to participate in db4o on Mono project development and improvement.

Once you've done the build you can test db4o functionality on Mono with a simple example:

MonoTest.cs

```
01  using System;
02  using System.IO;
03
04  using Db4obj ects.Db4o;
05  using Db4obj ects.Db4o.Query;
06
07
08  namespace Db4obj ects.Db4odoc.MonoTest
09  {
10  |
11  |     public class MonoTest
12  |     {
```

```
13 |         private const string Db4oFileName = "test.db4o";
14 |
15 |         public static void Main(string[] args)
16 |         {
17 |             File.Delete(Db4oFileName);
18 |             IObjectContainer db = Db4oFactory.OpenFile(Db4oFileName);
19 |             try
20 |             {
21 |                 Pilot pilot1 = new Pilot("Michael Schumacher", 100);
22 |                 db.Set(pilot1);
23 |                 Console.WriteLine("Stored {0}", pilot1);
24 |             }
25 |             finally
26 |             {
27 |                 db.Close();
28 |             }
29 |             db = Db4oFactory.OpenFile(Db4oFileName);
30 |             try
31 |             {
32 |                 IObjectSet result = db.Get(typeof(Pilot));
33 |                 ListResult(result);
34 |             }
35 |             finally
36 |             {
37 |                 db.Close();
38 |             }
39 |         }
40 |         // end Main
41 |
42 |         private static void ListResult(IObjectSet result)
43 |         {
44 |             System.Console.WriteLine(result.Count);
45 |             for (int i = 0; i < result.Count; i++)
```

```
46  {
47      System.Console.WriteLine(result[i]);
48  }
49  }
50 // end ListResult
51
52
53 class Pilot
54 {
55     private string _name;
56     private int _points;
57
58     public Pilot(string name, int points)
59     {
60         _name = name;
61         _points = points;
62     }
63
64     public string Name
65     {
66         get
67     {
68         return _name;
69     }
70     set
71     {
72         _name = value;
73     }
74 }
75
76     public int Points
77     {
78         get
```

```
79 |         {
80 |             return _points;
81 |         }
82 |     }
83 |
84 |     public override string ToString()
85 |     {
86 |         return string.Format("{0}/{1}", _name, _points);
87 |     }
88 | }
89 |
90 | }
91 | }
```

In order to compile and run the example you can use the following command:

```
mcs MonoTest.cs /r:Db4objects.Db4o.dll
```

```
mono MonoTest.exe
```

(It is assumed that Db4objects.Db4o.dll is in the same directory as the example file).

This revision (4) was last Modified 2007-09-15T17:21:37 by Tetyana.

# Cross-Platform Applications

db4o is a native java and .NET database. It reads Java and .NET objects and stores them in a platform independent format. In the runtime the database data is reconstructed into class objects using reflection. However, as the database storage is actually platform-independent, it does not matter if the data will be reconstructed into a java or .NET class object as soon as the class definition matches. Thus the same db4o database can be used both with Java and .NET application.

Further Reading:

- [Configuration](#)
- [Using Aliasing for Cross-Platform Development](#)
- [Limitations Of Db4o Cross-Platform Usage](#)

This revision (5) was last Modified 2007-09-02T13:42:56 by Tetyana.

# Configuration

In order to use the same database in Java and .NET application you will need to configure [Aliases](#). This is necessary due to the [difference in Java and .NET](#) class name format (it can be also helpful if you do not want to give the same names to the classes in different applications).

First of all you will need to alias the database itself:

Java:

```
Configuration configuration = Db4o.newConfiguration();

configuration.addAlias(new TypeAlias("Db4objects.Db4o.Ext.Db4oDatabase",
Db4objects.Db4o", "com.db4o.ext.Db4oDatabase"));
```

Then you will need to alias the persisted classes. If your class names match in Java and .NET, you can use a [WildcardAlias](#), which allows to alias all the classes in a package/namespace at once:

Java:

```
configuration.addAlias(new WildcardAlias("Db4objects.Db4odoc.Aliases.*",
Db4objects.Db4odoc", "com.db4odoc.aliases.*"));
```

If you want to alias only specific classes you can use [TypeAlias](#).

Java:

```
configuration.addAlias(new TypeAlias("Db4objects.Db4odoc.Aliases.Pilot",
Db4objects.Db4odoc", "com.db4odoc.aliases.Pilot"));
```

Remember that the configuration should be created and supplied to the object container on opening.  
This revision (3) was last Modified 2007-08-12T19:57:22 by Tetyana.

# Limitations Of Db4o Cross-Platform Usage

db4o cross-platform functionality is work in progress. Currently, it provides the basic features that enable you to use Java database on .NET and vice versa. However, it is recommended to familiarize yourself with the current limitations:

1. Some objects are treated differently in Java and .NET and could not be translated cleanly. This includes:

- Enumerations. In Java enumerations are similar to classes whereas in .NET enumeration is just a primitive type. For more information on how db4o treats both types see [Static Fields And Enums](#)
- Final Fields in Java behave differently in different platforms, which does not correspond to .NET readonly or const members. See [Final Fields](#)
- Collections. Please, keep an eye on [COR-766](#) Jira issue to see when collections cross-platform handling will be fixed.

2. Cross-platform client/server usage (Java server, .NET client or vice versa) is currently out of order ([COR-765](#)).

In general using db4o in cross-platform environment, you must try to keep your persistent class definitions simple and unambiguously interpreted on both platforms. Avoid constructs that exist only on Java or only on .NET platform.

Please, keep an eye on db4o [Jira](#) and [news](#) to stay informed about the latest progress on db4o cross-platform functionality.

This revision (1) was last Modified 2007-08-12T19:59:31 by Tetyana.

# Symbian OS

## Contents

- [Development Environment](#)
- [Programming specifics](#)

This topic applies to Java version only

Symbian OS is the global industry standard operating system for smartphones. You can find more information about it at <http://www.symbian.com/>.

UIQ (formerly known as User Interface Quartz) is a software platform based upon Symbian OS. UIQ-based devices support Java thus enabling you to use db4o.

## Development Environment

---

Db4o was tested for compatibility with UIQ 2.1 SDK. You can download the SDK from:

[http://developer.sonyericsson.com/site/global/docstools/symbian/p\\_symbian.jsp](http://developer.sonyericsson.com/site/global/docstools/symbian/p_symbian.jsp)

The UIQ Symbian SDK allows you to write applications for Symbian OS using your Windows PC and a suitable JDK ( JDK 1.1.8). The SDK comes with UIQ emulator, which can be run on a Windows-based computer allowing you to test and debug your application before deployment.

The SDK also installs some third party software, including JRE 1.3, which under some conditions can break Java installations already present ('java.dll not found' error message). In this case you can uninstalling the JRE that comes with Symbian to solve the problem.

The Emulator has its own file system ( you can get more information about how it is designed from the SDK documentation).

To be able to run Java applications and use db4o in the emulator, you will have to map \_epoc\_drive\_j to \epoc32\java. Consult your MS Windows documentation on how to set environment variables.

Environment variables can be set locally at the command prompt using the syntax

```
set _epoc_drive_j=\epoc32\java\
```

You can also launch your application on the Emulator from the Windows command prompt:

```
pjava -cd j:\demo DemoApp
```

You have to ensure that the correct version of the emulator VM executable (pjva.exe) is used - the correct path is /runtime/epoc32/release/wins/urel.

Command-line launch also allows you to pass arguments to a class's main(). Please, note that path names given to pjva are paths within the Emulator's drivespace only; they are not Windows paths.

Some platforms will require additional tuning to run the Emulator successfully. The following advices should help, if you are experiencing problems running the emulator:

- use a -cd (change directory) argument for pJava as well as a full -cp, both expressed in terms of the virtual file system.
- add the path to the JDK runtime classes (j:/lib/classes.zip, equivalent to /runtime/epoc32/java/lib/classes.zip) to the -cp argument.

The last thing you need to do is to copy the proper version of db4o jar (JDK1.1) to the emulator file system directory (/runtime/epoc32/java) and add its location to the classpath.

To make the startup process easier we recommend to create a batch file to run your application, which can look like this:

```
REM deploy all db4o files to C:\Symbian\UIQ_21\runtime\epoc32\java
```

```
SET SYMB_HOME=C:\Symbian\UIQ_21
```

```
SET SYMB_EPOC32=%SYMB_HOME%\runtime\epoc32
```

```
SET SYMB_BIN=%SYMB_EPOC32%\release\wins\urel
```

```
SET _epoc_drive_j=%SYMB_EPOC32%\java\
```

```
%SYMB_BIN%\pjva -cd J:\ -cp ..;J:\;J:\classes\;J:\db4o-5.0-java1.1.jar;J:\DemoApp.jar DemoClass
```

## Programming specifics

---

Tested version of Symbian JDK has problems with IO:

- seek() cannot move beyond the current file length;
- under certain (rare) conditions, calls to RandomAccessFile.length() seems to garble up the following reads.

To workaround these problems and make db4o file operations stable special SymbianIoAdapter is provided for Symbian OS:

```
Db4o.configure().io(new com.db4o.io.SymbianIoAdapter())
```

You can read more about using IOAdapters with db4o in [IOAdaper](#) chapter

The following example shows how SymbianIoAdapter can be used:

**SymbianTest.java**

```
01 /* Copyright (C) 2004 - 2006 db4objects Inc. http://www.db4o.com */  
02  
03 package com.db4odoc.fl.symbian;  
04  
05  
06 import java.io.File;  
07 import java.io.IOException;  
08  
09  
10 import com.db4o.Db4o;  
11 import com.db4o.ObjectContainer;  
12 import com.db4o.ObjectSet;  
13  
14  
15  
16 public class SymbianTest {  
17 |  
18 |     public static final String YAPFILENAME = "formula1.yap";  
19 |  
20 |     public static void main(String[] args) throws IOException {  
21 |         setObjects();  
22 |         setObjectsSymbian();  
23 |         getObjects();  
24 |         getObjectsSymbian();  
25 |     }  
26 |     // end main  
27 |  
28 |     public static void setObjects(){  
29 |         System.out.println("\nSetting objects using RandomAccessFileAdapter");  
30 |         new File(YAPFILENAME).delete();  
31 |         Db4o.configure().io(new com.db4o.io.RandomAccessFileAdapter());
```

```
32 |     try {
33 |         ObjectContainer db = Db4o.openFile(YAPFILENAME);
34 |         try {
35 |             db.set(new SymbianTest());
36 |         } finally {
37 |             db.close();
38 |         }
39 |     } catch (Exception ex) {
40 |         System.out.println("Exception accessing file: " + ex.getMessage());
41 |     }
42 | }
43 | // end setObjects
44 |
45 | public static void setObjectsSymbian() {
46 |     System.out.println("\nSetting objects using SymbianIoAdapter");
47 |     new File(YAPFILENAME).delete();
48 |     Db4o.configure().io(new com.db4o.io.SymbianIoAdapter());
49 |     try {
50 |         ObjectContainer db = Db4o.openFile(YAPFILENAME);
51 |         try {
52 |             db.set(new SymbianTest());
53 |         } finally {
54 |             db.close();
55 |         }
56 |     } catch (Exception ex) {
57 |         System.out.println("Exception accessing file: " + ex.getMessage());
58 |     }
59 | }
60 | // end setObjectsSymbian
61 |
62 | public static void getObjects() {
63 |     System.out.println("\nRetrieving objects using RandomAccessFileAdapter");
64 |     Db4o.configure().io(new com.db4o.io.RandomAccessFileAdapter());
65 |     try {
66 |         ObjectContainer db = Db4o.openFile(YAPFILENAME);
```

```
67 |     try {
68 |         ObjectSet result=db.get(new Object());
69 |         System.out.println("Objects in the database: " + result.size());
70 |     } finally {
71 |         db.close();
72 |     }
73 | } catch (Exception ex) {
74 |     System.out.println("Exception accessing file: " + ex.getMessage());
75 | }
76 | }
77 | // end getObjects
78 |
79 | public static void getObjectsSymbian() {
80 |     System.out.println("\nRetrieving objects using SymbianIOAdapter");
81 |     Db4o.configure().io(new com.db4o.io.SymbianIOAdapter());
82 |     try {
83 |         ObjectContainer db = Db4o.openFile(YAPFILENAME);
84 |         try {
85 |             ObjectSet result=db.get(new Object());
86 |             System.out.println("Objects in the database: " + result.size());
87 |         } finally {
88 |             db.close();
89 |         }
90 |     } catch (Exception ex) {
91 |         System.out.println("Exception accessing file: " + ex.getMessage());
92 |     }
93 | }
94 | // end getObjectsSymbian
95 | }
```

This revision (16) was last Modified 2007-05-07T13:50:01 by Tetyana.

# Servlets

This topic applies to Java version only

Running db4o as the persistence layer of a Java web application is easy. There is no installation procedure - db4o is just another library in your application. There are only two issues that make web applications distinct from standalone programs from a db4o point of view. One is the more complex classloader environment - db4o needs to know itself (of course) and the classes to be persisted. Please refer to the [classloader](#) chapter for more information.

The other issue is configuring, starting and shutting down the db4o server correctly. This can be done at the servlet API layer or within the web application framework you are using.

On the servlet API layer, you could bind db4o server handling to the servlet context via an appropriate listener. A very basic sketch might look like this:

Db4oServletContextListener.java

```
01 /* Copyright (C) 2004 - 2006 db4objects Inc. http://www.db4o.com */  
02  
03 package com.db4odoc.servlets;  
04  
05 import com.db4o.Db4o;  
06 import com.db4o.ObjectServer;  
07 import javax.servlet.*;  
08  
09  public class Db4oServletContextListener implements ServletContextListener {  
10 |   public static final String KEY_DB4O_FILE_NAME = "db4oFileName";  
11 |  
12 |   public static final String KEY_DB4O_SERVER = "db4oServer";  
13 |  
14 |   private ObjectServer server = null;  
15 |  
16  public void contextInitialized(ServletContextEvent event) {  
17 |   close();  
18 |   ServletContext context = event.getServletContext();  
19 |   String filePath = context.getRealPath("WEB-INF/db/"  
20 |     + context.getInitParameter(KEY_DB4O_FILE_NAME));
```

```

21     server = Db4o.openServer(filePath, 0);
22     context.setAttribute(KEY_DB40_SERVER, server);
23     context.log("db4o startup on " + filePath);
24 }
25
26 public void contextDestroyed(ServletContextEvent event) {
27     ServletContext context = event.getServletContext();
28     context.removeAttribute(KEY_DB40_SERVER);
29     close();
30     context.log("db4o shutdown");
31 }
32
33 private void close() {
34     if (server != null) {
35         server.close();
36     }
37     server = null;
38 }
39 }
```

This listener just has to be registered in the web.xml.

## web.xml

```

1 <context-param>
2   <param-name>db4oFileName</param-name>
3   <param-value>db4oweb.yap</param-value>
4 </context-param>
5 <listener>
6   <listener-class>
7     com.db4odoc.servlets.Db4oServletContextListener
8   </listener-class>
9 </listener>
```

Now db4o should be available to your application classes.

```
ObjectServer server=(ObjectServer)context.getAttribute("db4oServer")
```

A more complex and 'old school' example without using context listeners comes with the samples section of the db4o3 distribution that's still available from our web site.

However, We strongly suggest that you use the features provided by your framework and that you consider not exposing db4o directly to your application logic. (There is nothing db4o-specific about these recommendations, we would vote for this in the presence of any persistence layer.)

This revision (13) was last Modified 2007-05-07T13:53:11 by Tetyana.

# ASP.NET

## Contents

- Security Requirements
- Sample Application
- Persisting Objects in ASP.NET2
- More Information

This topic applies to .NET version only

You can use db4o as a persistent layer of your ASP.NET application. The actual integration of db4o into your web application is quite simple: you just need to add reference to Db4objects.Db4o.dll and use it in a client/server mode.

## Security Requirements

However, as it is characteristic to web-applications in general, there are some security permissions involved, which can in fact forbid db4o functionality in your ASP.NET application. So, before developing/deploying you should make sure that the required permissions would be granted to your application at the hosting server:

1. ASPNET user should have read/write permissions to the directory containing database file. Obviously this is necessary to work with the database.
2. Trust Level of your site should be set to "Full".
3. All the necessary permissions should be granted to db4o assembly. The easiest way to ensure this is to add full trust to db4o:
  - For .NET 1.1 you can set this using caspol utility:  
caspol -af Db4objects.Db4o.dll
  - For .NET versions >= 2.0 Db4objects.Db4o.dll should be installed in GAC

If full trust is not a suitable solution for, you can check the minimum security permissions that db4o dll needs to operate using permcalc.exe tool from your Visual Studio installation.

```
PermCalc.exe -Sandbox Db4objects.Db4o.dll
```

sandbox.PermCalc.xml

```

01  <?xml version="1.0"?>
02  <Sandbox>
03    <PermissionSet version="1" class="System.Security.PermissionSet">
04      <IPermission version="1" class="System.Security.Permissions.FileIOPermission, mscorel, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" Read="*AllFiles*"
05        PathDiscovery="*AllFiles*" />
06      <IPermission version="1" class="System.Security.Permissions.ReflectionPermission, mscorel, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
07        Flags="MemberAccess" />
08      <IPermission version="1" class="System.Security.Permissions.SecurityPermission, mscorel, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
09        Flags="UnmanagedCode, Execution, ControlEvidence" />
10     <IPermission Window="SafeSubWindows" Clipboard="OwnClipboard" version="1" class="System.Security.Permissions.UIPermission, mscorel, Version=2.0.0.0, Culture=neutral,
11   
```

```

    Publ i cKeyToken=b77a5c561934e089" />
08     <I Permi ssi on versi on="1" class="System.Security.Permissions.KeyContai nerPermi ssi on,
mscorlib, Version=2.0.0.0, Culture=neutral, Publ i cKeyToken=b77a5c561934e089"
Unrestricted="true" />
09   </Permi ssi onSet>
10 </Sandbox>

```

Consult with your web-server administrator to grant these permissions.

## Sample Application

---

Let's look at an example implementation: we will create a simple ASP.NET application, which will use db4o to store, retrieve and delete objects

The basic requirements for seamless db4o integration are:

- db4o server should open database file at application start and close it when application is terminated
- clients, accessing application through web-browser, should get their db4o-client connection on request.

In order to keep page-specific code clean we will implement db4o functionality in a separate Db4oHttpModule implementing IHttpModule interface.

Database file path can be saved in [Web.Config](#):

```
<appSettings>
  <add key="db4oFileName" value="~/Data/Test.yap" />
</appSettings>
```

Make sure that you have Data folder inside you web-site directory. ASP.NET user should have enough rights to create and modify files inside this folder. You should also make sure that Data folder is not accessible to anonymous user; otherwise web-server visitors will be able to download your database.

We will open db4o connection only with the first client request:

The following code will ensure that the database is closed upon termination:

This is basically all the functionality that is required from db4o module. In order to make use of it we need to register it in [Web.Config](#):

```
<httpModules>
  <add type="Db4objects.Db4odoc.Web.Data.Db4oHttpModule" name="Db4oHttpModule" />
</httpModules>
```

With the help of the created module we can access db4o database fairly easy:

To test the whole application you can use the following simple form:

## Persisting Objects in ASP.NET2

---

If you are creating your application in ASP.NET2 you should take into consideration the fact that the assembly names are generated automatically on each build by default. Db4o distinguish persisted classes by name, namespace and assembly, so after the assembly name change, you won't be able to access classes saved with the previous version of the assembly.

There are several workarounds:

- You can create a separate class library keeping db4o logic and persistent classes. This can also help if you need to access fully trusted db4o library from partially trusted ASP application
- You can build your ASP.NET2 application manually using aspnet\_compiler utility from .NET2 SDK.

```
aspnet_compiler.exe -v /WebSite -f -fixednames c:\WebSite -c -errorstack
```

- You can use db4o aliasing API to redirect saved classes to the new assembly name. The following method should be called before opening database file to make all the classes within the namespace available after assembly name change:

You can use TypeAlias for aliasing only specific class.

## More Information

---

Additional information about using db4o in ASP.NET environment can be found in db4o [Project Spaces](#). The following community projects featuring db4o ASP.NET membership providers exist:

- [db4o ASP.NET Providers](#)
- [db4oMembershipProvider for ASP.NET2](#)

This revision (22) was last Modified 2008-01-01T10:54:11 by Tetyana.

# Xml Import-Export In .NET

This topic applies to .NET version only

One of the most widely used platform independent formats of data exchange today is xml.

Db4o does not provide any specific API to be used for XML import/export, but with the variety of XML serialization tools available for Java and .NET (freeware and licensed) this is not really necessary.

All that you need to export your database/query results is:

1. Retrieve objects from the database.
2. Serialize them in XML format (using language, or external tools, or your own serializing software).
3. Save XML stream (to a disc location, into memory, into another database).

Import process is just the reverse:

1. Read XML stream
2. Create an objects from XML
3. Save objects to db4o

Let's go through a simple example. We will use .NET XmlSerializer. (You can use any other XML serialization tool, which is able to serialize/deserialize classes).

First, let's prepare a database:

We will save the database to XML file "formula1.xml":

After the method executes all car objects from the database will be stored in the export file as an array. Note that child objects (Pilot) are stored as well without any additional settings. You can check the created XML file to see how it looks like.

Now we can clean the database and try to recreate it from the XML file:

Easy, isn't it? Obviously there is much more about XML serialization: renaming fields, storing collections, selective persistence etc. You should be able to find detailed description together with the serialization library, which you will use.

This revision (5) was last Modified 2007-12-26T14:59:36 by Tetyana.

# Xml Import-Export In Java

This topic applies to Java version only

One of the most widely used platform independent formats of data exchange today is xml. Db4o does not provide any specific API to be used for XML import/export, but with the variety of XML serialization tools available for Java and .NET (freeware and licensed) this is not really necessary. All that you need to export your database/query results is:

1. Retrieve objects from the database.
2. Serialize them in XML format (using language, or external tools, or your own serializing software).
3. Save XML stream (to a disc location, into memory, into another database).

Import process is just the reverse:

1. Read XML stream
2. Create an objects from XML
3. Save objects to db4o

Let's go through a simple example. We will use xstream library (<http://xstream.codehaus.org/>) for object serialization, but any other tool capable of serializing objects into XML will do as well.

First, let's prepare a database:

Code attachment not found: /Resources/Reference/Platform\_Specific\_Issues/SerializeJava.Zip

We will save the database to XML file "formula1.xml":

Code attachment not found: /Resources/Reference/Platform\_Specific\_Issues/SerializeJava.Zip

After the method executes all car objects from the database will be stored in the export file as an array. Note that child objects (Pilot) are stored as well without any additional settings. You can check the created XML file to see how it looks like.

Now we can clean the database and try to recreate it from the XML file:

Code attachment not found: /Resources/Reference/Platform\_Specific\_Issues/SerializeJava.Zip

Easy, isn't it? Obviously there is much more about XML serialization: renaming fields, storing collections,

selective persistence etc. You should be able to find detailed description together with the serialization library, which you will use.

This revision (6) was last Modified 2007-02-01T21:20:44 by treeder.

# Classloader issues

This topic applies to Java version only

Db4o needs to know its own classes, of course, and it needs to know the class definitions of the objects it stores. (In Client/Server mode, both the server and the clients need access to the class definitions.) While this usually is a non-issue with self-contained standalone applications, it can become tricky to ensure this condition when working with plugin frameworks, where one might want to deploy db4o as a shared library for multiple plugins, for example.

More Reading:

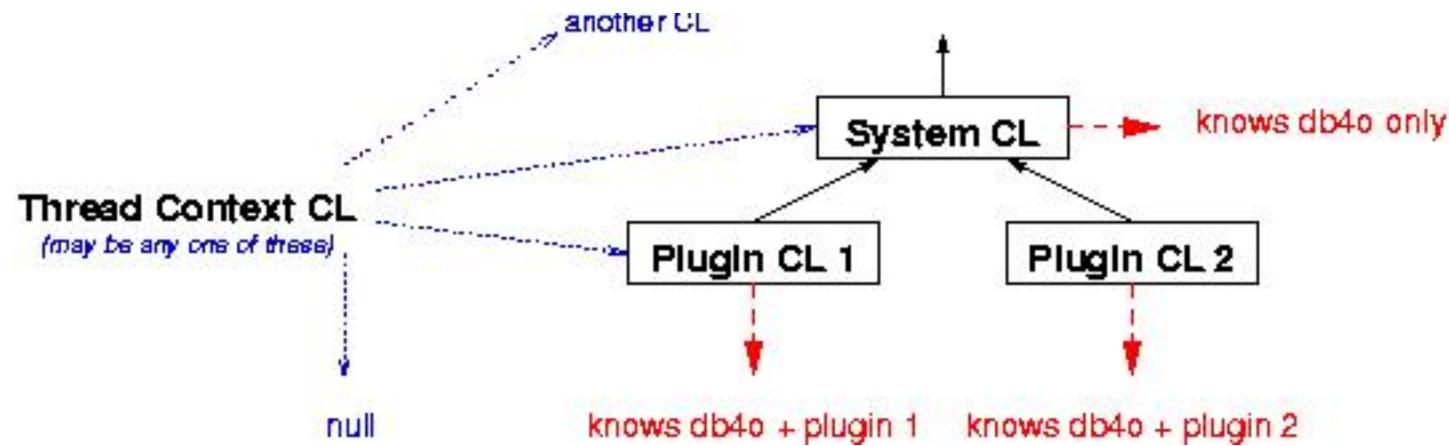
- [Classloader basics](#)
- [Configuration](#)
- [Special Cases](#)

This revision (8) was last Modified 2006-12-04T07:42:32 by Tetyana.

# Classloader basics

This topic applies to Java version only

Classloaders are organized in a tree structure, where classloaders deeper down the tree (usually) delegate requests to their parent classloaders and thereby 'share' their parent's knowledge.



An in-depth explanation of the classloaders functionality is beyond the scope of this documentation. Starting points might be found here:

<http://www.javaworld.com/javaworld/javaqa/2003-06/01-qa-0606-load.html>

<http://java.sun.com/developer/technicalArticles/Networking/classloaders/>

<http://java.sun.com/products/jndi/tutorial/beyond/misc/classloader.html>

This revision (8) was last Modified 2006-12-04T07:38:42 by Tetyana.

# Configuration

This topic applies to Java version only

Db4o can be configured to use a user-defined classloader.

```
Java: Db4o.configure().reflectWith(new JdkReflector(classloader))
```

This line will configure db4o to use the provided classloader. Note that, as with most db4o configuration options, this configuration will have to occur before the respective database has been opened.

The usual ways of getting a classloader reference are:

- Using the classloader the class containing the currently executed code was loaded from. (*this.getClass().getClassLoader()*)
- Using the classloader db4o was loaded from. (*Db4o.class.getClassLoader()*)
- Using the classloader your domain classes were loaded from. (*SomeDomainClass.class.getClassLoader()*)
- Using the context classloader that may have been arbitrarily set by the execution environment. (*Thread.currentThread().getContextClassLoader()*).

To choose the right classloader to use, you have to be aware of the classloader hierarchy of your specific execution environment. As a rule of thumb, one should configure db4o to use a classloader as deep/specialized in the tree as possible. In the above example this would be the classloader of the plugin db4o is supposed to work with.

This revision (7) was last Modified 2006-12-04T07:40:16 by Tetyana.

# Special Cases

## Contents

- [Servlet container](#)
- [Eclipse](#)
- [Running Without Classes](#)

This topic applies to Java version only

In your average standalone program you'll probably never have to face these problems, but there are standard framework classics that'll force you to think about these issues.

## Servlet container

---

In a typical servlet container, there will be one or more classloader responsible for internal container classes and shared libraries, and one dedicated classloader per deployed web application. If you deploy db4o within your web application, there should be no problem at all. When used as a shared library db4o has to be configured to use the dedicated web application classloader. This can be done by assigning the classloader of a class that's present in the web application only, or by using the context classloader, since all servlet container implementations we are aware of will set it accordingly.

You will find more detailed information on classloader handling in Tomcat, the reference servlet container implementation, here:

<http://jakarta.apache.org/tomcat/tomcat-4.1-doc/class-loader-howto.html>

## Eclipse

---

Eclipse uses the system classloader to retrieve its core classes. There is one dedicated classloader per plugin, and the classloader delegation tree will resemble the plugin dependency tree. The context classloader will usually be the system classloader that knows nothing about db4o and your business classes. So the best candidate is the classloader for one of your domain classes within the plugin.

## Running Without Classes

---

db4o can also cope with missing class definitions. This is a by-product of the work on our [object manager](#) application. Another implementation is a [server without persistent classes deployed](#).

This revision (10) was last Modified 2007-09-15T14:37:10 by Tetyana.



# Database For OSGi

This topic applies to Java version only

db4o osgi project (since db4o-6.3) provides a service, which allows to use db4o in OSGI environment. Its usage is the usage of an OSGI service, which is well documented in the Internet. Short and essential guide to OSGI service usage can be found [here](#), but you are surely free to use any suitable for you technique to access db4o osgi service.

The main purpose of db4o osgi service is to configure an OSGi bundle aware reflector for the database instance, so that classes that are owned by the client bundle are accessible to the db4o engine. To emulate this behavior when using db4o directly through the exported packages of the db4o osgi plugin, db4o can be configured like this:

```
Configuration config = Db4o.newConfiguration();

config.reflectWith(new JdkReflector(SomeData.class.getClassLoader()));

// ...

ObjectContainer database = Db4o.openFile(config,fileName);
```

Access through the service is recommended over the direct usage, though, as the service may implement further OSGI specific features in the future.

db4o osgi.jar can be found in /lib folder of the Java distribution with detailed API documentation in /doc/osgi folder.

If you are comfortable with OSGI and only need a short introduction to db4o osgi service you can read [Db4o-Osgi Usage](#).

For a more detailed explanation and example of db4o-service usage in an Eclipse plug-in see [Eclipse Plug-In With Db4o Service](#). This example will be also helpful for those who are new to OSGI.

This revision (5) was last Modified 2007-07-29T14:25:04 by Tetyana.

# Db4o-Osgi Usage

This topic applies to Java version only.

db4o-osgi service can be accessed like any other OSGI service:

```
ServiceReference serviceRef = _context.getServiceReference(Db4oService.class.  
getName( ));
```

```
Db4oService db4oService = (Db4oService)_context.getService(serviceRef);
```

db4o-osgi uses Bundle-ActivationPolicy:lazy header to define the lazy bundle loading policy (only utilized in some environments, like Eclipse).

Db4oService instance can be used as Db4o class in usual environment:

```
Configuration config = db4oService.newConfiguration();
```

```
ObjectContainer db = db4oService.openFile(config, filename);
```

Also available are methods for opening db4o server and client. For more information see the API documentation.

Once the service instance is obtained, you can continue to work with db4o API as usual.

This revision (3) was last Modified 2007-08-05T17:01:04 by Tetyana.

# Eclipse Plug-In With Db4o Service

This topic applies to Java version only.

The following example was created to show a practical usage of db4o osgi service. Though targeting a wide auditory, it can be especially helpful to people new to OSGI and plug-in development.

In this example we will create a simple Eclipse UI plug-in, which will store notes between Eclipse sessions using db4o as storage.

To be able to follow the explanation you will need:

- JDK 1.5
- db4o-6.3 for java ([download](#))
- eclipse IDE v.3.3. ([download](#)). Please, note that db4o-osgi uses Bundle-ActivationPolicy:lazy header to define the lazy bundle loading policy; this setting is only available since Eclipse v 3.3.

More Reading:

- [Creating A Plugin](#)
- [Code Overview](#)
- [Connecting To Db4o](#)
- [Testing MemoPlugin](#)

This revision (2) was last Modified 2007-08-05T17:02:30 by Tetyana.

# Creating A Plugin

This topic applies to Java version only.

First of all you will need to install db4o osgi into the Eclipse environment. You can do that by copying db4o osgi.jar into ECLIPSE\_HOME\plugins folder. If you do not want to do that, you can open Eclipse and create a usual java project from the db4o osgi sources.

Now you are ready to create a new plug-in project.

- Open Eclipse workspace if not yet opened.
- Select File/New from the menu and select "Plug-in Project" as the project type.
- Select MemoPlugin as the project name, leave the default values for the other settings and press "Next"
- Leave all the default values and press "Next"
- In the "Templates" screen select "Hello, World" template. This template creates a menu in the Eclipse environment, which we will use for our example. Click "Finish"

You might be asked to switch to "Plug-in development" perspective, which you can surely do.

You should see a MemoPlugin window opened in the environment. This window represents important plug-in properties and it can be opened by double-clicking plugin.xml file in Package Explorer.

You can use the tab-scroll at the bottom to navigate to different pages. Please, open the "Overview" page of the plugin.xml window and review the information presented there. Note, that this page can be used to start testing and debugging (see Testing paragraph).

Our plug-in will depend on db4o osgi bundle; therefore we must define this dependency somewhere. Select "Dependencies" hyperlink in the "Plug-in Content" paragraph. (You can gain the same effect by selecting "Dependencies" tab page.) In the "Required Plug-ins" list click "Add" and select "db4o-osgi".

Please, note that you should not specify Java Build path as in a normal Java project, otherwise the environment will find duplicates in your project dependencies.

This revision (1) was last Modified 2007-07-29T14:30:21 by Tetyana.

# Code Overview

This topic applies to Java version only.

Now the plug-in environment is configured and we can look at the code itself.

At this point in time the project contains the following classes:

- memoplugin.Activator
- memoplugin.actions.SampleAction

Activator class is called to start and stop the plug-in. It is responsible for managing its lifecycle. We will use it to initialize and clean up db4o resources.

SampleAction is a class that performs the action specified in the action set in the manifest file. It can be used to specify the behavior on the action. We will use it to call a custom dialog for memo viewing and editing.

From the said above we can see that we will need 2 more classes:

- Db4oProvider: will be used to keep db4o connection, provide it to the users on request, and close it on dispose.
- DataDialog: will provide a simple UI for viewing and editing the data, it will use Db4oProvider to access and store the data.

These 2 classes are very basic and are not specific to OSGI environment. Please, review their code below:

**Db4oProvider.java**

```

01 package memoplugin;
02
03 import com.db4o.ObjectContainer;
04 import com.db4o.osgi.Db4oService;
05 /**
06 * This class is used to store db4o osgi service instance and
07 * provide db4o services on request.
08 */
09 public class Db4oProvider {
10
11     private static ObjectContainer _db;
12     private static String FILENAME = "sample.db4o";
13
14     public static void Initialize(Db4oService db4oService) {
15         _db = db4oService.openFile(FILENAME);
16     }
17
18     public static ObjectContainer database() {
19         return _db;

```

```

20 |    }
21 |
22 public static void UnInitialize() {
23 |    _db. close();
24 |}
25 |
26 }

```

## DataDialog.java

```

001 package memoplugin.ui;
002
003 import java.util.ArrayList;
004
005 import org.eclipse.jface.dialogs.Dialog;
006 import org.eclipse.jface.dialogs.IDialogConstants;
007 import org.eclipse.jface.dialogs.MessageDialog;
008 import org.eclipse.swt.SWT;
009 import org.eclipse.swt.layout.GridData;
010 import org.eclipse.swt.widgets.Button;
011 import org.eclipse.swt.widgets.Composite;
012 import org.eclipse.swt.widgets.Control;
013 import org.eclipse.swt.widgets.Label;
014 import org.eclipse.swt.widgets.List;
015 import org.eclipse.swt.widgets.Shell;
016 import org.eclipse.swt.widgets.Text;
017
018 import memoplugin.Db4oProvider;
019
020 import com.db4o.ObjectSet;
021
022 public class DataDialog extends Dialog {
023 |    private static int ID_ADD = 100;
024 |    private static int ID_DELETE = 101;
025 |    private Shell _shell;
026     /**
027 |        * The title of the dialog.
028 |        */
029 |    private String title;

```

```
030 |
031 |     /**
032 |      * The message to display, or <code>null</code> if none.
033 |      */
034 |     private String message;
035 |
036 |     /**
037 |      * The input value; the empty string by default.
038 |      */
039 |     private String value = ""; //NON-NLS-1$
040 |
041 |
042 |     /**
043 |      * Add button widget.
044 |      */
045 |     private Button addButton;
046 |
047 |     /**
048 |      * Delete button widget.
049 |      */
050 |     private Button deleteButton;
051 |
052 |     /**
053 |      * Input text widget.
054 |      */
055 |     private Text text;
056 |
057 |     /**
058 |      * List widget.
059 |      */
060 |     private List list;
061 |
062 |
063 |     public DataDialog(Shell parentShell, String dialogTitle,
064 |                       String dialogMessage, String initialValue) {
065 |         super(parentShell);
066 |         this.title = dialogTitle;
067 |         message = dialogMessage;
068 |         if (initialValue == null) {
069 |             value = ""; //NON-NLS-1$
070 |         } else {
```

```
071 |         value = initialValue;
072 |     }
073 | }
074 |
075 |     /*
076 |     * (non-Javadoc)
077 |     *
078 |     * @see org.eclipse.jface.window.Window#configureShell(org.eclipse.swt.widgets.Shell)
079 | */
080 |     protected void configureShell(Shell shell) {
081 |         super.configureShell(shell);
082 |         _shell = shell;
083 |         if (title != null) {
084 |             shell.setText(title);
085 |         }
086 |     }
087 |
088 |     /*
089 |     * Clears the database before adding new data
090 |     */
091 |     private void clearDb() {
092 |         ObjectSet result = Db4oProvider.database().get(ArrayList.class);
093 |         while (result.hasNext()) {
094 |             Db4oProvider.database().delete(result.next());
095 |         }
096 |     }
097 |
098 |     /*
099 |     * (non-Javadoc)
100 |     * Makes sure that all the data is saved to the
101 |     * database before closing the dialog
102 |     */
103 |     protected void handleShellCloseEvent() {
104 |         clearDb();
105 |         ArrayList data = new ArrayList();
106 |         for (int i=0; i < list.getItemCount(); i++) {
107 |             data.add(list.getItem(i));
108 |         }
109 |         Db4oProvider.database().set(data);
110 |         Db4oProvider.database().commit();
111 |         Db4oProvider.database().ext().purge(ArrayList.class);
```

```

112 |         super. handleShellCloseEvent();
113 |
114 |
115 |     /*
116 |      * Button events handler
117 |     */
118 |     protected void buttonPressed(int buttonId) {
119 |         if (buttonId == ID_ADD) {
120 |             value = text.getText();
121 |             list.add(value);
122 |         } else if (buttonId == ID_DELETE) {
123 |             int selectedId = list.getSelectionIndex();
124 |             if (selectedId == -1) {
125 |                 new MessageDialog(_shell, "Error",
126 |                                 null, "No item selected", MessageDialog.ERROR,
127 |                                 new String[] {"Ok"}, 0).open();
128 |             } else {
129 |                 list.remove(selectedId);
130 |             }
131 |             value = null;
132 |         } else {
133 |             super.buttonPressed(buttonId);
134 |         }
135 |     }
136 |
137 |     /*
138 |      * (non-Javadoc)
139 |      *
140 |      * @see org.eclipse.jface.dialogs.Dialog#createButtonsForButtonBar(org.eclipse.swt.widgets.Composite)
141 |     */
142 |     protected void createButtonsForButtonBar(Composite parent) {
143 |         // create Add and Delete buttons by default
144 |         addButton = createButton(parent, ID_ADD,
145 |                                 "Add", true);
146 |         createButton(parent, ID_DELETE,
147 |                                 "Delete", false);
148 |         // do this here because setting the text will set enablement on the ok
149 |         // button
150 |         text.setFocus();
151 |         if (value != null) {

```

```

152 |         text.setText(value);
153 |         text.selectAll();
154 |     }
155 |
156 |
157 |     /*
158 |      * (non-Javadoc) Creates the visual dialog representation
159 |      */
160 |     protected Control createDialogArea(Composite parent) {
161 |         // create composite
162 |         Composite composite = (Composite) super.createDialogArea(parent);
163 |         // create message
164 |         if (message != null) {
165 |             Label label = new Label(composite, SWT.WRAP);
166 |             label.setText(message);
167 |             GridData gridData = new GridData(GridData.GRAB_HORIZONTAL
168 |                     | GridData.GRAB_VERTICAL | GridData.HORIZONTAL_ALIGN_FILL
169 |                     | GridData.VERTICAL_ALIGN_CENTER);
170 |             gridData.widthHint = convertHorizontalDLUsToPixels(IDialogConstants.
MINIMUM_MESSAGE_AREA_WIDTH);
171 |             label.setLayoutData(gridData);
172 |             label.setFont(parent.getFont());
173 |         }
174 |         text = new Text(composite, SWT.SINGLE | SWT.BORDER);
175 |         text.setLayoutData(new GridData(GridData.GRAB_HORIZONTAL
176 |                     | GridData.HORIZONTAL_ALIGN_FILL));
177 |
178 |         list = new List(composite, SWT.SINGLE | SWT.H_SCROLL | SWT.V_SCROLL);
179 |         GridData gridData = new GridData(SWT.FILL, SWT.FILL, true, true);
180 |         gridData.heightHint = 50;
181 |         list.setLayoutData(gridData);
182 |         ObjectSet result = Db4oProvider.database().query(ArrayList.class);
183 |         if (result.size() != 0){
184 |             ArrayList data = (ArrayList)result.next();
185 |             String[] items = new String[data.size()];
186 |             for (int i=0; i < data.size(); i++){
187 |                 items[i] = (String)data.get(i);
188 |             }
189 |             list.setItems(items);
190 |         }
191 |

```

```

192 |
193 |         applyDialogFont(composite);
194 |         return composite;
195 |     }
196 |
197 |
198 |     /**
199 |      * Returns the string typed into this input dialog.
200 |      *
201 |      * @return the input string
202 |     */
203 |     public String getValue() {
204 |         return value;
205 |     }
206 |
207 }

```

In order to call the above-mentioned DataDialog we will need to modify the generated run method in SampleAction class:

#### SampleAction.java: run

```

01 /**
02 |     * The action has been activated. The argument of the
03 |     * method represents the 'real' action sitting
04 |     * in the workbench UI.
05 |     * @see IWorkbenchWindowActionDelegate#run
06 |
07 |     public void run(IAction action) {
08 |         /*
09 |             * Call DataDialog to view and edit memo notes
10 |         */
11 |         DataDialog d = new DataDialog(window.getShell(), "db4o-osgi", "Enter an item to add to
the list:", null);
12 |         d.open();
13 }

```

This revision (3) was last Modified 2007-07-29T14:36:31 by Tetyana.

# Connecting To Db4o

This topic applies to Java version only.

The only thing left - is a connection to the db4o osgi plug-in. It can be established upon the plug-in start and terminated upon the plug-in stop:

Activator.java: start

```

01 ■■■/*
02 |   * (non-Javadoc)
03 |   * @see org.eclipse.ui.plugin.AbstractUIPlugin#start(org.osgi.framework.BundleContext)
04 |   * Obtains a db4o osgi service reference and registers it with Db4oProvider
05 | */
06 ■■■ public void start(BundleContext context) throws Exception {
07 |     super.start(context);
08 |     ServiceReference serviceRef = context.getServiceReference(Db4oService.class.getName());
09 |     Db4oService db4oService = (Db4oService) context.getService(serviceRef);
10 |     Db4oProvider.Initialize(db4oService);
11 }

```

Activator.java: stop

```

01 ■■■/*
02 |   * (non-Javadoc)
03 |   * @see org.eclipse.ui.plugin.AbstractUIPlugin#stop(org.osgi.framework.BundleContext)
04 |   * Unregisters the db4o osgi service from Db4oProvider
05 | */
06 ■■■ public void stop(BundleContext context) throws Exception {
07 |     Db4oProvider.UnInitialize();
08 |     plugin = null;
09 |     super.stop(context);
10 }

```

This revision (1) was last Modified 2007-07-29T14:38:30 by Tetyana.

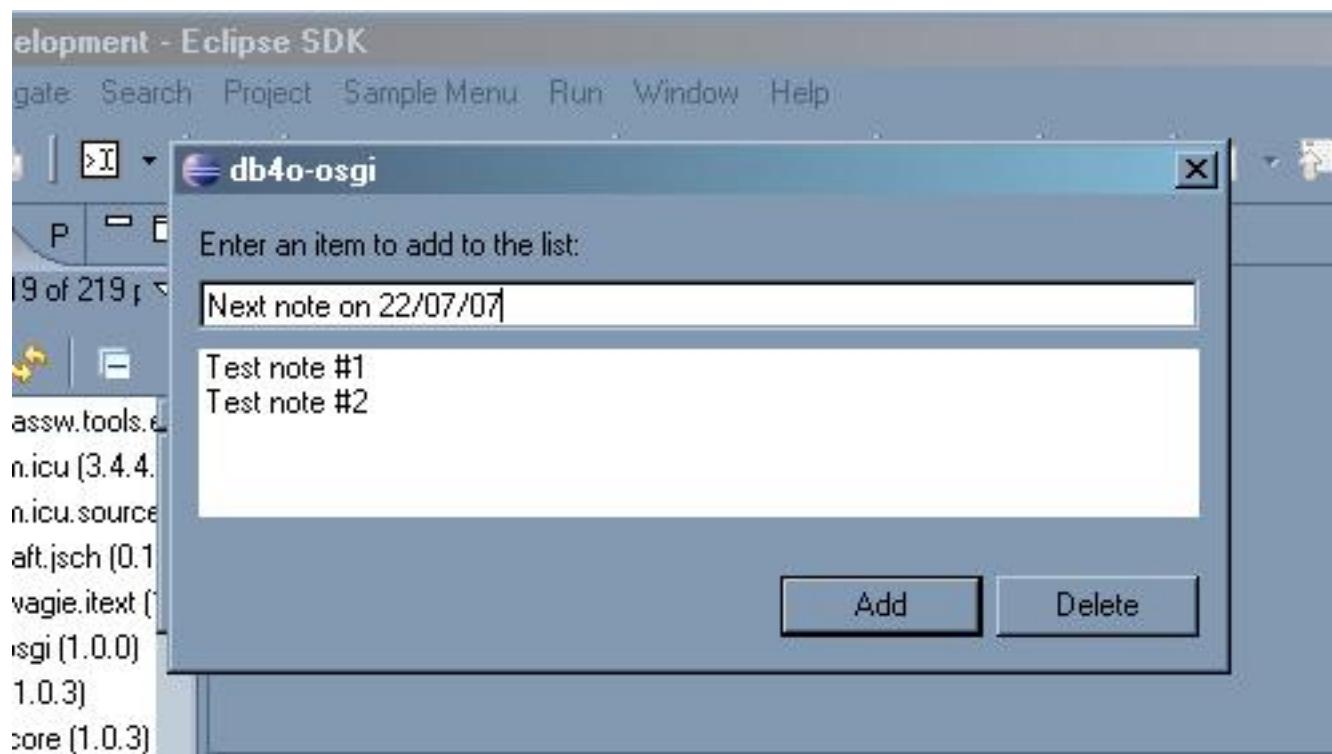
# Testing MemoPlugin

This topic applies to Java version only.

Eclipse environment makes it very easy to test plug-in projects: you do not need to exit your workspace or manually activate plug-ins.

In order to test the new MemoPlugin, open plugin.xml by double-clicking the file in Package Explorer, select Overview tab and click "Launch an Eclipse application" link.

If everything worked out right, you should see "Sample Menu" with a "Sample Action" submenu. Click "Sample Action", you should see a window like this:



Try to add and delete several items. You can close the window and open it again to check that the changes are saved. You can also test the same after restarting Eclipse to see that the changes are not lost between sessions.

Note, that the database file is created in the ECLIPSE\_HOME, so you must have write and create writes there.

This revision (2) was last Modified 2007-07-29T14:50:05 by Tetyana.

# Isolated Storage

This topic applies to .NET version only.

Isolated Storage is a special data storage mechanism provided by .NET framework.

Isolated Storage can allocate storage space for each user and each .NET assembly on the system. It solves the problem of finding a disk location with sufficient privileges for the application data storage. The code using Isolated Storage is only required to have `IsolatedStorageFilePermission`. Though Isolation Storage can allocate space for any user, special quotas exist for its size, which can be configured by system administrator.

You can make use of Isolated Storage as a location for your db4o database.

The following topics will discuss the usage of db4o over Isolated Storage, its benefits and limitations.

More Reading:

- [Isolated Storage IO Adapter](#)
- [Isolated Storage Example](#)
- [Benefits And Limitations](#)
- [Db4oIsolatedStorageFile](#)
- [IsolatedStorageFileAdapter](#)

This revision (2) was last Modified 2007-08-19T13:00:47 by Tetyana.

# Isolated Storage IO Adapter

This topic applies to .NET version only

In order to access Isolated Storage you must use `IsolatedStorageFileStream`, which is a class derived from `FileStream`. db4o by default uses [RandomAccessFileAdapter](#) or [CachedIoAdapter](#) (depending on version), which under the hood are using `FileStream` to read and write db4o database file. As `IsolatedStorageFileStream` provides the same interface as `FileStream` we can use it with a custom IO adapter to make db4o store its data in the Isolated Storage.

To start with let's take `Db4objects.Db4o.IO.RandomAccessFileAdapter` and `Sharpen.IO.RandomAccessFile` from db4o source code and modify them to meet our needs. Basicly, we must change all occurences of `FileStream` to `IsolatedStorageFileStream`, and modify file system function like create or delete to use Isolated Storage semantics.

First of all let's create new classes: `Db4oIsolatedStorageFile` and `IsolatedStorageFileAdapter`, and copy the contents of `RandomAccessFile` and `RandomAccessFileAdapter` accordingly into them.

The constructor for the new `Db4oIsolatedStorageFile` needs to be changed like this:

The rest of `Db4oIsolatedStorageFile` copies the syntax of the `RandomAccessFile`. The result of the modification can be reviewed here: [Db4oIsolatedStorageFile](#).

In the `IsolatedStorageFileAdapter` we will need to make the following changes:

- change the references of `RandomAccessFile` to `Db4oIsolatedStorage` file (constructor and `Open` method);
- remove `LockFile` and `UnlockFile` method calls (constructor and `Close` method);
- remove the call to translate file path to canonical path from the constructor;
- modify `Delete` and `Exists` methods to use the Isolated Storage instead of file system.

The result of the modification can be reviewed here: [IsolatedStorageFileAdapter](#).

The [next topic](#) will show a small example of db4o usage over isolated storage.

This revision (3) was last Modified 2007-08-19T13:36:48 by Tetyana.

# Isolated Storage Example

This topic applies to .NET version only

In order to use our special [IsolatedStorageFileAdapter](#) the following configuration method should be called.

In this example we will use [debug message level](#) = 1 to see what is happening in the console.

Let's try to save some objects and retrieve them. Remember, that the path in the Isolated Storage is not the path on the file system. For simplicity you can specify the database file with only the file name. For more information about creating directory structure in the Isolated Storage, please refer to the appropriate documentation.

Run the following examples and check the output in the console to see how the [IsolatedStorageFileAdapter](#) works:

There is one more thing to remember: delete file operation does not belong to db4o API, so if we need to delete the database from within the application we will need to use Isolated Storage functions:

This revision (2) was last Modified 2007-08-19T13:36:31 by Tetyana.

# Benefits And Limitations

This topic applies to .NET version only

The advantage of using Isolated Storage is in the ability to get storage space for any user rights. However, there are serious limitations, which should be considered before choosing Isolated Storage:

1. Isolated Storage quotas can be assigned to a user by system administrator, so the application must be ready to face a situation when the database file has no more space available to grow (in this case db4o switches to the [ReadOnly](#) mode automatically).
2. Isolated Storage is intended for a per-user use, so it does not make sense to try Isolated Storage for a multi-user access to db4o.
3. Code must have the `IsolatedStorageFilePermission` to work with isolated storage. This can be a problem if your machine is severely locked down by administrative security policies.

This revision (2) was last Modified 2007-08-19T13:40:12 by Tetyana.

# Db4oIsolatedStorageFile

This topic applies to .NET version only

The full example code can be downloaded by clicking on the top right button of the code block.

This revision (3) was last Modified 2007-08-19T13:28:08 by Tetyana.

# IsolatedStorageFileAdapter

This topic applies to .NET version only

The full example code can be downloaded by clicking on the top right button of the code block.

This revision (1) was last Modified 2007-08-19T13:27:27 by Tetyana.

# Android

## Contents

- [Step by Step installation](#)
- [Things to remember when using db4o under Android](#)

This topic applies to Java version only

[Android](#) is a new complete, open and free mobile platform. Android offers developers a java based software developer kit with lots of helpful APIs, including geolocation services. Of course, there is a database support as well: Android has a built-in support for SQLite database. The basic API is similar to standard JDBC API, however some effort was added to create some convenience methods:

```
public int delete(String table, String whereClause, String[] whereArgs)

public long insert(String table, String nullColumnHack, ContentValues values)

public Cursor query(boolean distinct, String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)

public long replace(String table, String nullColumnHack, ContentValues initialValues)
```

This may look better than SQL, but if you look closer you can see that all column names and selection criteria are specified as strings, so we still stay with a problem of run-time checking instead of compile-time.

Luckily even for this very early Android release we already have an alternative - db4o. Yes, db4o runs on Android out of the box and produces very competitive results as well. The following application compares db4o and SQLite usage for basic operations. It also contains some operation duration calculations that can be used to compare db4o vs SQLite performance. Please, note that these results are for an overview purpose only and there are many configuration settings that can change performance of both databases. Also running in emulator does not guarantee the same results as in real device. You can download the whole application code [here](#).

More Reading:

- [General Info](#)
- [Application Structure](#)
- [Opening A Database](#)

- [Storing Data](#)
- [Retrieving Data](#)
- [Changing Data](#)
- [Deleting Data](#)
- [Backup](#)
- [Closing A Database](#)
- [Schema Evolution](#)
- [Car](#)
- [Pilot](#)

## Step by Step installation

The steps to install db4o on Android are quite simple thanks to Eclipse.

1. Point your browser to <http://developer.db4o.com/files/default.aspx>. Here you have to choose stable, production or development release (I recommend the current production version) and then download the Java version of db4o.
2. Copy **db4o-xxx-java1.1.jar** available in the lib folder of your db4o installation to a lib folder in your Android project root directory.
3. Refresh the Eclipse project folders, click on lib, right-click on **db4o-xxx-java1.1.jar** and select "Add to build path".
4. You're done! You can now use db4o in your Android application and it will be deployed automatically when running the Android emulator.

## Things to remember when using db4o under Android

1. Use the Java 1.1 version of the db4o library available on the lib folder of your db4o installation (**db4o-xxx-java1.1.jar**). The other db4o Java versions still suffer from some issues with the way in which Android handles reflection.
2. Open the database relative to your parent activity (context) data directory:

```
Db4o.openFile(dbConfig(), db4oDBFullPath(context));  
private String db4oDBFullPath(Context ctx) {  
    return ctx.getDataDir() + " / " + "dbfile.yap";  
}
```

Otherwise Android security will prevent the creation of the database file.

This revision (8) was last Modified 2007-12-11T20:42:18 by German Viscuso.

# General Info

## Contents

- [Access Control](#)
- [Referential Integrity](#)
- [Transactions](#)
- [Database Size](#)

Both db4o and SQLite are embedded databases, i.e. they run within an application process, removing the overhead associated with a client-server configuration, although db4o can also be used in client-server mode. Both db4o and SQLite offer zero-configuration run modes, which allows to get the database up and running immediately.

## Access Control

---

SQLite relies solely on the file system for its database permissions and has no concept of user accounts. SQLite has database-level locks and does not support client/server mode.

db4o can use encryption or client/server mode for user access control. Client/server can also be used in embedded mode, i.e on the same device.

## Referential Integrity

---

Traditionally referential integrity in relational databases is implemented with the help of foreign keys. However SQLite [does not support](#) this feature. In db4o referential integrity is imposed by the object model, i.e. you can't reference an object that does not exist.

## Transactions

---

Both db4o and SQLite support ACID transactions.

In db4o all the work is transactional: transaction is implicitly started when the database is open and closed either by explicit commit() call or by close() call through implicit commit. Data is protected from system crash during all application lifecycle. If a crash occurs during commit itself, the commit will be restarted when the system is up again if the system had enough time to write the list of changes, otherwise the transaction will be rolled back to the last safe state.

In SQLite autocommit feature is used by default: transaction is started when a SQL command other than SELECT is executed and commit is executed as soon as pending operation is finished. Explicit BEGIN

and END TRANSACTION(COMMIT) or ROLLBACK can be used alternatively to specify user-defined transaction limits. Database crash always results in pending transaction rollback. Nested transactions are not supported.

## Database Size

---

Though embeddable db4o and SQLite support big database files:

- db4o up to 256 GB
- SQLite up to 2TB

all the data is stored in a single database file. db4o also supports clustered databases.

This revision (3) was last Modified 2007-12-09T08:32:45 by Tetyana.

# Application Structure

The presented application consists of simple screen with several buttons, which trigger database operations. The operations are performed on db4o and SQLite database serially and the results with the calculated time of execution are written on the screen. All the database access logic is encapsulated in Db4oExample and SqlExample classes accordingly.

The data stored to the databases is represented by [Car](#) and [Pilot](#) classes.

It is recommended to install the application and test described functionality while reading along.

This revision (1) was last Modified 2007-11-26T18:44:19 by Tetyana.

# Opening A Database

The database is opened with "Open DB" button.

SQLite:

SqlExample.java: database

```

01 public static SQLiteDatabase database() {
02     long startTime = 0;
03     try {
04         _db = _context.openDatabase(DATABASE_NAME, null);
05     } catch (FileNotFoundException e) {
06         try {
07             _db =
08                 _context.createDatabase(DATABASE_NAME, DATABASE_VERSION, 0,
09                                     null);
10             _db.execSQL("create table " + DB_TABLE_PILOT + " (" +
11                         + "id integer primary key autoincrement, " +
12                         + "name text not null, " +
13                         + "points integer not null);");
14             // Foreign key constraint is parsed but not enforced
15             // Here it is used for documentation purposes
16             _db.execSQL("create table " + DB_TABLE_CAR + " (" +
17                         + "id integer primary key autoincrement, " +
18                         + "model text not null, " +
19                         + "pilot integer not null, " +
20                         + "FOREIGN KEY (pilot) " +
21                         + "REFERENCES pilot(id) on delete cascade);");
22             _db.execSQL("CREATE INDEX CAR_PILOT ON " + DB_TABLE_CAR + " (pilot);");
23     } catch (FileNotFoundException e1) {
24         _db = null;
25     }
26 }
27 logToConsole(startTime, "Database opened: ", false);
28 return _db;
29 }
```

db4o:

## Db4oExample.java: database

```

01 public static ObjectContainer database() {
02     long startTime = 0;
03     try {
04         if (_container == null) {
05             startTime = System.currentTimeMillis();
06             _container = Db4o.openFile(configure(), db4oDBFullPath());
07         }
08     } catch (Exception e) {
09         Log.e(Db4oExample.class.getName(), e.toString());
10         return null;
11     }
12     logToConsole(startTime, "Database opened: ", false);
13     return _container;
14 }
```

## Db4oExample.java: configure

```

1 private static Configuration configure() {
2     Configuration configuration = Db4o.newConfiguration();
3     configuration.objectClass(Car.class).objectField("pilot").indexed(true);
4     configuration.objectClass(Pilot.class).objectField("points").indexed(true);
5     configuration.lockDatabaseFile(false);
6
7     return configuration;
8 }
```

db4o code is a bit more compact, but the main advantage of db4o is in the fact that all APIs are pure java, they are compile-time checked and can be transferred into IDE templates (database opening should be a template as it most probably be the same for all your db4o applications including tests).

This revision (2) was last Modified 2007-12-09T08:39:34 by Tetyana.

# Storing Data

"Store" button creates and stores 100 of car objects (each including a reference to Pilot object) to each database.

SQLite:

SqlExample.java: fillUpDB

```

01  public static void fillUpDB() throws Exception {
02      close();
03      _context.deleteDatabase(DATABASE_NAME);
04      SQLiteDatabase db = database();
05      if (db != null){
06          long startTime = System.currentTimeMillis();
07          for (int i=0; i<100; i++){
08              addCar(db, i);
09          }
10          logToConsole(startTime, "Stored 100 objects: ", false);
11          startTime = System.currentTimeMillis();
12      }
13  }
```

SqlExample.java: addCar

```

01 private static void addCar(SQLiteDatabase db, int number)
02 {
03     ContentValues initialValues = new ContentValues();
04
05     initialValues.put("id", number);
06     initialValues.put("name", "Tester");
07     initialValues.put("points", number);
08     db.insert(DB_TABLE_PILOT, null, initialValues);
09 }
```

```

10 |     initialValues = new ContentValues();
11 |
12 |     initialValues.put("model", "BMW");
13 |     initialValues.put("pilot", number);
14 |     db.insert(DB_TABLE_CAR, null, initialValues);
15 | }

```

db4o:

## Db4oExample.java: fillUpDB

```

01 public static void fillUpDB() throws Exception {
02     close();
03     new File(db4oDBFullPath()).delete();
04     ObjectContainer container=database();
05     if (container != null){
06         long startTime = System.currentTimeMillis();
07         for (int i=0; i<100; i++){
08             addCar(container, i);
09         }
10         logToConsole(startTime, "Stored 100 objects: ", false);
11         startTime = System.currentTimeMillis();
12         container.commit();
13         logToConsole(startTime, "Committed: ", true);
14     }
15 }

```

## Db4oExample.java: addCar

```

1 private static void addCar(ObjectContainer container, int points)
2 {
3     Car car = new Car("BMW");
4     car.setPilot(new Pilot("Tester", points));
5     container.set(car);

```

You can see that db4o handles adding objects to the database in a much more elegant way - #set(object) method is enough. In SQLite case it is much more difficult as you must store different objects into different tables. Some of the additional work that SQLite developer will have to do is not visible in this example, i.e:

- the developer will have to ensure that the sequence of insert commands starts from children objects and goes up to the parent (this can be a really difficult task for relational models including lots of foreign key dependencies);
- in most cases the data for insertion will come from business objects, which will mean that the object model will have to be transferred to relational model.

This revision (3) was last Modified 2007-12-09T08:46:52 by Tetyana.

# Retrieving Data

In order to test the retrieval abilities of both databases we will try to select a car with a pilot having 9 points:

SQLite:

SqlExample.java: selectCar

```

01 public static void selectCar() {
02     SQLiteDatabase db = database();
03     if (db != null) {
04         long startTime = System.currentTimeMillis();
05         Cursor c =
06             db.query("select c.model, p.name, p.points from car c, pilot p where c.
07 pilot = p.id and p.points = 9;", null);
08         if (c.count() == 0) {
09             logToConsole(0, "Car not found, refill the database to continue.", false);
10             return;
11         }
12         c.moveToFirst();
13         Pilot pilot = new Pilot();
14         pilot.setName(c.getString(1));
15         pilot.setPoints(c.getInt(2));
16
17         Car car = new Car();
18         car.setModel(c.getString(0));
19         car.setPilot(pilot);
20         logToConsole(startTime, "Selected Car (" + car + "): ", false);
21     }

```

db4o:

(Using SODA query)

Db4oExample.java: selectCar

```

01 public static void selectCar() {
02     ObjectContainer container = database();
03     if (container != null) {
04         Query query = container.query();

```

```
05 |     query.constraint(Car.class);
06 |     query.descend("pilot").descend("points").constraint(new Integer(9));
07 |
08 |     long startTime = System.currentTimeMillis();
09 |     ObjectSet result = query.execute();
10 |     if (result.size() == 0){
11 |         logToConsole(0, "Car not found, refill the database to continue.", false);
12 |     } else {
13 |         logToConsole(startTime, "Selected Car (" + result.next() + "): ", false);
14 |     }
15 | }
16 }
```

Of course SODA query is not the best db4o querying mechanism: the preferred mechanism - Native Queries - will be reviewed in the following chapters. However, SODA is the closest to SQL and can serve a good comparison. In the example above you can see that SQLite needs a lot of additional code to transfer the retrieved data into application's objects, whereas db4o does not need this code at all, as the result is already a collection of objects.

This revision (2) was last Modified 2007-12-09T08:59:14 by Tetyana.

# Changing Data

For this test we will select and update a car with a new pilot, where existing pilot has 15 points:

SQLite:

SqlExample.java: updateCar

```

01 public static void updateCar(){
02     SQLiteDatabase db = database();
03     if (db != null){
04         long startTime = System.currentTimeMillis();
05         // insert a new pilot
06         ContentValues updateValues = new ContentValues();
07
08         updateValues.put("id", 101);
09         updateValues.put("name", "Tester1");
10         updateValues.put("points", 25);
11         db.insert(DB_TABLE_PILOT, null, updateValues);
12
13         updateValues = new ContentValues();
14
15         // update pilot in the car
16         updateValues.put("pilot", 101);
17         int count = db.update(DB_TABLE_CAR, updateValues, "pilot in (select id from pilot
where points = 15)", null);
18     if (count == 0){
19         logToConsole(0, "Car not found, refill the database to continue.", false);
20     } else {
21         logToConsole(startTime, "Updated selected object: ", false);
22     }
23 }
24 }
```

db4o:

(Select Car using Native Query)

Db4oExample.java: updateCar

```

01 public static void updateCar(){
02     ObjectContainer container=database();
```

```

03 |         if (container != null){
04 |             try {
05 |                 long startTime = System.currentTimeMillis();
06 |                 ObjectSet result = container.query(new Predicate() {
07 |                     public boolean match(Object object) {
08 |                         if (object instanceof Car) {
09 |                             return ((Car) object).getPilot().getPoints() == 15;
10 |                         }
11 |                         return false;
12 |                     }
13 |                 });
14 |                 Car car = (Car) result.next();
15 |                 car.setPilot(new Pilot("Tester1", 25));
16 |                 container.set(car);
17 |                 logToConsole(startTime, "Updated selected object: ", false);
18 |             } catch (Exception e) {
19 |                 logToConsole(0, "Car not found, refill the database to continue.", false);
20 |             }
21 |         }
22 |

```

In this example db4o and SQLite actually behave quite differently. For SQLite in order to update a pilot in an existing car in the database the following actions are needed:

1. A new pilot should be created and saved to the database.
2. New pilot's primary key (101) should be retrieved (not shown in this example, but is necessary for a real database application).
3. An update statement should be issued to replace pilot field in the car table.

For db4o database the sequence will be the following:

1. Retrieve the car from the database
2. Update the car with a new pilot object

As you can see the only benefit of SQLite API is that the car can be selected and updated in one statement. But in the same time there are serious disadvantages:

- A new pilot record should be created absolutely separately (in a real database will also include ORM)
- The pilot's ID needs to be retrieved separately (we must make sure that it is a correct id)

In db4o we avoid these disadvantages as creating new pilot and updating the car value are actually combined in one atomic operation.

This revision (2) was last Modified 2007-12-09T09:33:29 by Tetyana.

# Deleting Data

The following methods will delete a car with a pilot having 5 points from each database:

SQLite:

**SqlExample.java: deleteCar**

```

01  public static void deleteCar() {
02 |     SQLiteDatabase db = database();
03 |     if (db != null) {
04 |         long startTime = System.currentTimeMillis();
05 |         int count = db.delete(DB_TABLE_CAR, "pilot in (select id from pilot where points
= 5)", null);
06 |         if (count == 0) {
07 |             logToConsole(0, "Car not found, refill the database to continue.", false);
08 |         } else {
09 |             logToConsole(startTime, "Deleted selected object: ", false);
10 |         }
11 |
12 |     }
13 }
```

db4o:

(Select Car using Native Query)

**Db4oExample.java: deleteCar**

```

01  public static void deleteCar() {
02 |     ObjectContainer container = database();
03 |     if (container != null) {
04 |         try {
05 |             long startTime = System.currentTimeMillis();
06 |             ObjectSet result = container.query(new Predicate() {
07 |                 public boolean match(Object object) {
08 |                     if (object instanceof Car) {
09 |                         return ((Car) object).getPilot().getPoints() == 5;
10 |                     }
11 |                     return false;
12 |                 }
13 |             });
14 |         }
```

```

14 |             Car car = (Car) result.next();
15 |             container.delete(car);
16 |             logToConsole(startTime, "Deleted selected object: ", false);
17 | } catch (Exception e){
18 |     logToConsole(0, "Car not found, refill the database to continue.", false);
19 | }
20 | }
21 | }
```

In this example db4o code looks much longer. But should we consider it a disadvantage? My opinion is - NO. Of course, SQLite seems to handle the whole operation in just one statement: db.delete(). But if you look attentively you will see that basically this statement just transfers all the difficult job to SQL: SQL statement should select a pilot with a given condition, then find a car. Using SQL can look shorter but it has a great disadvantage - it uses strings. So what will happen if the statement is wrong? You will never notice it till somebody in the running application will cause this statement to execute. Even then you might not see the reason immediately. The same applies to the schema changes - you may not even notice that you are using wrong tables and fields.

db4o helps to avoid all the above mentioned problems: query syntax is completely compile-checked and schema evolution will be spotted immediately by the compiler, so that you would not need to rely on code search and replace tools.

This revision (3) was last Modified 2007-12-09T09:52:14 by Tetyana.

# Backup

"Backup" button shows database backup ability.

On db4o a ExtObjectContainer#backup call is used to backup a database in use.

SQLite does not support a special API to make a backup. However, as you remember SQLite database is stored in a single database file, so the backup can be simply a matter of copying the database file. Unfortunately, this can't be done if the database is in use. In this case you can use [Android Debug Bridge](#) (adb) tool to access sqlite3 command-line application, which has .dump command for backing up database contents while the database is in use:

```
E:\>adb shell
```

```
# sqlite3 /data/data/com.db4odoc.android.compare/databases/android.db
```

```
sqlite3 /data/data/com.db4odoc.android.compare/databases/android.db
```

```
SQLite version 3.5.0
```

```
Enter ".help" for instructions
```

```
sqlite> .dump > android200711.dmp
```

```
.dump > android200711.dmp
```

```
BEGIN TRANSACTION;
```

```
COMMIT;
```

```
sqlite>.exit
```

```
.exit
```

```
# ^D
```

Ctrl+D command is used to close adb session.

This revision (1) was last Modified 2007-11-26T19:03:40 by Tetyana.

# Closing A Database

The following methods will close SQLite and db4o database accordingly:

SQLite:

**SqlExample.java: close**

```

01 ✉✉ /**
02 |     * Close database connection
03 | */
04 ✉✉     public static void close() {
05 ✉✉         if(_db != null){
06 |             long startTime = System.currentTimeMillis();
07 |             _db.close();
08 |             logToConsole(startTime, "Database committed and closed: ", false);
09 |             _db = null;
10 |         }
11 |     }

```

db4o:

**Db4oExample.java: close**

```

01 ✉✉ /**
02 |     * Close database connection
03 | */
04 ✉✉     public static void close() {
05 ✉✉         if(_container != null){
06 |             long startTime = System.currentTimeMillis();
07 |             _container.close();
08 |             logToConsole(startTime, "Database committed and closed: ", false);

```

```
09 |         _container = null;  
10 |     }  
11 | }
```

This revision (2) was last Modified 2007-11-26T19:05:28 by Tetyana.

# Schema Evolution

## Contents

- Schema Evolution in db4o
- Schema Evolution in SQLite
- Conclusion

When a new application development is considered it is important to think about its evolution. What happens if your initial model does not suffice and you need changes or additions? Let's look how db4o and SQLite applications can handle it.

To keep the example simple, let's add a registration record to our car:

RegistrationRecord.java

```
01 package com.db4odoc.android.compare.refactored;
02
03 import java.util.Date;
04
05 public class RegistrationRecord {
06     private String number;
07     private Date year;
08
09     public RegistrationRecord(String number, Date year) {
10         this.number = number;
11         this.year = year;
12     }
13
14     public String getNumber() {
15         return number;
16     }
17
18     public void setNumber(String number) {
19         this.number = number;
20     }
21
22     public Date getYear() {
23         return year;
24     }
25
26     public void setYear(Date year) {
27         this.year = year;
28     }
```

```
29 |
30 |
31 |
32 }
```

Now we will need to modify Car class to attach the record:

Car.java

```
01 /* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */
02 package com.db4odoc.android.compare.refactored;
03
04 import java.text.DateFormat;
05 import java.text.SimpleDateFormat;
06 import java.util.Calendar;
07 import java.util.GregorianCalendar;
08
09
10 public class Car {
11     private String model;
12     private Pilot pilot;
13     private RegistrationRecord registration;
14
15     public RegistrationRecord getRegistration() {
16         return registration;
17     }
18
19     public void setRegistration(RegistrationRecord registration) {
20         this.registration = registration;
21     }
22
23     public Car() {
24
25     }
26
27     public Car(String model) {
28         this.model = model;
29         this.pilot=null;
30     }
31
32     public Pilot getPilot() {
33         return pilot;
```

```

34 |     }
35 |
36 |     public void setPilot(Pilot pilot) {
37 |         this.pilot = pilot;
38 |     }
39 |
40 |     public String getModel() {
41 |         return model;
42 |     }
43 |
44 |     public String toString() {
45 |         if (registration == null){
46 |             return model + "[" + pilot + "]";
47 |         } else {
48 |             DateFormat df = new SimpleDateFormat("d/M/yyyy");
49 |             return model + ":" + df.format(registration.getYear());
50 |         }
51 |     }
52 |
53 |     public void setModel(String model) {
54 |         this.model = model;
55 |     }
56 |
57 | }

```

Ok, the application is changed to cater for new class. What about our databases?

## Schema Evolution in db4o

db4o supports such schema change on the fly: we can select values and update the new field too:

Db4oExample.java: selectCarAndUpdate

```

01 |     public static void selectCarAndUpdate() {
02 |         ObjectContainer container = database();
03 |         if (container != null){
04 |             Query query = container.query();
05 |             query.constrain(Car.class);
06 |             query.descend("pilot").descend("points").constrain(new Integer(15));
07 |
08 |             long startTime = System.currentTimeMillis();
09 |             ObjectSet result = query.execute();

```

```

10 |     result.reset();
11 |     if (!result.hasNext()) {
12 |         logToConsole(0, "Car not found, refill the database to continue.", false);
13 |     } else {
14 |         Car car = (Car)result.next();
15 |         logToConsole(startTime, "Selected Car (" + car + "): ", false);
16 |         startTime = System.currentTimeMillis();
17 |         car.setRegistration(new RegistrationRecord("A1", new Date(System.currentTimeMillis())));
18 |         logToConsole(startTime, "Updated Car (" + car + "): ", true);
19 |     }
20 |
21 | }
22 |

```

## Schema Evolution in SQLite

For SQLite database model should be synchronized with the object model:

SqlExample.java: upgradeDatabase

```

1 public static void upgradeDatabase(SQLiteDatabase db) {
2     db.execSQL("create table REG_RECORDS (
3             + "id text primary key, " + "year date);");
4     db.execSQL("CREATE INDEX IDX_REG_RECORDS ON REG_RECORDS (id);");
5     db.execSQL("alter table " + DB_TABLE_CAR + " add reg_record text;");
6
7 }

```

Now we can try to retrieve and update records:

SqlExample.java: selectCarAndUpdate

```

01 public static void selectCarAndUpdate() {
02     SQLiteDatabase db = database();
03     if (db != null) {
04         long startTime = System.currentTimeMillis();
05
06         db.execSQL("insert into REG_RECORDS (id,year) values ('A1', DATETIME('NOW'))");
07         ContentValues updateValues = new ContentValues();
08
09         // update car
10         updateValues.put("reg_record", "A1");

```

```

11 |         int count = db.update(DB_TABLE_CAR, updateValues, "pilot in (select id from pilot
where points = 15)", null);
12 |         if (count == 0){
13 |             logToConsole(0, "Car not found, refill the database to continue.", false);
14 |         } else {
15 |             Cursor c =
16 |                 db.query("select c.model, r.id, r.year from car c, " +
17 |                         "REG_RECORDS r, pilot p where c.reg_record = r.id " +
18 |                         "and c.pilot = p.id and p.points = 15;", null);
19 |         if (c.count() == 0) {
20 |             logToConsole(0, "Car not found, refill the database to continue.", false);
21 |             return;
22 |         }
23 |         c.moveToFirst();
24 |         String date = c.getString(2);
25 |         SimpleDateFormat sf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
26 |         try {
27 |             Date dt = sf.parse(date);
28 |             RegistrationRecord record = new RegistrationRecord(c.getString(1), dt);
29 |
30 |             Car car = new Car();
31 |             car.setModel(c.getString(0));
32 |             car.setRegistration(record);
33 |             logToConsole(startTime, "Updated Car (" + car + "): ", true);
34 |         } catch (ParseException e){
35 |             Log.e(Db4oExample.class.getName(), e.toString());
36 |         }
37 |
38 |     }
39 | }
40 | }
```

## Conclusion

You can see that schema evolution is much easier with db4o. But the main difficulty that is not visible from the example is that schema evolution with SQLite database can potentially introduce a lot of bugs that will be difficult to spot. For more information see [Refactoring and Schema Evolution](#).

This revision (1) was last Modified 2007-12-09T17:12:39 by Tetyana.

# Car

Car.java

```
01 /* Copyright (C) 2004 - 2007 db4objects Inc. http://www.db4o.com */  
02 package com.db4odoc.android.compare;  
03  
04  
05 public class Car {  
06     private String model;  
07     private Pilot pilot;  
08  
09     public Car() {  
10  
11     }  
12  
13     public Car(String model) {  
14         this.model = model;  
15         this.pilot = null;  
16     }  
17  
18     public Pilot getPilot() {  
19         return pilot;  
20     }  
21  
22     public void setPilot(Pilot pilot) {  
23         this.pilot = pilot;  
24     }  
25  
26     public String getModel() {  
27         return model;  
28     }
```

```
29 |
30     public String toString() {
31         return model + "[ " + pilot + " ]";
32     }
33
34     public void setModel(String model) {
35         this.model = model;
36     }
37 }
```

This revision (1) was last Modified 2007-11-26T18:48:10 by Tetyana.

# Pilot

Pilot.java

```
01 /* Copyright (C) 2004 - 2007 db4objects Inc. http://www.db4o.com */
02 package com.db4odoc.android.compare;
03
04 public class Pilot {
05     private String name;
06     private int points;
07
08     public Pilot() {
09
10    }
11
12     public Pilot(String name, int points) {
13         this.name=name;
14         this.points=points;
15    }
16
17     public int getPoints() {
18         return points;
19    }
20
21     public void addPoints(int points) {
22         this.points+=points;
23    }
24
25     public String getName() {
26         return name;
27    }
28}
```

```
29 public String toString() {  
30 |         return name + "/" + points;  
31 |     }  
32 |  
33 public void setName(String name) {  
34 |     this.name = name;  
35 | }  
36 |  
37 public void setPoints(int points) {  
38 |     this.points = points;  
39 | }  
40 }
```

This revision (1) was last Modified 2007-11-26T18:49:33 by Tetyana.

# Tuning

This topic set explains different configuration, debugging and diagnostics issues. This information will help you to fine-tune your db4o usage and chase away bugs and performance pitfalls.

More Reading:

- [Main Operations Performance](#)
- [IO Benchmark Tools](#)
- [Configuration](#)
- [Selective Persistence](#)
- [Indexing](#)
- [Performance Hints](#)
- [Debugging db4o](#)
- [Diagnostics](#)
- [Native Query Optimization](#)
- [Utility Methods](#)

This revision (2) was last Modified 2007-05-07T18:04:42 by Tetyana.

# Main Operations Performance

One of the most important factors in database usage is performance. In the same time it is something difficult to measure and predict as there are too many factors affecting it. These factors can be dependent or independent of database implementation. Independent factors, such as operating memory, processor speed etc are general for all applications and in many cases are given as initial conditions, which do not allow frequent or tuning at all (for example, embedded mobile devices). On the other hand, dependent factors can usually be changed programmatically and provide valuable effect.

The following articles will give you some average numbers of db4o performance, providing the testing code that can be easily modified to accommodate your object model and environment and pointing out the most influencing performance factors.

More Reading:

- [Insert Performance](#)

This revision (2) was last Modified 2008-01-11T16:29:10 by Tetyana.

# Insert Performance

The following chapters provide some performance testing examples, revealing the most influential performance factors. Together with the examples there are some approximate time measurement values that were achieved on a Toshiba Sattelite Pro A120 notebook with 1Gb RAM 120GB ATA drive running on Vista. Please, note that these values are not guaranteed and can vary considerably depending on a hardware and software used.

In most of the tests the following simple object was used:

InsertPerformanceBenchmark.java: Item

```
01 public static class Item {  
02 |  
03 |     public String _name;  
04 |     public Item _child;  
05 |  
06     public Item(){  
07 |  
08     }  
09 |  
10     public Item(String name, Item child){  
11 |         _name = name;  
12 |         _child = child;  
13 |     }  
14 }
```

In the tests Item objects were created with 3 levels of embedded Item objects. The amount of objects was varied for different tests.

Please, be cautious to compare results of different tests presented as different configurations are used in each test.

More Reading:

- [Hardware Resources](#)
- [Local And Remote Modes](#)
- [Commit Frequency](#)
- [Object Structure](#)
- [Indexes](#)
- [Inherited Objects](#)
- [Configuration Options](#)

This revision (1) was last Modified 2007-12-30T14:26:32 by Tetyana.

# Hardware Resources

Initial object storing requires little calculation, but can be resource consuming on disk access. Therefore the main hardware resource that will affect db4o insert performance is the hard drive. The faster is the hard drive the better performance you will get.

An alternative to a hard drive database storage can be a database file stored in RAM. This can be done by placing the database file in a designated RAM-drive or by using db4o memory io-adapter:

Java:

```
configuration.io(new MemoryIoAdapter());
```

The following test can be performed to compare performance of a hard drive and a RAM drive:

InsertPerformanceBenchmark.java: runRamDiskTest

```
01 private void runRamDiskTest() {
02 |
03 |     configureRamDrive();
04 |
05 |     initForHardDriveTest();
06 |     clean();
07 |     System.out.println("Storing " + _count + " objects of depth " + _depth + " on a hard
drive:");
08 |     open();
09 |     store();
10 |     close();
11 |
12 |     initForRamDriveTest();
13 |     clean();
14 |     System.out.println("Storing " + _count + " objects of depth " + _depth + " on a RAM
disk:");
15 |     open();
16 |     store();
17 |     close();
18 |
19 } }
```

InsertPerformanceBenchmark.java: configureRamDrive

```
1 private void configureRamDrive() {
2 |     Configuration config = Db4o.configure();
```

```

3 |         config.lockDatabaseFile(false);
4 |         config.weakReferences(false);
5 |         config.flushFileBuffers(true);
6 |     }

```

## InsertPerformanceBenchmark.java: initForHardDriveTest

```

1 private void initForHardDriveTest() {
2     _count = 30000;
3     _depth = 3;
4     _filePath = "performance.db4o";
5     _isClientServer = false;
6
7 }

```

## InsertPerformanceBenchmark.java: initForRamDriveTest

```

1 private void initForRamDriveTest() {
2     _count = 30000;
3     _depth = 3;
4     _filePath = "r:\\performance.db4o";
5     _isClientServer = false;
6
7 }

```

## InsertPerformanceBenchmark.java: store

```

01 private void store(){
02     startTimer();
03     for (int i = 0; i < _count; i++) {
04         Item item = new Item("load", null);
05         for (int j = 1; j < _depth; j++) {
06             item = new Item("load", item);
07         }
08         objectContainer.set(item);
09     }
10     objectContainer.commit();
11     stopTimer("Store " + totalObjects() + " objects");
12 }

```

The RAM driver was downloaded [here](#) and installed on R:\\ drive.

The following results were achieved for the [testing configuration](#):

Java:

Storing 100000 objects of depth 3 on a hard drive:

Store 300000 objects: 11912ms

Storing 100000 objects of depth 3 on a RAM disk:

Store 300000 objects: 9351ms

This revision (3) was last Modified 2007-12-30T14:31:07 by Tetyana.

# Local And Remote Modes

Of course local and client/server modes cannot give the same performance and it is difficult to say what will be the impact of inserting the objects over the network, as the network conditions can vary.

You can use the following test to compare the performance on your network:

InsertPerformanceBenchmark.java: runClientServerTest

```

01 private void runClientServerTest() {
02 |
03     configureClientServer();
04 |
05     init();
06     clean();
07     System.out.println("Storing " + _count + " objects of depth " + _depth + " locally:");
08     open();
09     store();
10     close();
11 |
12     initForClientServer();
13     clean();
14     System.out.println("Storing " + _count + " objects of depth " + _depth + " "
15     remotely: );
16     open();
17     store();
18     close();
19 }
```

InsertPerformanceBenchmark.java: configureClientServer

```

1 private void configureClientServer() {
2     Configuration config = Db4o.configure();
3     config.lockDatabaseFile(false);
4     config.weakReferences(false);
5     config.flushFileBuffers(false);
6     config.clientServer().singleThreadedClient(true);
7 }
```

InsertPerformanceBenchmark.java: init

```

1 private void init() {
2     _count = 10000;
3     _depth = 3;
4     _isClientServer = false;
5
6 }

```

## InsertPerformanceBenchmark.java: initForClientServer

```

1 private void initForClientServer() {
2     _count = 10000;
3     _depth = 3;
4     _isClientServer = true;
5     _host = "localhost";
6 }

```

## InsertPerformanceBenchmark.java: store

```

01 private void store() {
02     startTimer();
03     for (int i = 0; i < _count; i++) {
04         Item item = new Item("load", null);
05         for (int j = 1; j < _depth; j++) {
06             item = new Item("load", item);
07         }
08         objectContainer.set(item);
09     }
10     objectContainer.commit();
11     stopTimer("Store " + totalObjects() + " objects");
12 }

```

With a good and reliable network you can use the same methods to improve the insert performance as in a local mode. However, if your network connection is not always perfect you will need to use commits more often to ensure that the objects do not get lost. See the [next chapter](#) for recommendations on commit performance.

This revision (1) was last Modified 2007-12-30T14:33:59 by Tetyana.

# Commit Frequency

Commit is an expensive operation as it needs to physically access hard drive several times and write changes. However, only commit can ensure that the objects are actually stored in the database and won't be lost.

The following test compares different commit frequencies (one commit for all objects or several commits after a specified amount of objects). The test runs against a hard drive:

InsertPerformanceBenchmark.java: runCommitTest

```

01 private void runCommitTest() {
02 |
03 |     configureForCommitTest();
04 |     initForCommitTest();
05 |
06 |     clean();
07 |     System.out.println("Storing objects as a bulk: ");
08 |     open();
09 |     store();
10 |     close();
11 |
12 |     clean();
13 |     System.out.println("Storing objects with commit after each " + _commitInterval + " objects: ");
14 |     open();
15 |     storeWithCommit();
16 |     close();
17 }

```

InsertPerformanceBenchmark.java: configureForCommitTest

```

1 private void configureForCommitTest() {
2     Configuration config = Db4o.configure();
3     config.lockDatabaseFile(false);
4     config.weakReferences(false);
5     // flushFileBuffers should be set to true to ensure that
6     // the commit information is physically written
7     // and in the correct order
8     config.flushFileBuffers(true);
9 }

```

## InsertPerformanceBenchmark.java: initForCommitTest

```

1 ①②③private void initForCommitTest() {
2     _count = 100000;
3     _commitInterval = 10000;
4     _depth = 3;
5     _isClientServer = false;
6
7 }

```

## InsertPerformanceBenchmark.java: store

```

01 ①②③private void store() {
02     startTimer();
03     for (int i = 0; i < _count; i++) {
04         Item item = new Item("load", null);
05         for (int j = 1; j < _depth; j++) {
06             item = new Item("load", item);
07         }
08         objectContainer.set(item);
09     }
10     objectContainer.commit();
11     stopTimer("Store " + totalObjects() + " objects");
12 }

```

## InsertPerformanceBenchmark.java: storeWithCommit

```

01 ①②③private void storeWithCommit() {
02     startTimer();
03     int k = 0;
04     while (k < _count) {
05         for (int i = 0; i < _commitInterval; i++) {
06             Item item = new Item("load", null);
07             k++;
08             for (int j = 1; j < _depth; j++) {
09                 item = new Item("load", item);
10             }
11             objectContainer.set(item);
12         }
13     }
14     objectContainer.commit();

```

```
14 |     }
15 |     objectContainer.commit();
16 |     stopTimer("Store " + totalObjects() + " objects");
17 | }
```

Note, that you can get an OutOfMemory exception when running the part of the test with a single commit. To fix this use -Xmx500m setting for your Java machine.

The following results were achieved for the [testing configuration](#):

Java:

Storing objects as a bulk:

Store 300000 objects: 11974ms

Storing objects with commit after each 10000 objects:

Store 300000 objects: 14692ms

This revision (1) was last Modified 2007-12-30T14:34:56 by Tetyana.

# Object Structure

Object Structure naturally has a major influence on insert performance: inserting one object, which is a linked list of 1000 members, is much slower than inserting an object with a couple of primitive fields.

The following test compares storing time of similar objects with one different field:

InsertPerformanceBenchmark.java: runDifferentObjectsTest

```
01  private void runDifferentObjectsTest() {
02  |
03  |     configure();
04  |     init();
05  |     System.out.println("Storing " + _count + " objects with " + _depth + " levels of
embedded objects:");
06  |
07  |     clean();
08  |     System.out.println(" - primitive object with int field");
09  |     open();
10  |     storeSimplest();
11  |     close();
12  |
13  |     open();
14  |     System.out.println(" - object with String field");
15  |     store();
16  |     close();
17  |
18  |     clean();
19  |     open();
20  |     System.out.println(" - object with StringBuffer field");
21  |     storeWithStringBuffer();
22  |     close();
23  |
24  |     clean();
25  |     open();
26  |     System.out.println(" - object with int array field");
27  |     storeWithArray();
28  |     close();
29  |
30  |     clean();
31  |     open();
```

```

32 |     System.out.println(" - object with ArrayList field");
33 |     storeWithArrayList();
34 |     close();
35 |
36 L }
```

## InsertPerformanceBenchmark.java: configure

```

1 private void configure() {
2     Configuration config = Db4o.configure();
3     config.lockDatabaseFile(false);
4     config.weakReferences(false);
5     config.io(new MemoryIoAdapter());
6     config.flushFileBuffers(false);
7 L }
```

## InsertPerformanceBenchmark.java: init

```

1 private void init() {
2     _count = 10000;
3     _depth = 3;
4     _isClientServer = false;
5
6 L }
```

## InsertPerformanceBenchmark.java: storeSimplest

```

01 private void storeSimplest() {
02     startTimer();
03     for (int i = 0; i < _count; i++) {
04         SimplestItem item = new SimplestItem(i, null);
05         for (int j = 1; j < _depth; j++) {
06             item = new SimplestItem(i, item);
07         }
08         objectContainer.set(item);
09     }
10     objectContainer.commit();
11     stopTimer("Store " + totalObjects() + " objects");
12 L }
```

## InsertPerformanceBenchmark.java: store

```

01  private void store(){
02 |         startTimer();
03 |         for (int i = 0; i < _count ;i++) {
04 |             Item item = new Item("load", null);
05 |             for (int j = 1; j < _depth; j++) {
06 |                 item = new Item("load", item);
07 |             }
08 |             objectContainer.set(item);
09 |         }
10 |         objectContainer.commit();
11 |         stopTimer("Store "+ totalObjects() + " objects");
12 }

```

## InsertPerformanceBenchmark.java: storeWithStringBuffer

```

01  private void storeWithStringBuffer(){
02 |         startTimer();
03 |         for (int i = 0; i < _count ;i++) {
04 |             ItemWithStringBuffer item = new ItemWithStringBuffer(new StringBuffer("load"),
05 | null);
06 |             for (int j = 1; j < _depth; j++) {
07 |                 item = new ItemWithStringBuffer(new StringBuffer("load"), item);
08 |             }
09 |             objectContainer.set(item);
10 |         }
11 |         objectContainer.commit();
12 |         stopTimer("Store "+ totalObjects() + " objects");
13 }

```

## InsertPerformanceBenchmark.java: storeWithArray

```

01  private void storeWithArray(){
02 |         startTimer();
03 |         int[] array = new int[]{1, 2, 3, 4};
04 |         for (int i = 0; i < _count ;i++) {
05 |             int[] id = new int[]{1, 2, 3, 4};
06 |             ItemWithArray item = new ItemWithArray(id, null);
07 |             for (int j = 1; j < _depth; j++) {
08 |                 int[] id1 = new int[]{1, 2, 3, 4};
09 |                 item = new ItemWithArray(id1, item);
10 |             }
11 }

```

## Object Structure

```
11         objectContainer.set(item);
12     }
13     objectContainer.commit();
14     stopTimer("Store " + totalObjects() + " objects");
15 }
```

### InsertPerformanceBenchmark.java: storeWithArrayList

```
01 private void storeWithArrayList(){
02     startTimer();
03     ArrayList idList = new ArrayList();
04     idList.add(1);
05     idList.add(2);
06     idList.add(3);
07     idList.add(4);
08     for (int i = 0; i < _count; i++) {
09         ArrayList ids = new ArrayList();
10         ids.addAll(idList);
11         ItemWithArrayList item = new ItemWithArrayList(ids, null);
12         for (int j = 1; j < _depth; j++) {
13             ArrayList ids1 = new ArrayList();
14             ids1.addAll(idList);
15             item = new ItemWithArrayList(ids1, item);
16         }
17         objectContainer.set(item);
18     }
19     objectContainer.commit();
20     stopTimer("Store " + totalObjects() + " objects");
21 }
```

### InsertPerformanceBenchmark.java: SimplestItem

```
01 public static class SimplestItem {
02
03     public int _id;
04     public SimplestItem _child;
05
06     public SimplestItem()
07     {
08
09         public SimplestItem(int id, SimplestItem child) {
```

```

10 |         _id = id;
11 |         _child = child;
12 |     }
13 | }
```

## InsertPerformanceBenchmark.java: ItemWithArray

```

01 public static class ItemWithArray {
02 |
03 |     public int[] _id;
04 |     public ItemWithArray _child;
05 |
06 public ItemWithArray() {
07 | }
08 |
09 public ItemWithArray(int[] id, ItemWithArray child) {
10 |     _id = id;
11 |     _child = child;
12 | }
13 | }
```

## InsertPerformanceBenchmark.java: ItemWithArrayList

```

01 public static class ItemWithArrayList {
02 |
03 |     public ArrayList _ids;
04 |     public ItemWithArrayList _child;
05 |
06 public ItemWithArrayList() {
07 | }
08 |
09 public ItemWithArrayList(ArrayList ids, ItemWithArrayList child) {
10 |     _ids = ids;
11 |     _child = child;
12 | }
13 | }
```

## InsertPerformanceBenchmark.java: ItemWithStringBuffer

```

01 public static class ItemWithStringBuffer {
02 | }
```

```
03 |         public StringBuffer _name;
04 |         public ItemWithStringBuffer _child;
05 |
06 |         public ItemWithStringBuffer() {
07 |     }
08 |
09 |         public ItemWithStringBuffer(StringBuffer name, ItemWithStringBuffer child) {
10 |             _name = name;
11 |             _child = child;
12 |         }
13 |     }
```

The following results were achieved for the [testing configuration](#):

Java:

Storing 10000 objects with 3 levels of embedded objects:

- primitive object with int field

Store 30000 objects: 820ms

- object with String field

Store 30000 objects: 803ms

- object with StringBuffer field

Store 30000 objects: 2182ms

- object with int array field

Store 30000 objects: 810ms

- object with ArrayList field

Store 30000 objects: 2178ms

This revision (1) was last Modified 2007-12-30T14:35:54 by Tetyana.

# Indexes

One more feature that inevitably decreases the insert performance: indexes. When a new object with indexed field is inserted an index should be created and written to the database, which consumes additional resources. Luckily indexes do not only reduce the performance, actually they will improve the performance to a much more valuable degree during querying.

An example below provides a simple comparison of storing objects with and without indexes:

InsertPerformanceBenchmark.java: runIndexTest

```

01 ①②private void runIndexTest() {
02 |
03 |     init();
04 |     System.out.println("Storing " + _count + " objects with " + _depth + " levels of
embedded objects:");
05 |
06 |     clean();
07 |     configure();
08 |     System.out.println(" - no index");
09 |     open();
10 |     store();
11 |     close();
12 |
13 |     configureIndex();
14 |     System.out.println(" - index on String field");
15 |     open();
16 |     store();
17 |     close();
18 } }
```

InsertPerformanceBenchmark.java: configure

```

1 ①②private void configure() {
2     Configuration config = Db4o.configure();
3     config.lockDatabaseFile(false);
4     config.weakReferences(false);
5     config.io(new MemoryIoAdapter());
6     config.flushFileBuffers(false);
7 }
```

## InsertPerformanceBenchmark.java: configureIndex

```

1 ①②private void configureIndex() {
2     Configuration config = Db4o.configure();
3     config.lockDatabaseFile(false);
4     config.weakReferences(false);
5     config.io(new MemoryIoAdapter());
6     config.flushFileBuffers(false);
7     config.objectClass(Item.class).objectField("_name").indexed(true);
8 }

```

## InsertPerformanceBenchmark.java: init

```

1 ①②private void init() {
2     _count = 10000;
3     _depth = 3;
4     _isClientServer = false;
5
6 }

```

## InsertPerformanceBenchmark.java: store

```

01 ①②private void store() {
02     startTimer();
03     for (int i = 0; i < _count; i++) {
04         Item item = new Item("load", null);
05         for (int j = 1; j < _depth; j++) {
06             item = new Item("load", item);
07         }
08         objectContainer.set(item);
09     }
10     objectContainer.commit();
11     stopTimer("Store " + totalObjects() + " objects");
12 }

```

The following results were achieved for the [testing configuration](#):

Java:

Storing 10000 objects with 3 levels of embedded objects:

## Indexes

- no index

Store 30000 objects: 877ms

- index on String field

Store 30000 objects: 1076ms

This revision (1) was last Modified 2007-12-30T14:37:01 by Tetyana.

# Inherited Objects

Inherited objects are stored slower than simple objects. That is happening, because parent class indexes are created and stored to the database as well.

The following example shows the influence of a simple inheritance on the insert performance:

InsertPerformanceBenchmark.java: runInheritanceTest

```

01 private void runInheritanceTest() {
02 |
03 |     configure();
04 |     init();
05 |     clean();
06 |     System.out.println("Storing " + _count + " objects of depth " + _depth);
07 |     open();
08 |     store();
09 |     close();
10 |
11 |     clean();
12 |     System.out.println("Storing " + _count + " inherited objects of depth " + _depth);
13 |     open();
14 |     storeInherited();
15 |     close();
16 |
17 }
```

InsertPerformanceBenchmark.java: configure

```

1 private void configure() {
2     Configuration config = Db4o.configure();
3     config.lockDatabaseFile(false);
4     config.weakReferences(false);
5     config.io(new MemoryIoAdapter());
6     config.flushFileBuffers(false);
7 }
```

InsertPerformanceBenchmark.java: init

```

1 private void init()
```

## Inherited Objects

```
2 |     _count = 10000;
3 |     _depth = 3;
4 |     _isClientServer = false;
5 |
6 | }
```

## InsertPerformanceBenchmark.java: store

```
01|private void store() {
02 |     startTimer();
03 |     for (int i = 0; i < _count ;i++) {
04 |         Item item = new Item("load", null);
05 |         for (int j = 1; j < _depth; j++) {
06 |             item = new Item("load", item);
07 |         }
08 |         objectContainer.set(item);
09 |     }
10 |     objectContainer.commit();
11 |     stopTimer("Store "+ totalObjects() + " objects");
12 | }
```

## InsertPerformanceBenchmark.java: storeInherited

```
01|private void storeInherited() {
02 |     startTimer();
03 |     for (int i = 0; i < _count ;i++) {
04 |         ItemDerived item = new ItemDerived("load", null);
05 |         for (int j = 1; j < _depth; j++) {
06 |             item = new ItemDerived("load", item);
07 |         }
08 |         objectContainer.set(item);
09 |     }
10 |     objectContainer.commit();
11 |     stopTimer("Store "+ totalObjects() + " objects");
12 | }
```

## InsertPerformanceBenchmark.java: ItemDerived

```
1|public static class ItemDerived extends Item {
2 | }
```

```
3 public ItemDerived(String name, ItemDerived child) {  
4     super(name, child);  
5 }  
6 }
```

The following results were achieved for the [testing configuration](#):

Java:

Storing 10000 objects of depth 3

Store 30000 objects: 883ms

Storing 10000 inherited objects of depth 3

Store 30000 objects: 938ms

This revision (1) was last Modified 2007-12-30T14:37:52 by Tetyana.

# Configuration Options

Configuration options can also affect the insert performance. Some of them we've already came across in the previous topics:

[MemoryIoAdapter](#) - improves the insert performance, by replacing disk access with memory access:

Java:

```
configuration.io(new MemoryIoAdapter());
```

[lockDatabaseFile](#) - reduces the resources consumption by removing database lock thread. Should only be used for JVM versions < 1.4

Java:

```
configuration.lockDatabaseFile(false);
```

[weakReferences](#) - switching weak references off during insert operation releases extra resources and removed the cleanup thread.

Java:

```
configuration.weakReferences(false);
```

[FlushFileBuffers](#) - switching off flushFileBuffers can improve commit performance as the commit information will be cached by the operating system. However this setting is potentially dangerous and can lead to database corruption.

Java:

```
configuration.flushFileBuffers(false);
```

This revision (2) was last Modified 2007-12-30T14:40:27 by Tetyana.

# IO Benchmark Tools

This topic applies to Java version only

I/O access times play a [crucial role](#) in the overall performance of a database. To make their measurements easy and user-friendly we introduce two new tools to

1. measure the actual I/O performance of a system as seen by db4o
2. simulate the behaviour of a slower system on a faster one

All the code presented in this article can be found in the db4otools project / com.db4o.bench package.

SVN: <https://source.db4o.com/db4o/trunk/db4otools/>

Of course, you will also get it in your distribution. The code can be compiled with JDK 1.3 and higher.

More Reading:

- [First Steps](#)
- [Using Your Application To Generate The IO Pattern](#)
- [Simulating Slow IO On A Fast Machine](#)
- [IO Log File Statistics](#)

This revision (2) was last Modified 2008-02-03T14:00:52 by Tetyana.

# First Steps

This topic applies to Java version only

The main class of the benchmark is com.db4o.bench.loBenchmark.

Let's have a look at its run method to see what it does.

```
private void run(IoBenchmarkArgumentParser argumentParser) throws IOException
{
    runTargetApplication(argumentParser.objectCount());
    prepareDbFile(argumentParser.objectCount());
    runBenchmark(argumentParser.objectCount());
}
```

As you can see from this code, the benchmark consists of 3 stages:

1. Run a target application and log its I/O access pattern
2. Replay the recorded I/O operations once to prepare a database file.  
This step is necessary to ensure that during the grouped replay in the next step, none of the accesses will go beyond the currently existing file.
3. Replay the recorded I/O operations a second time. Operations are grouped by command type (read, write, seek, sync), and the total time executing all operations of a specific command type is measured. Grouping is necessary to avoid micro-benchmarking effects and to get time values above timer resolution.

We divide the numbers collected in stage 3 by the respective number of operations and we calculate the average time a particular command takes on the given system.

But enough of the theory for the moment, let's see how you can run the benchmark.

For this purpose there is the pair of an Ant script and a properties file:

- IoBenchmark.xml: The Ant script
- IoBenchmark.properties: Holding configurations for the Ant script

Both files are located in the root of db4otools.

To be able to run the benchmark from the Ant script, you have to put a db4o JAR file in the lib folder of the db4otools project. Insert the name of the JAR in the db4o.jar property in the property file, e.g.

```
db4o.jar=db4o-7.1.27.9109-jar5.jar
```

and you are ready to go!

To give it a first try, you can run the run.benchmark.small target of the Ant script, which is also the default target.

You should get output similar to this:

```
=====
Running db4o IoBenchmark
=====
Running target application ...
Preparing DB file ...
Running benchmark ...
-----
db4o IoBenchmark results with 1000 items
Statistics written to db4o-IoBenchmark-results-1000.log
-----
Results for READ
> operations executed: 14331
> time elapsed: 16 ms
> operations per millisecond: 895.6875
> average duration per operation: 0.001116460819203126 ms
READ 1116 ns

Results for WRITE
> operations executed: 9508
> time elapsed: 16 ms
> operations per millisecond: 594.25
> average duration per operation: 0.0016827934371055953 ms
WRITE 1682 ns

Results for SYNC
> operations executed: 7821
> time elapsed: 921 ms
> operations per millisecond: 8.49185667752443
> average duration per operation: 0.11775987725354814 ms
SYNC 117759 ns

Results for SEEK
> operations executed: 23839
> time elapsed: 16 ms
> operations per millisecond: 1489.9375
> average duration per operation: 6.711690926632829E-4 ms
SEEK 671 ns
```

As the output indicates, the results of this benchmark run will also be written to a file called db4o-IoBenchmark-results-1000.log. You can find this file in the db4otools directory.

The ns (nanosecond) values are our benchmark standard for the respective operation. Smaller numbers are better.

Note: It may be possible, that you get some zero values for time elapsed, and therefore infinity for operations per ms. This can occur if your machine is fast enough to execute all operations under 1ms. To overcome this you can run the run.benchmark.medium target which operates with more objects and takes

## First Steps

longer to complete.

This revision (2) was last Modified 2008-02-03T14:07:27 by Tetyana.

# Using Your Application To Generate The IO Pattern

This topic applies to Java version only

When you execute IoBenchmark, it uses a simple CRUD (create, read, update, delete) application as the target application. This application is located in the com.db4o.bench.crud package.

If you want to use your own application for generating the I/O access patterns, here's what you have to do:

- Use a LoggingIoAdapter, delegating to your default IoAdapter:

```
RandomAccessFileAdapter rafAdapter = new RandomAccessFileAdapter();
IoAdapter ioAdapter = new LoggingIoAdapter(rafAdapter, "filename.log");
Configuration config = Db4o.cloneConfiguration();
config.io(ioAdapter);
```

You'll also find this code in `com.db4o.bench.crud.CrudApplication#prepare()`.

- Change IoBenchmark to call your application by modifying the `runTargetApplication()` method.

You also have to exchange the calls to `CrudApplication.logFileName(itemCount)` in `prepareDbFile()` and `runBenchmark` with the file name of the log containing the I/O access pattern of your application. Using the code from above, this log file will be called "filename.log".

If you want to generate your log by interacting with your application, rather than having IoBenchmark calling it, do as follows:

- Use a LoggingIoAdapter in your application
- Interact with your application to create the log
- Remove the stage 1 from IoBenchmark and make it start in stage 2 with your log.

If you are using your own application to generate the I/O log file, check out the [IO Log File Statistics](#) section further down.

This revision (1) was last Modified 2008-02-03T14:12:38 by Tetyana.

# Simulating Slow IO On A Fast Machine

This topic applies to Java version only

The code for this section is located in the com.db4o.bench.delaying package.  
 To run delaying, the `System.nanoTime()` is needed. This method was introduced with Java 5.  
 If you only have older versions installed, get the latest here: <http://java.sun.com/javase/downloads/>  
 You also need a java5 db4o JAR file, otherwise you'll see a `NotImplementedException` when the benchmark tries to access `nanoTime()`.

Think of the following scenario: You develop software with db4o for a target system, that has much slower I/O than your developer system (e.g. an embedded device).

Wouldn't it sometimes be nice getting a feel for the expected speed your application will work with on the target system without having to deploy to it?

In particular, if you want to profile your system with a profiler like [JProbe](#), simulating the expected slow I/O on a device will help you identifying the bottlenecks in your application.

This is where the results of IoBenchmark and a DelayingIoAdapter enter the arena. If you run IoBenchmark on both the embedded device and your developer machine you get two results files. Copy the file from the slower device to the db4otools folder on the faster machine and set both filenames in IoBenchmark.properties:

```
results.file.1=db4o-IoBenchmark-results-30000_faster.log
results.file.2=db4o-IoBenchmark-results-30000_slower.log
```

It's not necessary that `results.file.1` holds the faster log, any order will work.

You are now set to run the benchmark in delayed mode. The expected result of such a run is, that the results of a delayed run on the faster machine should be close to those on the slow device.

To do a delayed run execute one of the `run.delayed.benchmark.*` targets of the Ant script.

At the beginning of the output - prior to the benchmark results - you'll notice additional information about the delaying:

```
=====
Running db4o IoBenchmark
=====
Delaying:
> machine1 (db4o-IoBenchmark-results-30000_faster.log) is faster!
> Required delays:
> [delays in nanoseconds] read: 8195 | write: 10669 | seek: 10098 | sync: 215121
> Adjusting delay timer to match required delays...
> Adjusted delays:
> [delays in nanoseconds] read: 4934 | write: 7387 | seek: 6849 | sync: 202203
Running target application ...
Preparing DB file ...
Running benchmark ...
[...]
```

Let's have a look at what exactly is going on when setting up delaying.

First there is a check for the validity of the two result files for delaying. To pass this check, one of the two supplied benchmark results file must contain the better values for all the 4 operations. This constraint exists because it's not possible to speed things up, only slowing them down.

Once this check is passed, the delays are calculated by simply subtracting the numbers found in the result files. The resulting numbers tell us, how long each I/O operation should be delayed on the faster machine to get the same behaviour as on the slower one.

The problem is now that just simply waiting for the calculated amount of time will make us wait for too long. This is due to additional setup time for each wait (method calls) and the "at least" semantics of the wait method itself.

To cope with this limitation there is a delay adjustment logic. It tries to find the actual delay to wait for such that the overall waiting time, including the setup method calls, matches the desired delay time.

However, there's a catch to this adjustment logic: On each machine there's a minimum delay that can be achieved with waiting, and this delay is not equal to zero (e.g. 400ns)! If the performance of the two machines is too close together, it is possible that when trying to adjust a delay, the outcome is below the minimum delay achievable.

In this case you'll see output like this:

```
>> Smallest achievable delay: 400
>> Required delay setting: 260
>> Using delay(0) to wait as short as possible.
>> Results will not be accurate.
```

To find out which delay actually was too small, and hence which results won't be accurate, take a look at the adjusted delays:

```
> Adjusted delays:
> [delays in nanoseconds] read: 0 | write: 7387 | seek: 6849 | sync: 202203
```

Here the read delay was too small and therefore the results for read are expected to be slower than targeted.

Once the delays are adjusted, they can be fed to the DelayingIoAdapter (as done in IoBenchmark#delayingIoAdapter):

```
IoAdapter rafFactory = new RandomAccessFileAdapter();
IoAdapter delFactory = new DelayingIoAdapter(rafFactory, _delays);
IoAdapter io = delFactory.open(dbFileName, false, 0, false);
```

If you now configure db4o with the IoAdapter io from above, each I/O operation will be delayed by the respective delay stored in \_delays!

The above IoAdapter setup is also exactly what you need in your own application to simulate the slower I/O of your target device on your faster machine.

This revision (1) was last Modified 2008-02-03T14:23:07 by Tetyana.

# IO Log File Statistics

This topic applies to Java version only

To get statistically meaningful results from the benchmark it is necessary that the I/O log file contains enough operations of each type. To get an overview on how well your I/O log file represents each operation, you can use the class LogStatistics in com.db4o.bench.logging.statistics.

Given the file name of an I/O log file LogStatistics will produce an HTML file that contains a table with statistics:

	<b>Count</b>	<b>%</b>	<b>Bytes</b>	<b>%</b>
Reads	664'631	30.64	52'214'278	70.83
Writes	360'001	16.6	21'508'576	29.17
Seeks	1'024'632	47.23		
Syncs	120'005	5.53		
Total	2'169'269		73'722'854	

Average byte count per read: 78

Average byte count per write: 59

This is the output of LogStatistics when run with the file generated by CrudApplication with 30k objects, which is the default setting for the Ant target run.benchmark.medium.

Typically sync operations are much rarer than seek operations. If you look at the source of CrudApplication you'll see that extra commit calls when deleting objects. These were inserted to get a higher sync count in the I/O log. It's possible that you also have to "tune" your application in a similar way to get good statistics.

This revision (1) was last Modified 2008-02-03T14:25:14 by Tetyana.

# Configuration

## Contents

- [Working With Configuration](#)
  - [Global Configuration](#)
  - [Object Container Configuration](#)
- [Further Reading](#)

db4o provides a wide range of configuration methods to request special behaviour.

For a complete list of all available methods see [Configuration](#), [ClientServerConfiguration](#), [ObjectClass](#) and [ObjectField](#) interfaces, and API documentation for the com.db4o.config package/namespace.

The following paragraphs contain some useful hints around using configuration calls.

## Working With Configuration

---

Configuration can be obtained using the following methods:

Java:

```
Configuration configuration = Db4o.newConfiguration();
```

To apply the configuration to an ObjectContainer/Client/Server use one of the following methods:

Java:

```
Db4o.openFile(configuration, databaseFileName)
```

```
Db4o.openServer(configuration, databaseFileName, port)
```

```
Db4o.openClient(configuration, hostName, port, user, password)
```

Configuration settings are **not** stored in db4o database files. Accordingly you will need to pass the same configuration object **every time** you open an ObjectContainer/Client/Server.

For using db4o in client/server mode it is recommended to use the same configuration on the server and on the client. To set this up nicely it makes sense to create one application class with one method that returns the required configuration and to deploy this class both to the server and to all clients.

Client/Server specific configuration calls reside in ClientServerConfiguration interface (.NET conventional name IClientServerConfiguration). These settings are accessed with:

Java:

```
ClientServerConfiguration configuration = Db4o.newConfiguration().clientServer();
```

For db4o versions before 6.0 the above-mentioned method of working with the configuration was not available. Instead, you could use:

- global db4o configuration context;
- object container instance configuration.

These methods are still available, however their usage is not advised due to their limitations.

## Global Configuration

---

Global configuration can be set using:

Java:

```
Db4o.configure()
```

Starting from version 6.0 you can obtain a clone of the global configuration:

Java:

```
Configuration configuration = Db4o.cloneConfiguration()
```

When an ObjectContainer/ObjectServer is opened, the global configuration context is cloned and copied into the newly opened ObjectContainer/ObjectServer. Subsequent calls against the global context with Db4o.configure() have no effect on open ObjectContainers/ObjectServers.

Global configuration has a number of disadvantages:

- the settings apply to all the object containers in the current VM/runtime
- the settings can't be reset.

## Object Container Configuration

---

Object Container instance configuration can be used for an open ObjectContainer/Client/Server:

Java:

```
objectContainer.ext().configure()  
objectServer.ext().configure()
```

Closing the container will erase all the settings.

The obvious disadvantages are:

- configuration can not be created before the container is opened;
- some configuration methods effect the way the system works on opening an object container therefore have no influence on the open object container;
- some configuration methods only effect the creation of a database;
- configuration settings will need to be repeated as many times as the new container gets opened.

## Further Reading

---

Some configuration switches are discussed in more detail in the following chapters:

- [Performance Hints](#)
- [Indexing](#)
- [Encryption](#)

This revision (25) was last Modified 2007-09-23T16:40:28 by Eric Falsken.

# General Configuration

Db4o configuration resides in com.db4o.config package/Db4object.Db4o.Config namespace.

Below is a short description of all general configuration methods. For client/server specific settings see [ClientServer Configuration](#) interface. For class and field specific configuration see [ObjectClass Configuration](#) and [ObjectField Configuration](#) interface.

Java:

```
public void activationDepth(int depth);
```

sets the activation depth to the specified value.

Java:

```
public void classActivationDepthConfigurable(boolean flag);
```

turns maximumActivationDepth setting of ObjectClass/IObjectClass configuration on and off.

This feature is turned on by default.

For more information on the activation depth see [Activation](#) topic.

Java:

```
public void addAlias(Alias alias);
```

```
public void removeAlias(Alias alias);
```

these methods, as the names suggest, add or remove aliases. Aliases should be configured before opening a database file or connecting to a server. For more information on aliases please check [Aliases](#).

Java:

```
public void allowVersionUpdates(boolean flag);
```

Turns automatic database file format version updates on. For more information see [Updating Db4o File](#)

## Format.

Java:

```
public void automaticShutDown(boolean flag);
```

turns automatic shutdown of the engine on and off.

Depending on the JDK, db4o uses one of the following two methods to shut down, if no more references to the ObjectContainer are being held or the JVM terminates:

- JDK 1.3 and above:

```
Runtime.addShutdownHook( )
```

- JDK 1.2 and below:

```
System.runFinalizersOnExit(true) and code in the finalizer
```

Some JVMs have severe problems with both methods. For these rare cases the autoShutDown feature may be turned off.

The default and recommended setting is true.

Java:

```
public void blockSize(int bytes);
```

sets the storage data blocksize for new ObjectContainers. For more information read [Increasing The Maximum Database File Size](#).

Java:

```
public void bTreeNodeSize(int size);
```

configures the size of BTree nodes in indexes.

Java:

```
public void bTreeCacheHeight(int height);
```

configures caching of BTree nodes.

For more information see [B-Tree tuning](#).

Java:

```
public void callbacks(boolean flag);
```

turns [object callbacks](#) on and off. [No callbacks](#) article explains how this setting can be useful for tuning.

Java:

```
public void callConstructors(boolean flag);
```

advises db4o to try instantiating objects with/without calling constructors.

Not all JDKs / .NET-environments support this feature. db4o will attempt, to follow the setting as good as the environment supports. In doing so, it may call implementation-specific features like

`sun.reflect.ReflectionFactory#newConstructorForSerialization` on the Sun Java 1.4.x/5 VM (not available on other VMs) and `FormatterServices.GetUninitializedObject()` on the .NET framework (not available on CompactFramework).

This setting may also be overridden for individual classes in `ObjectClass#callConstructor(boolean)`.

The default setting depends on the features supported by your current environment.

For more information see [Object Construction](#).

Java:

```
public void detectSchemaChanges(boolean flag);
```

tuning feature: configures whether db4o checks all persistent classes upon system startup for added or removed fields. For more information see [No schema changes](#).

Java:

```
public DiagnosticConfiguration diagnostic();
```

returns the configuration interface for diagnostics. See [Diagnostics](#) for more information.

Java:

```
public void disableCommitRecovery();
```

turns commit recovery off.

db4o uses a two-phase commit algorithm. In a first step all intended changes are written to a free place in the database file, the "transaction commit record". In a second step the actual changes are performed. If the system breaks down during commit, the commit process is restarted when the database file is opened the next time. On very rare occasions (possibilities: hardware failure or editing the database file with an external tool) the transaction commit record may be broken. In this case, this method can be used to try to open the database file without commit recovery. The method should only be used in emergency situations after consulting db4o support.

Java:

```
public void flushFileBuffers(boolean flag);
```

configures file buffers to be flushed during transaction commits. For more information see [FlushFileBuffers](#).

Java:

```
public void discardFreeSpace(int byteCount);
```

tuning feature: configures the minimum size of free space slots in the database file that are to be reused. For more information see [Discarding Free Space](#).

Java:

```
public void encrypt(boolean flag);
```

configures the use of [encryption](#). This method is deprecated, please use:

Java:

```
Db4o.configure().io(new XTeaEncryptionFileAdapter(password))
```

or any of the other XTeaEncryptionFileAdapter constructors.

Java:

```
public void exceptionsOnNotStorable(boolean flag);
```

configures whether Exceptions are to be thrown, if objects can not be stored. For more information see [ExceptionsOnNotStorable](#). The default for this setting is false.

Java:

```
public FreespaceConfiguration freespace();
```

returns the [freespace](#) configuration interface.

Java:

```
public void generateUUIDs(int setting);
```

configures db4o to generate UUIDs for stored objects.

setting parameter can have one of the following values:

- 1 - off
- 1 - configure classes individually
- Integer.MAX\_VALUE/Int32.MaxValue - on for all classes

This method is deprecated. Use the next method instead.

Java:

```
public void generateUUIDs(ConfigScope)
```

configures db4o to generate UUIDs for stored objects. For more information see [Unique Universal IDs](#).

Java:

```
public void generateVersionNumbers(int setting);
```

configures db4o to generate version numbers for stored objects. This method is deprecated. Use the next method instead.

Java:

```
public void generateVersionNumbers(ConfigScope setting);
```

configures db4o to generate version numbers for stored objects. Version numbers are used for [db4o Replication System \(dRS\)](#)

Java:

```
public void internStrings(boolean doIntern);
```

Configures db4o to call intern() on strings upon retrieval. For more information see Java/.NET documentation for string class.

Java:

```
public void io(IoAdapter adapter);
```

allows to configure db4o to use a customized byte IO adapter. [IO Adapter](#) topic discusses how to use a pluggable IO-adapter and how to create your own IO-adapter.

Java:

```
public void messageLevel(int level);
```

Sets the detail level of db4o messages (from 1 to 3). For more information see [Debugging db4o](#)

Java:

```
public void lockDatabaseFile(boolean flag);
```

can be used to turn the database file locking thread off. This tuning setting makes sense with java version 1.4 and higher. For more information see [No lock file thread](#).

Java:

```
public ObjectClass objectClass(Object clazz);
```

returns an ObjectClass/IObjectClass to configure the specified class.

The clazz parameter can be any of the following:

- a fully qualified classname as a string.
- a Class/Type object.
- any other object to be used as a template.

Java:

```
public void optimizeNativeQueries(boolean optimizeNQ);
```

If set to true, db4o will try to optimize native queries dynamically at query execution time, otherwise it will run native queries in unoptimized mode as SODA evaluations. For more information see [Native Query Optimization](#).

Java:

```
public boolean optimizeNativeQueries();
```

returns true if native queries will be optimized dynamically. For more information see [Native Query Optimization](#).

Java:

```
public void password(String pass);
```

protects the database file with a password when [simple encryption](#) is used. This method is deprecated . Please use:

Java:

```
Db4o.configure().io(new XTeaEncryptionFileAdapter(password))
```

or any of the other XTeaEncryptionFileAdapter constructors.

Java:

```
public QueryConfiguration queries();
```

returns the Query configuration interface. For more information see [Query Modes](#).

Java:

```
public void readOnly(boolean flag);
```

turns readOnly mode on and off.

Readonly mode allows opening an unlimited number of reading processes on one database file. It can be convenient when deploying db4o database files on CD-ROM.

Java:

```
public void reflectWith(Reflector reflector);
```

Configures the use of a specially designed reflection implementation.

On platforms that do not support reflection, customized reflection implementations may be written to enable db4o. For more information see [Db4o Reflection API](#).

Java:

```
public void refreshClasses();
```

forces analysis of all Classes during a running session.

This method may be useful in combination with a modified ClassLoader and allows exchanging classes during a running db4o session.

Calling this method on the global Configuration context will refresh the classes in all db4o sessions in the running VM. Calling this method in an ObjectContainer Configuration context, only the classes of the respective ObjectContainer will be refreshed.

Java:

```
public void reserveStorageSpace(long byteCount);
```

tuning feature only: reserves a number of bytes on creation of a database file.

An allocation of a fixed number of bytes at one time makes it more likely that the database will be stored in one chunk on the mass storage. Less read/write head movement can result in improved performance

Note: Allocated space will be lost on abnormal termination of the database engine (hardware crash, VM crash). A Defragment run will recover the lost space. For the best possible performance, this method should be called before the Defragment run to configure the allocation of storage space to be slightly greater than the anticipated database file size.

The default configuration: 0.

Java:

```
public void setBlobPath(String path) throws IOException;
```

configures the path to be used to store and read Blob data. For more information see [Blobs](#)

Java:

```
public void setClassLoader(Object classLoader);
```

configures db4o to use a custom ClassLoader. This method is deprecated, use instead:

Java:

```
reflectWith(new JdkReflector(classLoader))
```

Java:

```
public void setOut(PrintStream outStream);
```

Assigns a stream to print db4o event messages.

For more information see [Customizing The Debug Message Output](#)

Java:

```
public void testConstructors(boolean flag);
```

[/filter]

tuning feature: configures whether db4o should try to instantiate one instance of each persistent class on system startup. For more information see [No test instances](#)

[filter=java]

Java:

```
public void unicode(boolean flag);
```

configures the storage format of Strings.

This method needs to be called before a database file is created. db4o database files keep their string format after creation.

Turning Unicode support off reduces the file storage space for strings by factor 2 and improves performance.

The default setting: true

Java:

```
public void updateDepth(int depth);
```

specifies the global updateDepth. For more information see [Update Depth](#)

Java:

```
public void weakReferences(boolean flag);
```

turns weak reference management on or off. For more information see [Turning Off Weak References](#).

Java:

```
public void weakReferenceCollectionInterval(int milliseconds);
```

configures the timer for WeakReference collection.

The default setting is 1000 milliseconds.

Configure this setting to zero to turn WeakReference collection off. For more information see [Weak References](#)

Java:

```
public ClientServerConfiguration clientServer();
```

returns an interface for client/server specific configuration.

This revision (13) was last Modified 2007-06-04T16:55:04 by Tetyana.

# ClientServer Configuration

Client/server configuration methods are defined in ClientServerConfiguration/IClientServerConfiguration interface. You can access them through:

Java:

```
public Configuration.ClientServerConfiguration clientServer();
```

Java:

```
void prefetchIDCount(int prefetchIDCount);
```

Sets the number of IDs to be prefetched from the server for new objects created on the client. For more information see [Prefetching IDs For New Objects](#)

Java:

```
void prefetchObjectCount(int prefetchObjectCount);
```

Sets the number of objects to be prefetched for an ObjectSet in client/server mode. For more information see [Prefetching Objects For Query Results](#)

Java:

```
public void setMessageRecipient(MessageRecipient messageRecipient);
```

sets the MessageRecipient to receive Client Server messages. For more information see [Messaging](#).

Java:

```
public MessageSender getMessageSender();
```

returns the MessageSender for this Configuration context. For more information see [Messaging](#).

Java:

```
public void timeoutClientSocket(int milliseconds);
```

configures the time a client waits for a message response from the server. The default value: 300000ms (5 minutes).

Java:

```
public void timeoutServerSocket(int milliseconds);
```

configures the timeout of the server side socket.

All server connection threads are checking if the server was shut down on a regular basis. Use this method to configure the interval for the checks.

The default value: 5000ms (5 seconds)

Java:

```
public void timeoutPingClients(int milliseconds);
```

.NET:

configures the delay time after which the server starts pinging connected clients to check the connection. If no client messages are received by the server for the configured interval, the server sends a "PING" message to the client and waits for an "OK" response. After 5 unsuccessful attempts, the client connection is closed.

This value may need to be increased for single-threaded clients, since they can't respond instantaneously.

The default timeout value is 180000ms (3 minutes).

Java:

```
public void singleThreadedClient(boolean flag);
```

.NET:

configures the client messaging system to be single threaded or multithreaded.

Recommended settings:

- true for low resource systems.
- false for best asynchronous performance and fast GUI response.

The default value is:

- for .NET Compactframework: true
- all other platforms: false

Java:

```
public void maxBatchQueueSize(int maxSize);
```

.NET:

configures to batch messages between client and server. By default, batch mode is disabled. For more information see [Batch Mode](#)

Java:

```
public void batchMessages(boolean flag);
```

configures the maximum memory buffer size for batched message. If the size of batched messages is greater than `maxSize`, batched messages will be sent to server. For more information see [Batch Mode](#)

This revision (13) was last Modified 2007-06-04T17:59:29 by Tetyana.

# ObjectClass Configuration

ObjectClass/IObjectClass provides an interface for class configuration.

Java:

```
ObjectClass oc = Db4o.configure().objectClass(clazz);
```

clazz parameter here can be one of the following:

- a fully qualified classname as a String.
- a Class object.
- any other object to be used as a template.

Java:

```
public void callConstructor(boolean flag);
```

advises db4o to try instantiating objects of this class with/without calling constructors. For more information see [Object Construction](#).

Java:

```
public void cascadeOnActivate(boolean flag);
```

sets cascaded activation behavior for this class. For more information see [Activation](#).

Java:

```
public void cascadeonDelete(boolean flag);
```

sets cascaded delete behavior for this class. For more information see [Delete Behavior](#).

Java:

```
public void cascadeOnUpdate(boolean flag);
```

sets cascaded update behavior for this class. For more information see [Update Depth](#).

Java:

```
public void compare(ObjectAttribute attributeProvider);
```

registers an attribute provider for special query behavior.

The query processor will compare the object returned by the attribute provider instead of the actual object, both for the constraint and the candidate persistent object. For more information see [Custom Query Comparator](#).

Java:

```
public void enableReplication(boolean setting);
```

enable replication of the specified class. Must be called before databases are created or opened so that db4o will control versions and generate UUIDs for objects of this class, which is required for using replication. For more information see [db4o Replication System \(dRS\)](#)

Java:

```
public void generateUUIDs(boolean setting);
```

generate UUIDs for stored objects of this class. For more information see [Unique Universal IDs](#).

Java:

```
public void generateVersionNumbers(boolean setting);
```

generate version numbers for stored objects of this class. For more information see [db4o Replication System \(dRS\)](#)

Java:

```
public void indexed(boolean flag);
```

turns the class index on or off.

db4o maintains an index for each class to be able to deliver all instances of a class in a query. If the class

index is never needed, it can be turned off with this method to improve the performance on creation and deletion of the class objects.

Common cases where a class index is not needed:

- The application always works with subclasses or superclasses.
- There are convenient field indexes that will always find instances of a class.
- The application always works with IDs.

For more information see [No Class Index](#).

Java:

```
public void maximumActivationDepth (int depth);
```

sets the maximum activation depth for this class to the desired value. For more information see [Activation](#).

Java:

```
public void minimumActivationDepth (int depth);
```

sets the minimum activation depth to the desired value. For more information see [Activation](#).

Java:

```
public ObjectField objectField (String fieldName);
```

returns an ObjectField object to configure the specified field. For more information see [ObjectField Configuration](#).

Java:

```
public void persistStaticFieldValues();
```

turns on storing static field values for this class. For more information see [Static Fields And Enums](#).

Java:

```
public void readAs(Object clazz);
```

creates a temporary mapping of a persistent class to a different class.

If the meta-information for this ObjectClass has been stored to the database file, it will be read from the database file as if it was representing the class specified by the clazz parameter passed to this method.

The clazz parameter can be any of the following:

- a fully qualified classname as a String.
- a Class object.
- any other object to be used as a template.

This method will be ignored if the database file already contains meta-information for clazz.

This method is deprecated, use [Aliases](#) instead.

Java:

```
public void rename (String newName);
```

renames a stored class. For more information see [Refactoring and Schema Evolution](#).

Java:

```
public void storeTransientFields (boolean flag);
```

allows to specify if transient fields are to be stored. The default for every class is false.

For more information see [Storing Transient Fields](#).

Java:

```
public void translate (ObjectTranslator translator);
```

registers a translator for this class. For more information see [Translators](#).

Java:

```
public void updateDepth (int depth);
```

specifies the updateDepth for this class. For more information see [Update Depth](#).

This revision (10) was last Modified 2008-04-14T16:32:18 by Tetyana.

# ObjectField Configuration

ObjectField/IObjectField is an interface providing configuration methods for class fields. ObjectField instance can be obtained through:

Java:

```
ObjectField of = Db4o.configure().objectClass(clazz).objectField("fieldName");
```

Java:

```
public void cascadeOnActivate(boolean flag);
```

sets cascaded activation behavior. For more information see [Activation](#).

Java:

```
public void cascadeonDelete(boolean flag);
```

sets cascaded delete behavior. For more information see [Delete Behavior](#).

Java:

```
public void cascadeOnUpdate(boolean flag);
```

sets cascaded update behavior. For more information see [Update Depth](#).

Java:

```
public void indexed(boolean flag);
```

turns indexing on or off. For more information see [Indexing](#).

Java:

```
public void rename(String newName);
```

renames a field of a stored class. For more information see [Refactoring and Schema Evolution](#).

Java:

```
public void queryEvaluation (boolean flag);
```

toggles query evaluation.

All fields are evaluated by default. Use this method to turn query evaluation off for specific fields.

This revision (5) was last Modified 2007-06-04T18:09:17 by Tetyana.

# Selective Persistence

Sometimes your persistent classes may have fields, which are useless or even undesirable to store. References to classes, which objects are constructed at runtime, can be an example.

How to avoid saving these fields to db4o?

More Reading:

- [Transient Fields In Java](#)
- [Transient Fields In .NET](#)
- [Transient Classes](#)
- [Storing Transient Fields](#)

This revision (2) was last Modified 2007-01-29T06:34:56 by Tetyana.

# Transient Fields In Java

This topic applies to Java version only

You can use the *transient* keyword to indicate that a field is not part of the persistent state of an object:

Test.java

```

01 /* Copyright (C) 2004 - 2007 db4objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.selectivepersistence;
04
05 public class Test {
06     transient String transientField;
07
08     String persistentField;
09
10     public Test(String transientField, String persistentField) {
11         this.transientField = transientField;
12         this.persistentField = persistentField;
13     }
14
15     public String toString() {
16         return "Test: persistent: " + persistentField
17             + ", transient: " + transientField;
18     }
19
20 }
```

The following example demonstrates the effect of transient keyword on db4o:

MarkTransientExample.java: saveObjects

```

01 private static void saveObjects(Configuration configuration) {
02     new File(DB4O_FILE_NAME).delete();
03     ObjectContainer container = Db4o.openFile(configuration, DB4O_FILE_NAME);
```

```
04 |     try
05 |     {
06 |         Test test = new Test("Transient string", "Persistent string");
07 |         container.set(test);
08 |     }
09 |     finally
10 |     {
11 |         container.close();
12 |     }
13 | }
```

## MarkTransientExample.java: retrieveObjects

```
01 private static void retrieveObjects()
02 {
03     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
04     try
05     {
06         ObjectSet result = container.query(Test.class);
07         listResult(result);
08     }
09     finally
10     {
11         container.close();
12     }
13 }
```

This revision (2) was last Modified 2007-01-29T06:32:48 by Tetyana.

# Transient Fields In .NET

This topic applies to .NET version only

There are different ways to prevent fields persistence in .NET:

- You can use the [com.db4o.Transient] or [NonSerialized] attribute to indicate that a field is not part of the persistent state of an object.
- You can use any built-in .NET attribute or define your own attribute class and mark it transient for db4o.

For example, let's create a FieldTransient attribute and mark it to prevent object persistence:

Let's use the newly-defined FieldTransient attribute and the system-provided Transient, and compare the results:

We will save and retrieve both Test and TestCustomized objects, having transient fields defined in different manner:

You will see the identical results independently of the way the transiency is defined.

This revision (2) was last Modified 2007-04-28T10:48:50 by Tetyana.

# Transient Classes

Some of the classes are not supposed to be persistent. Of course you can avoid saving their instances in your code and mark all their occurrences in another classes as transient ([Java/.NET](#)). But that needs some attention and additional coding. You can achieve the same result in an easier way using TransientClass interface:

Java:

```
com.db4o.types.TransientClass
```

TransientClass is a marker interface, which guarantees that the classes implementing it will never be added to the class metadata. In fact they are just skipped silently by db4o persistence mechanism.

An example of the TransientClass implementation is db4o object container (we do not need to save a database into itself).

Let's look how it works on an example. We will create a simplest class implementing TransientClass interface:

**NotStorable.java**

```
01 /* Copyright (C) 2004 - 2007 db4objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.selective.persistence;
04
05 import com.db4o.types.TransientClass;
06
07
08 public class NotStorable implements TransientClass {
09 |
10 | public String toString(){
11 | | return "NotStorable class";
12 | }
13 |}
```

NotStorable class will be used as a field in two test objects: [Test1](#) and [Test2](#).

In our example we will use the default configuration and save Test1 and Test2 objects just as usual:

## TransientClassExample.java: saveObjects

```

01 ①private static void saveObjects() {
02 |     new File(DB40_FILE_NAME).delete();
03 |     ObjectContainer container = Db4o.openFile(DB40_FILE_NAME);
04 ②③try {
05 |         // Save Test1 object with a NotStorable class field
06 |         Test1 test1 = new Test1("Test1", new NotStorable());
07 |         container.set(test1);
08 |         // Save Test2 object with a NotStorable class field
09 |         Test2 test2 = new Test2("Test2", new NotStorable(), test1);
10 |         container.set(test2);
11 ④⑤} finally {
12 |     container.close();
13 | }
14 }
```

Now let's try to retrieve the saved objects:

## TransientClassExample.java: retrieveObjects

```

01 ①private static void retrieveObjects() {
02 |     ObjectContainer container = Db4o.openFile(DB40_FILE_NAME);
03 ③④try {
04 |         // retrieve the results and check if the NotStorable
05 |         // instances were saved
06 |         ObjectSet result = container.get(null);
07 |         listResult(result);
08 ④⑤} finally {
09 |     container.close();
10 | }
11 }
```

If you will run the example code you will see that all the instances of NotStorable class are set to null.  
This revision (4) was last Modified 2007-01-29T16:14:01 by Tetyana.

# Test1

Code attachment not found: /Resources/Reference/Tuning>Selective\_Persistence/Transient\_Classes/SelPersistJava.Zip

This revision (4) was last Modified 2007-01-29T16:12:59 by Tetyana.

## Test2

Code attachment not found: /Resources/Reference/Tuning>Selective\_Persistence/Transient\_Classes/SelPersistJava.Zip

This revision (1) was last Modified 2007-01-29T16:12:21 by Tetyana.

# Storing Transient Fields

Java:

```
Db4o.configure().objectClass(clazz).storeTransientFields(true)
```

This setting can be used to turn on storing of transient fields to db4o. It can be sometimes useful for debug purposes.

In order to test how it works add the following method to the example in [Transient Fields In Java/Transient Fields In .NET](#):

MarkTransientExample.java: configureSaveTransient

```
1 private static Configuration configureSaveTransient() {
2 |   Configuration configuration = Db4o.newConfiguration();
3 |   configuration.objectClass(Test.class).storeTransientFields(true);
4 |   return configuration;
5 }
```

This revision (2) was last Modified 2007-03-21T19:12:05 by Tetyana.

# Indexing

db4o allows to index fields to provide maximum querying performance. To request an index to be created, you would issue the following API method call in your global [db4o configuration method](#) before you open an ObjectContainer/ObjectServer:

// assuming

```
class Foo{  
    String bar;  
}
```

Java:

```
configuration.objectClass(Foo.class).objectField("bar").indexed(true);
```

If the configuration is set in this way, an index on the Foo#bar field will be created (if not present already) the next time you open an ObjectContainer/ObjectServer and you use the Foo class the first time in your application.

Contrary to all other [configuration calls](#) indexes - once created - will remain in a database even if the index configuration call is not issued before opening an ObjectContainer/ObjectServer.

To drop an index you would also issue a configuration call in your db4o configuration method:

Java:

```
configuration.objectClass(Foo.class).objectField("bar").indexed(false);
```

[/filter]

[filter=net]

.NET:

```
configuration.ObjectClass(typeof(Foo)).ObjectField("bar").Indexed(false);
```

Actually dropping the index will take place the next time the respective class is used. db4o will tell you when it creates and drops indexes, if you choose a message level of 1 or higher:

Java:

```
configuration.messageLevel(1);
```

For creating and dropping indexes on large amounts of objects there are two possible strategies:

1. Import all objects with indexing off, configure the index and reopen the ObjectContainer/ObjectServer.
2. Import all objects with indexing turned on and commit regularly for a fixed amount of objects (~10,000).
  1. will be faster.
  2. will keep memory consumption lower.

For more information see [Enable Field Indexes](#) chapter.

This revision (13) was last Modified 2007-05-07T16:49:45 by Tetyana.

# Performance Hints

The following is an overview over possible tuning switches that can be set when working with db4o. Users that do not care about performance may like to read this chapter also because it provides a side glance at db4o features with *Alternate Strategies* and some insight on how db4o works.

More Reading:

- [Enable Field Indexes](#)
- [Discarding Free Space](#)
- [Calling constructors](#)
- [Defragment](#)
- [Turning Off Weak References](#)
- [No Shutdown Thread](#)
- [No callbacks](#)
- [No schema changes](#)
- [No lock file thread](#)
- [No test instances](#)
- [Increasing The Maximum Database File Size](#)
- [FlushFileBuffers](#)
- [B-Tree tuning](#)
- [Inheritance hierarchies](#)
- [Persistent and transient fields](#)
- [Activation strategies](#)
- [Automatic Shutdown](#)
- [Reserving Storage Space](#)
- [Unicode](#)
- [Prefetching IDs For New Objects](#)
- [Prefetching Objects For Query Results](#)
- [No Class Index](#)
- [No Field Evaluation](#)
- [RandomAccessFileAdapter](#)
- [Commit Strategies](#)
- [Optimizing Native Queries](#)

This revision (3) was last Modified 2006-11-15T11:56:34 by Tetyana.

# Enable Field Indexes

## Contents

- [Advantage](#)
- [Alternate Strategies](#)

For class Car with field "pilot":

Java:

```
Db4o.configure().objectClass(Car.class).objectField("pilot").indexed(true)
```

## Advantage

---

The fastest way to improve the performance of your queries is to enable indexing on some of your class's key fields. You can read how to do it in [Indexing](#) chapter of this documentation.

Further step of index tuning is to optimize indexes for Class.Field1.Field2 access. What will give us the best performance:

- index on Field1;
- index on Field2;
- index on both fields?

To find the answer let's consider classes Car and Pilot from the previous chapters. In order to see indexing influence we will put 10000 new cars in our storage (note that for db4o version > 5.6 the amount of objects should be much more to see the differences in execution time due to BTree based index optimized for big amounts of data):

IndexedExample.java: fillUpDB

```
01 | private static void fillUpDB() {
02 |     new File(DB4O_FILE_NAME).delete();
03 |     ObjectContainer container=Db4o.openFile(DB4O_FILE_NAME);
04 |     try {
05 |         for (int i=0; i<10000; i++) {
06 |             AddCar(container, i);
07 |         }
08 |     }
09 |     finally {
10 |         container.close();
11 |     }
```

12 }

## IndexedExample.java: addCar

```

1 private static void AddCar(ObjectContainer container, int points)
2 {
3     Car car = new Car("BMW");
4     car.setPilot(new Pilot("Tester", points));
5     container.set(car);
6 }

```

Now we have lots of similar cars differing only in the amount of pilots' points - that will be our constraint for the query.

## IndexedExample.java: noIndex

```

01 private static void noIndex() {
02     ObjectContainer container=Db4o.openFile(DB4O_FILE_NAME);
03     try {
04         Query query = container.query();
05         query.constrain(Car.class);
06         query.descend("pilot").descend("points").constrain(new Integer(99));
07
08         long t1 = System.currentTimeMillis();
09         ObjectSet result = query.execute();
10         long t2 = System.currentTimeMillis();
11         long diff = t2 - t1;
12         System.out.println("Test 1: no indexes");
13         System.out.println("Execution time="+diff + " ms");
14         listResult(result);
15     }
16     finally {
17         container.close();
18     }
19 }

```

You can check execution time on your workstation using interactive version of this tutorial.

Let's create index for pilots and their points and test the same query again:

## IndexedExample.java: fullIndex

```

01  private static void fullIndex() {
02 |     Configuration configuration = Db4o.newConfiguration();
03 |     configuration.objectClass(Car.class).objectField("pilot").indexed(true);
04 |     configuration.objectClass(Pilot.class).objectField("points").indexed(true);
05 |     ObjectContainer container=Db4o.openFile(configuration, DB4O_FILE_NAME);
06 |     try {
07 |         Query query = container.query();
08 |         query.constrain(Car.class);
09 |         query.descend("pilot").descend("points").constrain(new Integer(99));
10 |
11 |         long t1 = System.currentTimeMillis();
12 |         ObjectSet result = query.execute();
13 |         long t2 = System.currentTimeMillis();
14 |         long diff = t2 - t1;
15 |         System.out.println("Test 2: index on pilot and points");
16 |         System.out.println("Execution time=" + diff + " ms");
17 |         listResult(result);
18 |     }
19 |     finally {
20 |         container.close();
21 |     }
22 }

```

That result is considerably better and proves the power of indexing.

But do we really need 2 indexes? Will single pilot or points index suffice? Let's test this as well:

## IndexedExample.java: pilotIndex

```

01  private static void pilotIndex() {
02 |     Configuration configuration = Db4o.newConfiguration();
03 |     configuration.objectClass(Car.class).objectField("pilot").indexed(true);
04 |     configuration.objectClass(Pilot.class).objectField("points").indexed(false);
05 |     ObjectContainer container=Db4o.openFile(configuration, DB4O_FILE_NAME);
06 |     try {
07 |         Query query = container.query();

```

```

08 |         query.constrain(Car.class);
09 |         query.descend("pilot").descend("points").constrain(new Integer(99));
10 |
11 |         long t1 = System.currentTimeMillis();
12 |         ObjectSet result = query.execute();
13 |         long t2 = System.currentTimeMillis();
14 |         long diff = t2 - t1;
15 |         System.out.println("Test 3: index on pilot");
16 |         System.out.println("Execution time=" + diff + " ms");
17 |         listResult(result);
18 |
19 |     } finally {
20 |         container.close();
21 |     }
22 |

```

## IndexedExample.java: pointsIndex

```

01 | private static void pointsIndex() {
02 |     Configuration configuration = Db4o.newConfiguration();
03 |     configuration.objectClass(Car.class).objectField("pilot").indexed(false);
04 |     configuration.objectClass(Pilot.class).objectField("points").indexed(true);
05 |     ObjectContainer container=Db4o.openFile(configuration, DB4O_FILE_NAME);
06 |     try {
07 |         Query query = container.query();
08 |         query.constrain(Car.class);
09 |         query.descend("pilot").descend("points").constrain(new Integer(99));
10 |
11 |         long t1 = System.currentTimeMillis();
12 |         ObjectSet result = query.execute();
13 |         long t2 = System.currentTimeMillis();
14 |         long diff = t2 - t1;
15 |         System.out.println("Test 4: index on points");
16 |         System.out.println("Execution time=" + diff + " ms");
17 |         listResult(result);
18 |
19 |     } finally {

```

```
20 |         container.close();  
21 |     }  
22 L }
```

Single index does not increase query performance on second level fields.

To maximize retrieval performance on encapsulated fields of different levels of enclosure

Class.Field1.Field2.Field3(.FieldN)

indexes for each field level should be created:

Class.Field1.Indexed(true)

Field1Class.Field2.Indexed(true)

Field2Class.Field3.Indexed(true)

...

Field(N-1)Class.FieldN.Indexed(true)

## Alternate Strategies

---

Field indexes dramatically improve query performance but they may considerably reduce storage and update performance. The best way to decide where to put the indexes is to test them on completed application with typical typical data load.

This revision (12) was last Modified 2007-07-11T06:51:13 by Tetyana.

# Discarding Free Space

## Contents

- [Advantage](#)
- [Effect](#)
- [Alternate Strategies](#)

Java: `Db4o.configure().freespace().discardSmallerThan(byteCount)`

Configures the minimum size of free space slots in the database file that are to be reused.

2 extremes for byteCount value:

- `Integer.MAX_VALUE` - discard all free slots for the best possible startup time. The downside: database files will necessarily grow faster
- 0 - default setting, all freespace is reused. The downside: increased memory consumption and performance loss for maintenance of freespace lists in RAM

## Advantage

---

Allows fine-tuning of performance/size relation for your environment.

## Effect

---

When objects are updated or deleted, the space previously occupied in the database file is marked as "free", so it can be reused. db4o maintains two lists in RAM, sorted by address and by size. Adjacent entries are merged. After a large number of updates or deletes have been executed, the lists can become large, causing RAM consumption and performance loss for maintenance. With this method you can specify an upper bound for the byte slot size to discard.

## Alternate Strategies

---

Regular defragment will also keep the number of free space slots small. See:

Java: `com.db4o.defragment.Defragment`

If defragment can be frequently run, it will also reclaim lost space and decrease the database file to the minimum size. Therefore `#discardSmallerThan(maxValue)` may be a good tuning mechanism for setups

Discarding Free Space

with frequent defragment runs.

This revision (8) was last Modified 2006-11-23T19:10:43 by Tetyana.

# Calling constructors

## Contents

- [Advantage](#)
- [Effect](#)
- [Alternate Strategies](#)

Java:

```
Db4o.configure().callConstructors(true)
```

## Advantage

---

will configure db4o to use constructors to instantiate objects.

## Effect

---

On VMs where this is supported (Sun Java VM > 1.4, .NET, Mono) db4o tries to create instances of objects without calling a constructor. On Java VMs db4o is using reflection for this feature so this may be considerably slower than using a constructor. For the best performance on Java it is recommended to add a public zero-parameter constructor to every persistent class and to turn constructors on. Benchmarks on .NET have shown that the default setting ( #callConstructors(false) ) is faster.

## Alternate Strategies

---

Constructors can also be turned on for individual classes only with

```
Java: Db4o.configure().objectClass(Foo.class).callConstructor(true)
```

There are some classes (e.g. java.util.Calendar) that require a constructor to be called to work. Further details can be found in the [Constructors](#) chapter

This revision (13) was last Modified 2007-08-04T11:34:14 by Tetyana.

# Defragment

## Contents

- [Advantage](#)
- [Effect](#)
- [Alternate Strategies](#)

Java: `Defragment.defrag( "sample.db4o" )`

## Advantage

---

It is recommended to run Defragment frequently to reduce the database file size and to remove unused fields and freespace slots.

## Effect

---

db4o does not discard fields from the database file that are no longer being used. Within the database file quite a lot of space is used for transactional processing. Objects are always written to a new slot when they are modified. Deleted objects continue to occupy 8 bytes until the next Defragment run. Defragment cleans all this up by writing all objects to a completely new database file. The resulting file will be smaller and faster.

## Alternate Strategies

---

Instead of deleting objects it can be an option to mark objects as deleted with a "deleted" boolean field and to clean them out (by not copying them to the new database file) during the Defragment run. Two advantages:

1. Deleted objects can be restored.
2. In case there are multiple references to a deleted object, none of them would point to null.

This revision (6) was last Modified 2007-05-07T15:30:10 by Tetyana.

# Turning Off Weak References

## Contents

- [Advantage](#)
- [Effect](#)
- [Alternate Strategies](#)

Java:

```
Db4o.configure().weakReferences(false)
```

## Advantage

---

will configure db4o to use hard direct references instead of weak references to control instantiated and stored objects.

## Effect

---

A db4o database keeps a reference to all persistent objects that are currently held in RAM, whether they were stored to the database in this session or instantiated from the database in this session. This is how db4o can "know" than an object is to be updated: Any "known" object must be an update, any "unknown" object will be stored as "new". (Note that the reference system will only be in place as long as an ObjectContainer is open. Closing and reopening an ObjectContainer will clean the references system of the ObjectContainer and all objects in RAM will be treated as "new" afterwards.) In the default configuration db4o uses weak references and a dedicated thread to clean them up after objects have been garbage collected by the VM. Weak references need extra resources and the cleanup thread will have a considerable impact on performance since it has to be synchronized with the normal operations within the ObjectContainer. Turning off weak references will improve speed.

The downside: To prevent memory consumption from growing consistently, the application has to take care of removing unused objects from the db4o reference system by itself. This can be done by calling

Java:

```
ExtObjectContainer.purge(object)
```

## Alternate Strategies

---

Java:

`ExtObjectContainer.purge(object)`

can also be called in normal weak reference operation mode to remove an object from the reference cache. This will help to keep the reference tree as small as possible. After calling `#purge(object)` an object will be unknown to the ObjectContainer so this feature is also suitable for batch inserts.

This revision (9) was last Modified 2007-05-07T15:44:48 by Tetyana.

# No Shutdown Thread

## Contents

- [Advantage](#)
- [Effect](#)
- [Alternate Strategies](#)

Java:

```
Db4o.configure().automaticShutDown(false)
```

## Advantage

---

can prevent the creation of a shutdown thread on some platforms.

## Effect

---

On some platforms db4o uses a ShutDownHook to cleanly close all database files upon system termination. If a system is terminated without calling ObjectContainer#close() for all open ObjectContainers, these ObjectContainers will still be usable but they will not be able to write back their freespace management system back to the database file. Accordingly database files will be observed to grow.

## Alternate Strategies

---

Database files can be reduced to their minimal size with [Defragment](#)

This revision (8) was last Modified 2007-05-07T15:46:34 by Tetyana.

# No callbacks

## Contents

- [Advantage](#)
- [Effect](#)
- [Alternate Strategies](#)

Java:

```
Db4o.configure().callbacks(false)
```

## Advantage

---

will prevent db4o from looking for callback methods in all persistent classes on system startup.

## Effect

---

Upon system startup, db4o will scan all persistent classes for methods with the same signature as the methods defined in com.db4o.ext.ObjectCallbacks, even if the interface is not implemented. db4o uses reflection to do so and on constrained environments this can consume quite a bit of time. If callback methods are not used by the application, callbacks can be turned off safely.

## Alternate Strategies

---

Class configuration features are a good alternative to callbacks. The most recommended mechanism to cascade updates is:

Java:

```
Db4o.configure().objectClass("yourPackage.yourClass").cascadeOnUpdate(true)
```

This revision (4) was last Modified 2007-05-07T15:47:38 by Tetyana.

# No schema changes

## Contents

- [Advantage](#)
- [Effect](#)
- [Alternate Strategies](#)

Java:

```
Db4o.configure().detectSchemaChanges(false)
```

## Advantage

---

will prevent db4o from analysing the class structure upon opening a database file.

## Effect

---

Upon system startup, db4o will use reflection to scan the structure of all persistent classes. This process can take some time, if a large number of classes are present in the database file. For the best possible startup performance on "warm" database files (all classes already analyzed in a previous startup), this feature can be turned off.

## Alternate Strategies

---

Instead of using one database file to store a huge and complex class structure, a system may be more flexible and faster, if multiple database files are used. In embedded client/server setup, database files can also be switched from the client side with

Java:

```
((ExtClient) objectContainer).switchToFile(databaseFile)
```

This revision (9) was last Modified 2007-09-01T14:25:44 by Tetyana.

# No lock file thread

## Contents

- [Advantage](#)
- [Effect](#)
- [Alternate Strategies](#)

Java:

```
configuration.lockDatabaseFile(false)
```

## Advantage

---

will prevent the creation of a lock file thread on Java platforms without NIO (< JDK 1.4.1).

## Effect

---

If file locking is not available on the system, db4o will regularly write a timestamp lock information to the database file, to prevent other VM sessions from accessing the database file at the same time. Uncontrolled concurrent access would inevitably lead to corruption of the database file. If the application ensures that it can not be started multiple times against the database file, db4o file locking may not be necessary.

## Alternate Strategies

---

Database files can safely be opened from multiple sessions in readonly mode. Use:

Java:

```
configuration.readOnly(true)
```

This revision (7) was last Modified 2007-05-07T15:51:32 by Tetyana.

# No test instances

## Contents

- [Advantage](#)
- [Effect](#)
- [Alternate Strategies](#)

Java:

```
configuration.testConstructors(false)
```

## Advantage

---

will prevent db4o from creating a test instance of persistent classes upon opening a database file.

## Effect

---

Upon system startup, db4o attempts to create a test instance of all persistent classes, to ensure that a public zero-parameter constructor is present. This process can take some time, if a large number of classes are present in the database file. For the best possible startup performance this feature can be turned off.

## Alternate Strategies

---

In any case it's always good practice to create a zero-parameter constructor. If this is not possible because a class from a third party is used, it may be a good idea to write a [translator](#) that translates the third party class to one's own class.

This revision (7) was last Modified 2007-05-07T16:17:31 by Tetyana.

# Increasing The Maximum Database File Size

## Contents

- [Advantage](#)
- [Effect](#)
- [Alternate Strategies](#)

Java:

```
configuration.blockSize(newBlockSize); Defragment.defrag( "sample.db4o" )
```

## Advantage

---

Increasing the block size from the default of 1 to a higher value permits you to store more data in a db4o database.

## Effect

---

By default db4o databases can have a maximum size of 2GB. By increasing the block size that db4o should internally use, the upper limit for database files sizes can be raised to multiples of 2GB. Any value between 1 byte (2GB) to 127 bytes (254GB) can be chosen as the block size.

Because of possible padding for objects that are not exact multiples in length of the block size, database files will naturally tend to be bigger if a higher value is chosen. Because of less file access cache hits a higher value will also have a negative effect on performance.

A very good choice for this value is 8 bytes, because that corresponds to the slot length of the pointers (address + length) that db4o internally uses.

## Alternate Strategies

---

It can also be very efficient to use multiple ObjectContainers instead of one big one. Objects can be freely moved, copied and [replicated](#) between Objectcontainers.

This revision (9) was last Modified 2007-05-07T16:20:04 by Tetyana.

# FlushFileBuffers

## Contents

- [Advantage](#)
- [Effect](#)
- [Alternate Strategies](#)

Java:

```
configuration.flushFileBuffers(false)
```

## Advantage

---

Setting FlushFileBuffers to false can considerably improve the performance saving time on physical disk access.

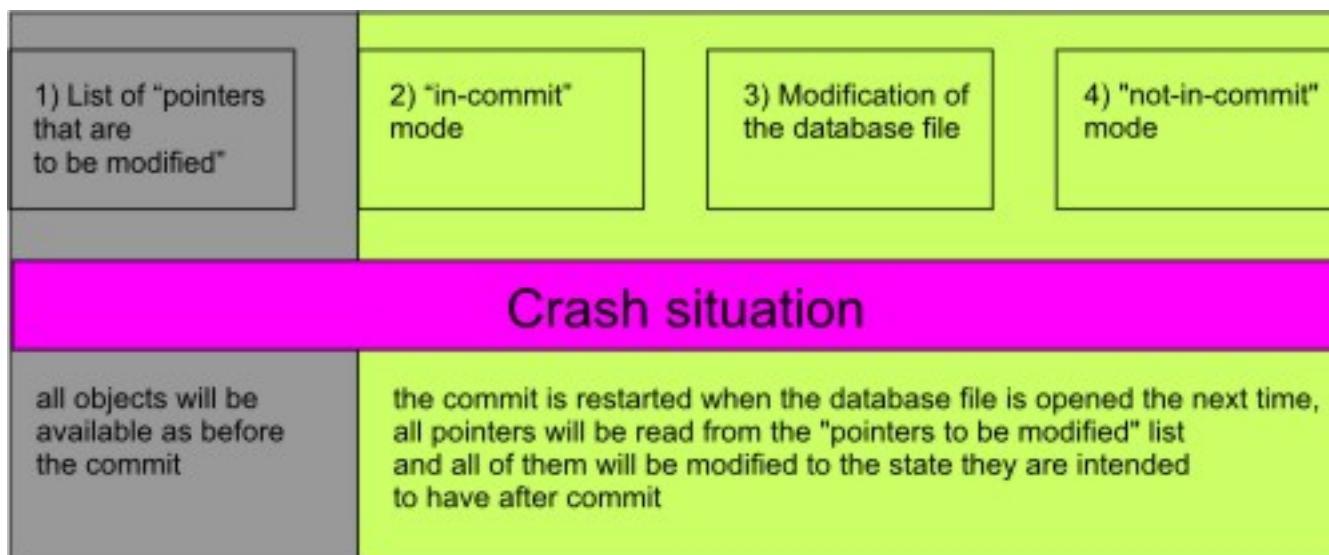
## Effect

---

FlushFileBuffers setting is provided to ensure correct transaction flow in cases of hardware, power or operating system failures.

ACID transaction is ensured when disc writes are fulfilled in the following order.

1. a list of "pointers that are to be modified" is written to the database file;
2. the database file is switched into "in-commit" mode;
3. the pointers are actually modified in the database file;
4. the database file is switched to "not-in-commit" mode.



The Configuration.FlushFileBuffers(true) setting ensures that after each stage of commit process all the buffered data is written to the database file. The write process is comparatively slow and can have a strong impact on performance.

Setting FlushFileBuffers(false) reduces the time spent on transaction commit. From the other side this setting can be potentially dangerous on systems using in-memory file caching. The buffer cache is usually used to improve writing performance. Instead of carrying out all writes immediately, the kernel stores data temporally in the buffer cache, waiting to see if it is possible to group several writes together. Cached file changes can also be reversed. For example, if the same place in a file was changed several times it is enough to write only the final change.

In case of transaction commit such cache management means that transaction data may be lost. Lets consider the case when crash occurs on stage 2-4 and list of "pointers to be modified" is still in cache (completely or partly). After the database file is reopened the commit will be restarted using the list of pointers that is supposed to be written to disc. But in fact we do not know, whether the list was written to disc completely or part of it was still in cache and lost during restart, - so the database can be corrupted.

## Alternate Strategies

---

On operating systems that cache file access, this configuration has to be set to true to ensure each step of transaction being written in order.

Java:

```
configuration.flushFileBuffers(true)
```

Otherwise file caching can be switched off in OS settings.

This revision (14) was last Modified 2007-05-20T13:33:30 by Tetyana.

# B-Tree tuning

## Contents

- [Advantage](#)
- [Effect](#)

Db4o uses special B-tree indexes for increased query performance and reduced memory consumption (the feature was introduced since version 5.4 for class indexes and since 5.7 for field indexes).

## Advantage

---

B-trees are optimized for scenarios when part or all of a data set is on secondary storage such as a hard disk, since disk accesses are extremely expensive operations. B-trees minimize the number of disk accesses required to find data by traversing a sorted tree structure and only need a single disk access per level of the tree.

In order to use B-tree capabilities for field indexes you will simply need to define indexed fields in your classes:

Java:

```
configuration.objectClass(Foo.class).objectField("field").indexed(true)
```

## Effect

---

The caching behaviour of the B-trees can be configured with the following two switches:

Java:

```
configuration.bTreeCacheHeight(height)
```

configures the size of BTree nodes in indexes.

Java:

```
configuration.bTreeNodeSize(size)
```

configures caching of B-tree nodes. Clean B-tree nodes will be unloaded on #commit and #rollback unless they are configured as cached here.

Higher values for the cache height will get you better performance at more RAM consumption.

With the node size you can fine-tune exactly how many reads the B-tree will need to get to leaf nodes. Lower values will allow a lower memory footprint and more efficient reading and writing of small slots. Higher values will reduce the overall number of read and write operations and allow better performance at the cost of more RAM use.

If you raise the number of elements per node and/or the cache depth, you will use more RAM but achieve higher performance. In principle, if you set the node size to a very high value and cache the first node, you should get exactly the same behavior as with the old class indexes.

For now the default settings are 1 for the height of the cache and 100 for the size of the nodes.

When testing B-tree you should remember that B-trees only really start to give you performance advantages with larger numbers of objects. With object counts of 1,000 or 10,000 the old flat index is highly efficient because everything is kept in memory. Using tests with more than 100,000 objects you will really see things degrade with:

- performance, because of a full purge called each time on commit
- memory consumption, because the index will be reloaded completely immediately when the next object is added.

This revision (7) was last Modified 2007-05-07T16:25:05 by Tetyana.

# Inheritance hierarchies

## Contents

- [Advantage](#)
- [Effect](#)
- [Alternate strategies](#)

Do not create inheritance hierarchies, if you don't need them.

## Advantage

---

Avoiding inheritance hierarchies will help you to get better performance as only actual classes will be kept in the class index and in the database.

## Effect

---

Every class in the hierarchy requires db4o to maintain a class index. It is also true for abstract classes and interfaces since db4o has to be able to run a query against them.

## Alternate strategies

---

Class hierarchies and interfaces may be valuable for your application design. You can also use interface/superclass to query for implementations/subclasses.

This revision (3) was last Modified 2006-11-15T08:45:20 by Tetyana.

# Persistent and transient fields

## Contents

- [Advantage](#)
- [Effect](#)
- [Alternate strategies](#)

Do not create fields that you don't need for persistence

## Advantage

---

Storing only needed information will help to keep your database footprint as small as possible.

## Effect

---

If your persistent class contains fields that do not need to be stored you should mark them as transient to prevent them from being stored:

Java:

```
public class NotStorable {  
    private transient int length;  
    . . .  
}
```

You can use [Callbacks](#) or [Translators](#) to set transient fields on retrieval.

Also avoid storing classes having only transient information - their indexes' maintenance will produce unnecessary performance overhead.

## Alternate strategies

---

In some cases you may want to persist class meta-information without the actual object data. The example can be database singleton for [remote code execution](#)

This revision (9) was last Modified 2007-05-07T16:29:51 by Tetyana.

# Activation strategies

## Contents

- [Advantage](#)
- [Effect](#)
- [Alternate strategies](#)

Java:

```
configuration.activationDepth(activationDepth)
```

## Advantage

---

Db4o default activation depth is 5. This setting gives you control over activation depth level depending on your application requirements.

## Effect

---

The two extremes:

- `activationDepth = maximum integer value` - will pop the whole object graph into the memory on every retrieved object. Can be a reasonable solution for shallow objects' design. No need to bother about manual activation;
- `activationDepth = 0` - will reduce memory consumption to the lowest level though leaving all the activation logic for your code.

## Alternate strategies

---

If your object is not fully activated due to the default configuration settings you can activate it manually:

Java: `ObjectContainer#activate(object,depth)`

or use specific object settings:

Java:

```
configuration.objectClass("yourClass").minimumActivationDepth(minimumDepth)
configuration.objectClass("yourClass").maximumActivationDepth(maximumDepth)
```

## Activation strategies

```
configuration.objectClass("yourClass").cascadeOnActivate (bool)  
configuration.objectClass("yourClass").objectField("field").cascadeOnActivate  
(bool)
```

For more information on activation strategies see [Activation chapter](#).

This revision (10) was last Modified 2007-05-07T16:34:31 by Tetyana.

# Automatic Shutdown

## Contents

- [Advantage](#)
- [Effect](#)
- [Alternate Strategies](#)

This topic applies to Java version only

`configuration.automaticShutDown(flag)`

## Advantage

---

Automatic shutdown ensures that all db4o processes are terminated correctly.

## Effect

---

Depending on the JDK, db4o uses one of the following two methods to shut down, if no more references to the ObjectContainer are being held or the JVM terminates:

- JDK 1.3 and above:  
`Runtime.addShutdownHook( ) / code ]`
- JDK 1.2 and below:  
`System.runFinalizersOnExit(true)`

and code in the finalizer.

AutomaticShutDown setting is true by default.

## Alternate Strategies

---

Some JVMs have severe problems with both methods. For these rare cases the automaticShutDown feature may be turned off.

This revision (1) was last Modified 2007-03-06T13:24:10 by Tetyana.

# Reserving Storage Space

## Contents

- [Advantage](#)
- [Effect](#)
- [Alternate Strategies](#)

Java:

```
configuration.reserveStorageSpace(byteCount)
```

## Advantage

---

The allocation of a fixed number of bytes at one time makes it more likely that the database will be stored in one chunk on the mass storage. Less read/write head movement can result in improved performance.

## Effect

---

ReserveStorageSpace setting reserves an additional space (byteCount) in a database file.

The functionality of this setting depends on the context:

- global context: new database files will be created with an additional amount of bytes reserved;
- ObjectContainer context: each call to reserveStorageSpace will allocate an extra space in the database file.

Note: Allocated space will be lost on abnormal termination of the database engine (hardware crash, VM crash). A Defragment run will recover the lost space. For the best possible performance, this method should be called before the Defragment run to configure the allocation of storage space to be slightly greater than the anticipated database file size.

## Alternate Strategies

---

An alternative strategy can be to use MemoryIoAdapter:

Java:

```
MemoryIoAdapter adapter = new MemoryIoAdapter();
```

```
configuration.io(adapter);
```

You can control the growth of the memory file by:

Java:

```
adapter.growBy(100);
```

And you can control the size of the file on disk using RandomAccessFile API. For more information see [MemoryIoAdapter](#).

This revision (2) was last Modified 2007-05-07T16:36:48 by Tetyana.

# Unicode

## Contents

- [Advantage](#)
- [Effect](#)
- [Alternate Strategies](#)

Java:

```
configuration.unicode(false)
```

## Advantage

---

Turning Unicode support off reduces the file storage space for strings by factor 2 and improves performance.

## Effect

---

Enables/disables Unicode string storage format. Unicode allows you to store string data in any language to db4o.

This method needs to be called **before** a database file is created. db4o database files keep their string format after creation.

The default setting is true

## Alternate Strategies

---

You can create your own string marshallers, using TypeHandler4 interface.

This revision (4) was last Modified 2007-11-21T13:35:42 by Tetyana.

# Prefetching IDs For New Objects

## Contents

- [Advantage](#)
- [Effect](#)
- [Alternate Strategies](#)

Java:

```
void ClientServerConfiguration.prefetchIDCount( int prefetchIDCount );
```

Sets the number of IDs to be pre-allocated in the database for new objects created on the client

## Advantage

---

Prefetching several IDs for the new objects created on the client allows to improve the performance by reducing client-server communication.

## Effect

---

When a new object is created on a client, the client should contact the server to get the next available object ID. PrefetchIDCount allows to specify how many IDs should be pre-allocated on the server and prefetched by the client. This method helps to reduce client-server communication.

PrefetchIDCount can be tuned to approximately match the usual amount of objects created in one operation to improve the performance.

If PrefetchIDCount =1 a client will have to connect to the server for each new objects created

If PrefetchIDCount is bigger than the amount of new objects to be created the database will keep unnecessary preallocated space.

The default PrefetchIDCount is 10.

## Alternate Strategies

---

A possible alternative can be using [Messaging](#) for bulk inserts on the server.

This revision (3) was last Modified 2007-05-07T16:41:05 by Tetyana.

# Prefetching Objects For Query Results

## Contents

- [Advantage](#)
- [Effect](#)

Java:

```
void ClientServerConfiguration.prefetchObjectCount(int prefetchObjectCount);
```

Sets the number of objects to be prefetched for an ObjectSet in C/S mode.

## Advantage

---

PrefetchObjectCount setting allows to tune the way object results are delivered from the server.

## Effect

---

Long query results can take significant time to be transferred from the server to the client. This can lead to blocking communication channel. In order to prevent this, query results are fetched from the server in portions. The default portion size is 10 objects.

PrefetchObjectCount setting allows you to change the default prefetched amount.

`prefetchObjectCount = 1` will enforce the client to send a request for each next object in the result set. This can be used to improve the performance for large objects.

With the default setting(10) the client receives the first 10 objects, the next 10 will be delivered on the first request.

Bigger values of prefetchObjectCount can improve the performance for small objects when the whole result set is required for immediate processing on the client.

PrefetchObjectCount setting also influences the processing of queries in lazy and snapshot modes.

This revision (2) was last Modified 2007-03-08T19:35:23 by Tetyana.

# No Class Index

## Contents

- [Advantage](#)
- [Effect](#)
- [Alternate Strategies](#)

Java:

```
configuration.objectClass( "package.classname" ).indexed(false);
```

Turns class index off.

## Advantage

---

Allows to improve the performance to delete and create objects of a class.

## Effect

---

db4o maintains an index for each class to be able to deliver all instances of a class in a query. In some cases class index is not necessary:

- the application always works with subclasses or superclasses;
- there are convenient field indexes that will always find instances of a class;
- the application always works with IDs.

`Indexed(false)` setting will save resources on maintaining the class index on create and delete of the class objects.

## Alternate Strategies

---

Object creation performance can be improved using [`configuration.callConstructors\(true\)`](#) setting.

General read/write performance can be improved with the [`configuration.reserveStorageSpace\(byteCount\)`](#) setting.

This revision (4) was last Modified 2007-05-07T16:42:29 by Tetyana.

# No Field Evaluation

## Contents

- [Advantage](#)
- [Effect](#)
- [Alternate Strategies](#)

Java:

```
configuration.ObjectClass(clazz).ObjectField("field").queryEvaluation(false)
```

## Advantage

---

Improves performance by reducing query evaluation volume

## Effect

---

All fields are evaluated by default. Use this method to turn query evaluation off for specific fields, which are never used in queries. This setting can help you to increase the querying performance by reducing the evaluation task.

## Alternate Strategies

---

Consider marking fields as [transient](#) if their persistence is not necessary.

This revision (2) was last Modified 2007-05-07T16:43:41 by Tetyana.

# RandomAccessFileAdapter

## Contents

- [Advantage](#)
- [Effect](#)
- [Alternate Strategies](#)

This topic applies to db4o versions 6.2 and higher.

Java:

```
configuration.io(new RandomAccessFileAdapter())
```

## Advantage

---

Decreases memory consumption by using an IO adapter without caching.

## Effect

---

Since db4o version 6.2 [CachedIoAdapter](#) is used by default. This IO adapter has some valuable advantages, however the disadvantage is increased memory consumption. Using RandomAccessFileAdapter can help to keep the memory consumption to the minimum.

## Alternate Strategies

---

An alternative way to control the memory consumption is to configure it in the CachedIoAdapter constructor:

Java:

```
configuration.io(new CachedIoAdapter(delegateAdapter, page_size, page_count));
```

page\_size \* page\_count - will define the maximum amount of memory used for caching.

This revision (3) was last Modified 2007-05-07T16:44:32 by Tetyana.

# Commit Strategies

## Contents

- [Advantage](#)
- [Effect](#)
- [Best Strategies](#)

Java:

```
objectContainer.commit();
```

Objects created or instantiated within one db4o transaction are written to a temporary transaction area in the database file and are only durable after the transaction is committed.

Transactions are committed implicitly when the ObjectContainer is closed.

Java:

```
objectContainer.close();
```

## Advantage

---

Committing a transaction makes sure that all the changes are effectively written to a storage location. Commit uses a special sequence of actions, which ensures ACID transactions. The following operations are done during commit:

- flushing modified class indexes
- flushing changes of in-memory field indexes to file-based indexes
- writing all intended pointer changes as a "pre-log" to the file
- writing all pointer changes
- reorganizing the free-space system
- deleting the "pre log"

## Effect

---

Commit is a costly operation as it includes disk writes and flushes of the operating system disk cache. Too many commits can decrease your application's performance. On the other hand long transaction increases the risk of loosing your data in case of a system or a hardware failure.

# Best Strategies

---

- You should call commit() at the end of every logical operation, at a point where you want to make sure that all the changes done get permanently stored to the database.
- If you are doing a bulk insert of many (say >100 000) objects, it is a good idea to commit after every few thousand inserts, depending on the complexity of your objects. If you don't do that, there is not only a risk of losing the objects in a case of a failure, but also a good chance of running out of memory and slowing down the operations due to memory flushes to disk. The exact amount of inserts that can be done safely and effectively within one transaction should be calculated for the concrete system and will depend on available system resources and size and complexity of objects.
- Don't forget to close db4o ObjectContainer before the application exits to make sure that all the changes will be saved to disk during implicit commit.

This revision (2) was last Modified 2007-08-20T12:15:03 by Tetyana.

# Optimizing Native Queries

## Contents

- [Advantage](#)
- [Effect](#)
- [Alternate Strategies](#)

## Advantage

---

Optimized Native Queries allow to achieve considerable performance improvements.

## Effect

---

[Native Queries](#) allow to express a database query in a native object-oriented language. This solution is elegant and straightforward as no mixture of concepts (object and relational) occurs. However, the challenge is to make this solution performant.

If the NQ code is run as is, it requires instantiation of all the members of a class. This is very slow in most cases. In order to improve the performance a special optimizer is used by db4o. The idea of the optimization is to analyze the code in a Native Query and provide an alternative in a database query language. This can be done in runtime or build time.

For detailed information about optimization strategies, please, see [Native Query Optimization](#).

## Alternate Strategies

---

Obviously, optimization is not possible in cases, when a native query does not have a database query alternative. To reveal those cases [db4o Diagnostic](#) system should be used.

This revision (2) was last Modified 2007-08-04T12:08:11 by Tetyana.

# Debugging db4o

Debugging is, in general, a cumbersome and tiring task. And it tends to be harder when various subsystems are tightly coupled, like db4o library and your application. How can db4o help you with the process?

More Reading:

- [Debug Messaging System](#)
- [Customizing The Debug Message Output](#)
- [ExceptionsOnNotStorable](#)
- [Using DTrace](#)
- [Reading Db4o File](#)

This revision (4) was last Modified 2006-11-15T11:54:06 by Tetyana.

# Debug Messaging System

Db4o messaging system is a special tool, which makes db4o functionality more transparent to the user. It can be used:

- in debugging session - to find out where the problem can reside;
- for learning - to watch, what does db4o actually do with the objects.

In order to activate messaging before opening a database file use:

Java:

```
configuration.messageLevel(level)
```

where *level* can be:

level = 0: no messages;

level > 0: normal messages;

level > 1: state messages (new object, object update, delete);

level > 2: activation messages (object activated, deactivated).

In order to set up a convenient output stream for the messages, call:

Java:

```
configuration.setOut(outStream)
```

By default the output is sent to System.out.

For more information on #setOut call see [Customizing The Debug Message Output](#).

#messageLevel(level) also can be set after a database has been opened:

```
Java: ObjectContainer#ext().configure().messageLevel(level)
```

The same applies for #setOut().

Let's use the simplest example to see all types of debug messages:

## DebugExample.java: setCars

```
01 private static void setCars()
02 {  
03     // Set the debug message level to the maximum  
04     Configuration configuration = Db4o.newConfiguration();  
05     configuration.messageLevel(3);  
06  
07     // Do some db4o operations  
08     new File(DB4O_FILE_NAME).delete();  
09     ObjectContainer container=Db4o.openFile(configuration, DB4O_FILE_NAME);  
10    try {  
11        Car car1 = new Car("BMW");  
12        container.set(car1);  
13        Car car2 = new Car("Ferrari ");  
14        container.set(car2);  
15        container.deactivate(car1, 2);  
16        Query query = container.query();  
17        query.constrain(Car.class);  
18        ObjectSet results = query.execute();  
19        listResults(results);  
20    } finally {  
21        container.close();  
22    }  
23 }
```

Output looks quite messy, but allows you to follow the whole process. For debugging purposes messaging system provides a timestamp and internal ID information for each object (first number in state and activate messages).

This revision (13) was last Modified 2007-05-07T16:52:30 by Tetyana.

# Customizing The Debug Message Output

[Debug Messaging System](#) topic explains how to activate the debug messages in your application. However the default console output has very limited possibilities and can only be used effectively in console applications. To use debug messaging system in any environment db4o gives you an API to redirect debug output to another stream:

Java:

```
Configuration.setOut(java.io.PrintStream)
```

An example below shows how to create and use a log file for debug messages:

DebugExample.java: setCarsWithFileOutput

```

01 private static void setCarsWithFileOutput() throws FileNotFoundException
02 {
03     // Create StreamWriter for a file
04     FileOutputStream fos = new FileOutputStream("Debug.txt");
05     PrintStream debugWriter = new PrintStream(fos);
06
07     // Redirect debug output to the specified writer
08     Configuration configuration = Db4o.newConfiguration();
09     configuration.setOut(debugWriter);
10
11     // Set the debug message level to the maximum
12     configuration.messageLevel(3);
13
14     // Do some db4o operations
15     new File(DB4O_FILE_NAME).delete();
16     ObjectContainer container=Db4o.openFile(configuration, DB4O_FILE_NAME);
17     try {
18         Car car1 = new Car("BMW");
19         container.set(car1);
20         Car car2 = new Car("Ferrari");

```

```
21 |     container.set(car2);  
22 |     container.deactivate(car1, 2);  
23 |     Query query = container.query();  
24 |     query.constrain(Car.class);  
25 |     ObjectSet results = query.execute();  
26 |     listResult(results);  
27 | } finally {  
28 |     container.close();  
29 |     debugWriter.close();  
30 | }  
31 }
```

Using a log file for debug messages has several advantages:

- debug information is available after the application has terminated;
- console output is not polluted with debug messages;
- debug information from the clients can be available on the server.

You can always switch back to the default setting using:

Java:

```
Configuration.setOut(System.out)
```

This revision (3) was last Modified 2007-01-18T19:21:03 by Tetyana.

# ExceptionsOnNotStorable

There is another setting that can be of great value in debug process:

Java:

```
configuration.exceptionsOnNotStorable (true)
```

In some environments (especially those that provide plugin mechanics or perform some kind of class reloading) you may encounter strange problems due to classloader issues. These environments include servlet containers, Eclipse plugins, reloading JUnit test runners, etc.

In most of those cases db4o will fail quietly (i.e. will not deliver any results on queries), unless you have configured #exceptionsOnNotStorable(true) - then you may see messages related to not being able to store db4o internal classes or set db4o internal translators, etc.

Db4o uses the context classloader by default. This is an appropriate choice in most situations, but it's not really reliable, since the concrete context classloader depends on the environment you're running in. Therefore you can explicitly specify the classloader to be used for db4o operation by calling

Java:

```
configuration.reflectWith(new JdkReflector(classLoader))
```

Basically you just have to find out the appropriate classloader and configure db4o accordingly. The right choice depends on the specific classloader hierarchy of your application context. Two examples:

- In servlet containers, there usually should be no problem, since most containers automatically set the context classloader to the webapp classloader. So it shouldn't matter, whether the db4o.jar resides in your webapp's WEB\_INF/lib (where it will be loaded by the appropriate classloader itself, anyway) or in one of the shared lib folders (where the classloader responsible for loading db4o will not be able to see webapp specific classes).
- In Eclipse, the context classloader is the system classloader, which is agnostic of plugin-specific classes. You'll have to configure db4o to use your plugin's classloader, e.g. MyPlugin.class.getClassLoader(). (If the db4o.jar resides in your plugin, you'll get the same effect by just using Db4o.class.getClassLoader()).

The approach to solving classloader problems (not only for db4o, but generally) is:

- identify the classes/libs db4o needs to know
- identify the classloader hierarchy of your application context
- use the most generic classloader that knows all needed classes, either directly or indirectly via delegation

See also:[Classloader issues](#)

#exceptionsOnNotStorable(true) will also help you to identify classes that db4o cannot persist.

db4o needs a constructor that it can use to create user objects. Ideally this is a zero-parameter constructor (declared public for Java JDK versions prior to JDK 1.2). If db4o does not find a zero-parameter constructor, it iterates through all other constructors and internally attempts to create an instance of an object by passing appropriate null parameters. If this is successful with any of the present constructors, this constructor is used.

There are classes that do not have usable constructors. `java.net.URL` is an example from the Java JDK. In this case you have the following options:

- add a zero-parameter constructor specifically for db4o;
- derive from the class and add a zero-parameter constructor;
- add a custom translator.

If you need to quickly implement a solution for one of the JDK classes, and querying members is not an issue, you may choose to use the built-in serializable translator. Here is an example, how this is done for `java.net.URL`:

Java:

```
configuration.objectClass("java.net.URL").translate(new TSerializable());
```

The above code needs to be executed every time before the db4o engine is started. See also:  
[Constructors, Translators](#).

Another db4o system, which can give you a valuable feedback about db4o functioning in your application is [Diagnostics](#).

This revision (10) was last Modified 2007-05-07T16:55:45 by Tetyana.

# Using DTrace

This topic applies to Java version only.

If you are interested in internal db4o debugging you can make use of com.db4o.DTrace class. DTrace is basically a logging agent, which gets called for db4o core events, which can provide information valuable for debugging. The list of currently available events can be found in DTrace class, they are represented by DTrace type static variables, for example:

```
public static DTrace ADD_TO_CLASS_INDEX;

public static DTrace BEGIN_TOP_LEVEL_CALL;
```

DTrace information is not enabled by default, as it can make the system really slow. In order to enable DTrace make the following changes in DTrace class and rebuild the core:

- set `public static final boolean enabled = true`
- modify the `configure` method to tell DTrace that you are interested in all ranges and IDs
  - `addRangeWithLength(0, Integer.MAX_VALUE);`
- make sure that no `turnAllOffExceptFor` method is called in the `configure()` method

With this DTrace setup you will basically see everything that is happening logged to the console in detail. However, this information can be excessive and difficult to handle. That is why DTrace provides different configurations, allowing to limit the range of information you are interested in.

1. `turnAllOffExceptFor(DTrace[] these)`

This method allows you to pass an array of DTrace events, which you want to see in the console. For example:

```
turnAllOffExceptFor(new DTrace[] { ADD_TO_CLASS_INDEX ,
BEGIN_TOP_LEVEL_CALL })
```

2. `addRange(long)`

```
addRangeWithEnd(long start, long end)
```

```
addRangeWithLength(long start, long length)
```

These methods allow to specify a range of addresses in a database file that you are interested in. `addRange` methods are especially useful for debugging a database file structure. Note, that in db4o

internal object ID corresponds to the object's address in the file.

### 3. trackEventsWithoutRange( )

These method will allow all events with no range specified to log their information.

The format of the output message is the following:

: [event number] : [start address] : [start address] :[information]

*event number* - sequential event number

*start address* - start of the event address range (optional)

*end address* - end of the event address range (optional)

*information* - informational message from the event

If DTrace log messages are not enough for you to track the problem, you can use DTrace in debug mode. Use `breakOnEvent(long)` method to specify on which address DTrace must break and put a breakpoint inside `breakPoint()` method.

As it was mentioned before DTrace events are already created in the most important execution points of db4o core. However, if you need more events, feel free to add them, encapsulating the calls with `if (DTrace.enabled)` to make sure that your code is removed from distributions by the compiler.

This revision (3) was last Modified 2007-09-14T18:09:04 by Tetyana.

# Reading Db4o File

For debugging, learning and teaching purposes, the db4o file format can be modified to be (nearly ?) human readable.

To do this, simply compile the sources with Deploy.debug set to true, run an application that creates a db4o database file and look at the file with any editor.

With the Deploy.debug setting all pointers in the database file will be readable with their physical address as a readable number.

All other slots will be identifiable by a single character at the beginning. An index that explains the character constants can be found in com.db4o.internal.Const4.

To understand the format best, you may want to look at the [File Header](#) structure and at the #readThis() methods of classes derived from PersistentBase, like ClassMetadataRepository for instance.

This functionality proved to be very useful when db4o was originally written. By marking freespace with XXXXes a bug in the format could be spotted immediately by visual inspection of a database file.

To navigate through a database file in your favourite editor, it will work best if you write a macro for this editor that allows you to mark and select a number in the database file and to hit a button in the editor to jump to the corresponding offset in the database file (number of characters from the beginning).

Such macro for Microsoft Word is presented below:

## OffsetNavigator.Vb

```

1 Sub SearchOffset()
2     Dim pos As Integer
3     pos = Val(Selection.Text)
4     If pos = 0 Then
5         MsgBox ("The selection is not a number")
6     Else
7         ActiveDocument.Content.Characters(pos).Select
8     End If
9 End Sub

```

To make use of it, open Visual Basic Macro editor within your Word environment, create a new Macro in the Normal template and paste the code above. In order to make its usage easy assign a key sequence to call the macro command:

- open Tools/Customize/Commands/Keyboard;
- select "Macros" as a Category and the newly-created macro name in the Commands list;
- press a new key sequence for the command and press "Assign".

Now you can navigate through the human-readable db4o file using the selected key sequence.

This revision (1) was last Modified 2007-09-15T13:59:47 by Tetyana.

# Diagnostics

Db4o engine provides user with a special mechanism showing runtime diagnostics information. This functionality can become your guide to excellent performance and low memory consumption. Diagnostics can be switched on in the configuration before opening the database file:

Java:

```
configuration.diagnostic().addListener(new DiagnosticListener())
```

where DiagnosticListener is a callback interface tracking diagnostic messages from different parts of the system:

```
public interface DiagnosticListener {  
    public void onDiagnostic(Diagnostic d);  
}
```

Db4o provides 2 different listeners:

- DiagnosticToConsole (Java, prints diagnostic messages to the console);
- DiagnosticToTrace (.NET, prints diagnostic messages to the debug output window).

Every diagnostic message is represented by it's own type, all possible types can be found in the com.db4o.diagnostic package/namespace.

At the present moment the following diagnostic classes are implemented:

- ClassHasNoFields
- LoadedFromClassIndex
- NativeQueryNotOptimized
- UpdateDepthGreaterOne
- DescendIntoTranslator

More Reading:

- [ClassHasNoFields](#)
- [LoadedFromClassIndex](#)
- [NativeQueryNotOptimized](#)
- [UpdateDepthGreaterOne](#)
- [Diagnostic Messages Filter](#)
- [DescendIntoTranslator](#)

This revision (6) was last Modified 2007-05-07T16:57:08 by Tetyana.

# ClassHasNoFields

This diagnostic type provides information about classes in your persistent class hierarchy that have no persistent fields. The diagnostic message appears when the class is saved to the database. It is recommended to remove such classes from the database to avoid the overhead for the maintenance of class indexes.

Let's look at the following example:

Empty.java

```
01 /* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */
02 package com.db4odoc.diagnostics;
03
04 import java.util.Calendar;
05 import java.text.DateFormat;
06
07
08 public class Empty {
09 |
10     public Empty() {
11     }
12 |
13     public String CurrentTime()
14     {
15         Calendar cl = Calendar.getInstance();
16         DateFormat df = DateFormat.getDateInstance();
17         String time = df.format(cl.getTime());
18         return time;
19     }
20 |
21     public String ToString()
22     {
23         return CurrentTime();
24     }
25 }
```

## DiagnosticExample.java: setEmptyObject

```
1 [+]private static void setEmptyObject(ObjectContainer container) {  
2 |     Empty empty = new Empty();  
3 |     container.set(empty);  
4 } 
```

## DiagnosticExample.java: testEmpty

```
01 [+]private static void testEmpty() {  
02 |     Configuration configuration = Db4o.newConfiguration();  
03 |     configuration.agnostic().addListener(new DiagnosticToConsole());  
04 |     new File(DB40_FILE_NAME).delete();  
05 |     ObjectContainer container=Db4o.openFile(configuration, DB40_FILE_NAME);  
06 [+]     try {  
07 |         setEmptyObject(container);  
08 |     }  
09 [+]     finally {  
10 |         container.close();  
11 |     }  
12 } 
```

&gt;

Diagnostic message is produced when the execution point reaches

```
db.set(empty)
```

Empty class does not keep any information and can be left in the application code; there is no need to put it in the database.

This revision (6) was last Modified 2006-11-14T13:15:25 by Tetyana.

# LoadedFromClassIndex

This diagnostic class is provided to keep track of all the queried fields in your application that have no indexes. The output produced can give a comprehensive picture of index tuning required.

This revision (3) was last Modified 2006-11-14T14:03:17 by Tetyana.

# NativeQueryNotOptimized

This diagnostics object informs you that a Native Query cannot be optimized. It means that it will be run by instantiating all objects of the candidate class. Try to simplify your query expression.  
 For an example let's look at a predicate using 2 different unrelated clauses.

## ArbitraryQuery.java

```

01 /* Copyright (C) 2004 - 2007 db4objects Inc. http://www.db4o.com */
02 package com.db4odoc.diagnostics;
03
04 import com.db4o.query.Predicate;
05
06 public class ArbitraryQuery extends Predicate<Pilot>{
07     public int[] points;
08
09     public ArbitraryQuery(int[] points) {
10         this.points=points;
11     }
12
13     public boolean match(Pilot pilot) {
14         for (int i = 0; i < points.length; i++) {
15             if (((Pilot)pilot).getPoints() == points[i])
16             {
17                 return true;
18             }
19         }
20         return ((Pilot)pilot).getName().startsWith("Rubens");
21     }
22 }
```

## DiagnosticExample.java: queryPilot

```

1 private static void queryPilot(ObjectContainer container) {
2     int[] i = new int[]{19, 100};
3     ObjectSet result = container.query(new ArbitraryQuery(i));
```

```
4 |     listResult(result);
5 | }
```

## DiagnosticExample.java: testArbitrary

```
01 private static void testArbitrary() {
02 |     Configuration configuration = Db4o.newConfiguration();
03 |     configuration.diagnostic().addListener(new DiagnosticToConsole());
04 |     new File(DB4O_FILE_NAME).delete();
05 |     ObjectContainer container=Db4o.openFile(configuration, DB4O_FILE_NAME);
06 |     try {
07 |         Pilot pilot = new Pilot("Rubens Barrichello", 99);
08 |         container.set(pilot);
09 |         queryPilot(container);
10 |     }
11 |     finally {
12 |         container.close();
13 |     }
14 | }
```

This revision (4) was last Modified 2006-10-26T14:10:30 by Tetyana.

# UpdateDepthGreaterOne

UpdateDepth configuration setting allows you to specify the level of objects' enclosure where update command will still be valid:

Java:

```
configuration.updateDepth(depth)
```

This setting has a considerable impact on performance and can make the application very slow. It is recommended to keep the default configuration setting (UpdateDepth(0)) and specify UpdateDepth for selected classes, where cascaded update will be really useful:

Java:

```
configuration.objectClass(Car.class).updateDepth(3)
```

This revision (8) was last Modified 2008-03-29T10:29:21 by Tetyana.

# Diagnostic Messages Filter

The standard listeners can potentially produce quite a lot of messages. By writing your own DiagnosticListener you can filter that information.

On the stage of application tuning you can be interested in optimizing performance through indexing. Diagnostics can help you with that giving information about queries that are running on un-indexed fields. Having this information you can decide which queries are frequent and heavy and should be indexed, and which have little performance impact and do not need an index. Field indexes dramatically improve query performance but they may considerably reduce storage and update performance.

In order to get rid of all unnecessary diagnostic information and concentrate on indexes let's create special diagnostic listener:

IndexDiagListener.java

```

01 /* Copyright (C) 2004 - 2007 db4objects Inc. http://www.db4o.com */
02 package com.db4odoc.diagnostics;
03
04 import com.db4o.diagnostics.*;
05
06 public class IndexDiagListener implements DiagnosticListener
07 {
08     public void onDiagnostic(Diagnostic d) {
09         if (d.getClass().equals(LoadedFromClassIndex.class)) {
10             System.out.println(d.toString());
11         }
12     }
13 }
```

We can check the efficacy of IndexDiagListener using queries from the previous paragraphs:

DiagnosticExample.java: testIndexDiagnostics

```

01 private static void testIndexDiagnostics() {
02     new File(DB4O_FILE_NAME).delete();
03
04     Configuration configuration = Db4o.newConfiguration();
05     configuration.diagnostics().addListener(new IndexDiagListener());
```

```
06 |     configuration.updateDepth(3);  
07 |  
08 |     ObjectContainer container=Db4o.openFile(configuration, DB4O_FILE_NAME);  
09 |     try {  
10 |         Pilot pilot1 = new Pilot("Rubens Barrichello", 99);  
11 |         container.set(pilot1);  
12 |         Pilot pilot2 = new Pilot("Michael Schumacher", 100);  
13 |         container.set(pilot2);  
14 |         queryPilot(container);  
15 |         setEmptyObject(container);  
16 |         Query query = container.query();  
17 |         query.constraint(Pilot.class);  
18 |         query.descend("points").constraint(new Integer(99));  
19 |         ObjectSet result = query.execute();  
20 |         listResult(result);  
21 |     }  
22 |     finally {  
23 |         container.close();  
24 |     }  
25 | }
```

Potentially this piece of code triggers all the diagnostic objects, but we are getting only index warning messages due to IndexDiagListener.

This revision (9) was last Modified 2007-05-07T16:59:33 by Tetyana.

# DescendIntoTranslator

Translator API provides a special way of storing and retrieving objects. In fact the actual class is not stored in the database. Instead the information from that class is stored in a primitive object (object array) and the class is recreated during instantiation or activation.

Let's look how queries handle translated classes. Diagnostics system will help us to see, what is going on.

In our example class Car is configured to be saved and retrieved with CarTranslator class. CarTranslator saves only car model information appending it with the production date.

**CarTranslator.java**

```

01  /* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */
02  package com.db4odoc.diagnostics;
03
04  import com.db4o.*;
05  import com.db4o.config.*;
06
07  public class CarTranslator
08  implements ObjectConstructor {
09  public Object onStore(ObjectContainer container,
10  Object applicationObject) {
11  Car car = (Car) applicationObject;
12
13  String fullModel;
14  if (hasYear(car.getModel())){
15    fullModel = car.getModel();
16  } else {
17    fullModel = car.getModel() + getYear(car.getModel());
18  }
19  return new Object[]{fullModel};
20}
21
22 private String getYear(String carModel) {
23  if (carModel.equals("BMW")){
24    return " 2002";
25  } else {

```

```

26 |         return " 1999";
27 |     }
28 |
29 | }
30 |
31 |     private boolean hasYear(String carModel) {
32 |         return false;
33 |     }
34 |
35 |     public Object onInstantiate(ObjectContainer container, Object storedObject) {
36 |         Object[] raw=(Object[]) storedObject;
37 |         String model=(String) raw[0];
38 |         return new Car(model);
39 |     }
40 |
41 |     public void onActivate(ObjectContainer container,
42 |                             Object applicationObject, Object storedObject) {
43 |     }
44 |
45 |     public Class storedClass() {
46 |         return Object[].class;
47 |     }
48 | }
```

TranslatorDiagListener is implemented to help us filter only those diagnostic messages, that concern translated classes (filtering diagnostics messages is explained in [Diagnostic Messages Filter chapter](#)).

We did not get any diagnostic messages here and the result shows the stored cars with extended model values.

To test Native Queries we will use the predicate, which retrieves only cars, produced in year 2002:

#### NewCarModel.java

```

01 /* Copyright (C) 2004 - 2007 db4objects Inc. http://www.db4o.com */
02 package com.db4odoc.diagonstics;
03
04 import com.db4o.query.Predicate;
05
```

```

06 [+] public class NewCarModel extends Predicate<Car> {
07 [+]   public boolean match(Car car) {
08 |     return ((Car)car).getModel().endsWith("2002");
09 |   }
10 [ ]

```

## DiagnosticExample.java: retrieveTranslatedCarsNQ

```

01 [+] private static void retrieveTranslatedCarsNQ() {
02 |   Configuration configuration = Db4o.newConfiguration();
03 |   configuration.agnostic().addListener(new TranslatorDiagnosticListener());
04 |   configuration.exceptionsOnNotStorable(true);
05 |   configuration.objectClass(Car.class).translate(new CarTranslator());
06 |   configuration.objectClass(Car.class).callConstructor(true);
07 |   ObjectContainer container = Db4o.openFile(configuration, DB4O_FILE_NAME);
08 [+]   try {
09 |     ObjectSet result = container.query(new NewCarModel());
10 |     listResult(result);
11 [+]   } finally {
12 |     container.close();
13 [ ]
14 [ ]

```

A diagnostic message should appear pointing out, that the query is not correct in our case. Let's try to correct it using unoptimized NQ and evaluations.

## DiagnosticExample.java: retrieveTranslatedCarsNQUnopt

```

01 [+] private static void retrieveTranslatedCarsNQUnopt() {
02 |   Configuration configuration = Db4o.newConfiguration();
03 |   configuration.optimizeQueries(false);
04 |   configuration.agnostic().addListener(new TranslatorDiagnosticListener());
05 |   configuration.exceptionsOnNotStorable(true);
06 |   configuration.objectClass(Car.class).translate(new CarTranslator());
07 |   configuration.objectClass(Car.class).callConstructor(true);
08 |   ObjectContainer container = Db4o.openFile(configuration, DB4O_FILE_NAME);
09 [+]   try {

```

```
10 |     ObjectSet result = container.query(new NewCarModel());
11 |     listResult(result);
12 | } finally {
13 |     container.close();
14 | }
15 | }
```

In both cases we the results are correct. Native Query optimization cannot be used with the translated classes, because the actual values of the translated fields are only known after instantiation and activation. That also means that translated classes can have a considerable impact on database performance and should be used with care.

This revision (13) was last Modified 2006-11-14T13:18:35 by Tetyana.

# Native Query Optimization

## Contents

- [Optimization Theory](#)
- [NQ Optimization For Java](#)

Native Queries will run out of the box in any environment. If optimization is turned on (default) Native Queries will be converted to [SODA](#) where this is possible, allowing db4o to use indexes and optimized internal comparison algorithms. Otherwise Native Query may be executed by instantiating all objects, using [SODA Evaluations](#). Naturally performance will not be as good in this case.

## Optimization Theory

---

For Native Query Java bytecode and .NET IL code are analyzed to create an AST-like expression tree. Then the flow graph of the expression tree is analyzed and converted to a SODA query graph.

For example:

Java:

```
List<Pilot> pilots = container.query(new Predicate<Pilot>() {
    public boolean match(Pilot
pilot) {
        return pilot.
getName().equals("Michael Schumacher")
&& pilot.getPoints() == 100;
    }
}) ;
```

First of all the following code will be extracted:

```
query#constrain(Pilot)
```

Then a more complex analysis will be run to convert the contents of the #match method into a SODA-understandable syntax. On a simple example it is easy to see what will happen:

Java:

```
return pilot.getName().equals("Michael Schumacher") && pilot.getPoints() == 100;
```

easily converts into:

```
CANDIDATE.name == "Michael Schumacher"
```

```
CANDIDATE.points == 100
```

## NQ Optimization For Java

---

NQ optimisation on Java requires db4onqopt.jar and bloat.jar to be present in the CLASSPATH.

The Native Query optimizer is still under development to eventually "understand" all Java constructs. Current optimization supports the following constructs well:

- compile-time constants
- simple member access
- primitive comparisons
- `#equals()` on primitive wrappers and Strings
- `#contains()/#startsWith()/#endsWith()` for Strings
- arithmetic expressions
- boolean expressions
- static field access
- array access for static/predicate fields
- arbitrary method calls on static/predicate fields (without candidate based params)
- candidate methods composed of the above
- chained combinations of the above

This list will constantly grow with the latest versions of db4o.

Note that the current implementation doesn't support polymorphism and multiline methods yet.

db4o for Java supplies three different possibilities to run optimized native queries, optimization at

1. [query execution time](#)
2. [deployment time](#)
3. [class loading time](#)

For more information on NQ optimization see [Monitoring Optimization](#).

This revision (21) was last Modified 2008-03-15T17:44:47 by Tetyana.

# Optimization at Query Execution Time

This topic applies to Java version only

**Note:** This will not work with JDK1.1.

To enable code analysis and optimization of native query expressions at query execution time, you just have to add db4o-x.x-nqopt.jar and bloat-1.x.jar to your CLASSPATH. Optimization can be turned on and off with the following configuration setting:

```
Db4o.configure().optimizeNativeQueries(boolean optimizeNQ)
```

This revision (5) was last Modified 2006-12-04T07:51:16 by Tetyana.

# NQ Optimization At Load Time

This topic applies to Java version only

**Note:** This will not work with JDK1.1.

Native Query predicates can be optimized when they are loaded into JVM. In order to do that you should make use of db4o [Enhancement Tools](#).

The idea is very simple:

- you create your application without any worries about NQ optimization
- when the application is ready, you use a special starter class, which calls a special classloader to instrument your predicates and start the application.

Let's look how this is done on an example. We will use a well-known [Pilot](#) class, store it and use NQ to retrieve it:

NQExample.java: main

```
1 public static void main(String[] args) {
2     storePilots();
3     selectPilot5Points();
4 }
```

NQExample.java: storePilots

```
01 private static void storePilots() {
02     new File(DB4O_FILE_NAME).delete();
03     ObjectContainer container = database(configureNQ());
04     if (container != null) {
05         try {
06             Pilot pilot;
07             for (int i = 0; i < OBJECT_COUNT; i++) {
08                 pilot = new Pilot("Test Pilot #" + i, i);
09                 container.set(pilot);
10             }
11             for (int i = 0; i < OBJECT_COUNT; i++) {
12                 pilot = new Pilot("Professional Pilot #" + (i + 10), i + 10);
13                 container.set(pilot);
14             }
15             container.commit();
16         } catch (Db4oException ex) {
```

```

17 |         System.out.println("Db4o Exception: " + ex.getMessage());
18 |     } catch (Exception ex) {
19 |         System.out.println("System Exception: " + ex.getMessage());
20 |     } finally {
21 |         closeDatabase();
22 |     }
23 | }
24 L }
```

## NQExample.java: selectPilot5Points

```

01 L private static void selectPilot5Points() {
02 |     ObjectContainer container = database(configureNQ());
03 |     if (container != null) {
04 |         try {
05 |             List<Pilot> result = container.query(new Predicate<Pilot>() {
06 |                 public boolean match(Pilot pilot) {
07 |                     // pilots with 5 points are included in the
08 |                     // result
09 |                     return pilot.getPoints() == 5;
10 |                 }
11 |             });
12 |             listResult(result);
13 |         } catch (Exception ex) {
14 |             System.out.println("System Exception: " + ex.getMessage());
15 |         } finally {
16 |             closeDatabase();
17 |         }
18 |     }
19 L }
```

We will need to create a starter class, which will call the main method of the NQExample:

## NQEnhancedStarter.java: main

```

1 L public static void main(String[] args) throws Exception {
2 |     // Create class filter to point to the predicates to be optimized
3 |     ClassFilter filter = new ByNameClassFilter("com.db4odoc.nqoptimize.", true);
4 |     // Create NQ optimization class edit
```

```
5 |     Bl ootClassEdit[] edits = { new TranslateNQToSODAEdit() };
6 |     URL[] urls = { new File("/work/workspaces/db4o/nqtest/bin").toURI().toURL() };
7 |     // launch the application using the class edit and the filter
8 |     Db4oInstrumentationLauncher.launch(edits, urls, NQExample.class.getName(), new String[]
{} );
9 | }
```

That's all. Now you can run your application using NQEnhancedStarter and all the predicated will be optimized while they are loaded. This will also save time on optimization at runtime.

This revision (7) was last Modified 2007-11-05T18:35:31 by Tetyana.

# Pilot

Pilot.java

```
01/* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */  
02  
03 package com.db4odoc.nqoptimize;  
04  
05public class Pilot{  
06     private String name;  
07  
08     private int points;  
09  
10    public Pilot(String name, int points) {  
11        this.name = name;  
12        this.points = points;  
13    }  
14  
15    public String getName() {  
16        return name;  
17    }  
18  
19    public void setName(String name) {  
20        this.name = name;  
21    }  
22  
23    public int getPoints() {  
24        return points;  
25    }  
26  
27    public boolean equals(Object obj) {  
28        if (obj instanceof Pilot) {
```

```
29 |     return (((Pilot) obj).getName().equals(name) &&
30 |             ((Pilot) obj).getPoints() == points);
31 |
32 |     return false;
33 |
34 |
35 [§] public String toString() {
36 |     return name + "/" + points;
37 |
38 |
39 [§] public int hashCode() {
40 |     return name.hashCode() + points;
41 |
42 }
```

This revision (2) was last Modified 2007-11-04T19:19:18 by Tetyana.

# NQ Optimization At Build Time

This topic applies to Java version only

Note: Instrumented optimized classes will work with JDK1.1, but the optimization process itself requires at least JDK 1.3.

In the [previous topic](#) we discussed how NQ optimization can be enabled on classes while they are loaded. In this topic we will look at even more convenient and performant way of enhancing classes to optimize NQ: during application build time.

For our example we will take the same classes as in the [previous example](#), with the exception of NQEnhancedStarter class, which won't be needed for build-time enhancement. Its functionality will be fulfilled by the build script. For this example we will create an ant script, which should be run after the classes (or jar) is built.

For simplistic example our build script should:

- Use classes, created by normal build script
- Create a new enhanced-bin folder for the enhanced classes
- Use NQAntClassEditFactory to create TranslateNQToSODAEedit (can be based on class filter)
- Call Db4oFileEnhancerAntTask#execute, which will call Db4oClassInstrumenter#enhance passing the previously created TranslateNQToSODAEedit to optimize NQ in the supplied classes.

This can be done with the following script:

## Build.Xml

```
01 <?xml version="1.0"?>
02
03 <!--
04   NQ optimization build time enhancement sample.
05 -->
06
07 <project name="nqenhance" default="buildall">
08
09 <!--
10   Set up the required class path for the enhancement task.
```

```
11 In a production environment, this will be composed of jars, of
course.

12 -->

13 <path id="db4o.enhance.path">
14   <pathel ement path="${basedir}" />
15   <fileset dir="lib">
16     <include name="**/*.jar"/>
17   </fileset>
18 </path>
19

20 <!-- Define enhancement task. -->
21 <taskdef
22   name="db4o-enhance"
23   classname="com.db4o.instrumentation.ant.Db4oFileEnhancerAntTask"
24   classpathref="db4o.enhance.path"
25   loaderref="db4o.enhance.loader" />
26

27 <typedef
28   name="native-query"
29   classname="com.db4o.nativequery.main.NQAntClassEditFactory"
30   classpathref="db4o.enhance.path"
31   loaderref="db4o.enhance.loader" />
32

33

34 <target name="buildall">
35
36   <!-- Create enhanced output directory-->
37   <mkdir dir="${basedir}/enhanced-bin" />
38   <delete dir="${basedir}/enhanced-bin" quiet="true">
39     <include name="**/*"/>
40   </delete>
41
42   <db4o-enhance targetdir="${basedir}/enhanced-bin">
```

```

43      <classpath refid="db4o enhance. path" />
44          <!-- Use compiled classes as an input -->
45      <sources dir="${basedir}/bin" />
46
47          <!-- Call transparent activation enhancement -->
48      <native-query />
49
50
51  </db4o-enhance>
52
53 </target>
54
55
56
57 </project>
```

In order to test this script:

- Create a new project, consisting of NQExample and Pilot classes from the [previous example](#)
- Add lib folder to the project root and copy the following jars from db4o distribution:
  - bloat-1.0.jar
  - db4o-x.x-classedit.jar
  - db4o-x.x-java5.jar
  - db4o-x.x-nqopt.jar
  - db4o-x.x-tools.jar
 (Note, that the described functionality is only valid for db4o releases after 7.0)
- Build the project with your IDE or any other build tools (it is assumed that the built class files go to the project's bin directory)
- Copy build.xml into the root project folder and execute it

Successfully executed build script will produce an instrumented copy of the project classes in enhanced-bin folder. You can check the results by running the following batch file from bin and enhanced-bin folders:

```
set CLASSPATH=.;${PROJECT_ROOT}\lib\db4o-x.x-java5.jar
```

```
java com.db4odoc.nqoptimize.NQExample
```

Of course, the presented example is very simple and limited in functionality. In fact you can do a lot more things using the build script:

- o Add TA instrumentation in the same enhancer task
- o Use ClassFilter to select classes for enhancement
- o Use regex to select classes for enhancement
- o Use several source folders
- o Use jar as the source for enhancement

An example of the above features can be found in our [Project Spaces](#).

This revision (9) was last Modified 2008-04-14T16:34:17 by Tetyana.

# NQ Optimization On CF2.0

This topic applies to .NET version only

This revision (9) was last Modified 2007-11-28T14:39:00 by Tetyana.

# Build-time Optimization For .NET

This topic applies to .NET version only

This revision (8) was last Modified 2007-11-27T19:26:00 by Tetyana.

# Monitoring Optimization

This topic applies to Java version only.

This feature is not complete and can only be used for experiments

Currently you can only attach a listener to the ObjectContainer:

NQExample.java: nqListener

```

1 public static void nqListener() {
2     ObjectContainer db = Db4o.openFile(DBFILENAME);
3     ((InternalObjectContainer)db).getNativeQueryHandler().addListener(
4         new Db4oQueryExecutionListener() {
5             public void notifyQueryExecuted(NQOptimizationInfo info) {
6                 System.out.println(info);
7             }
8         });
9 }
```

The listener will be notified on each native query call and will be passed the Predicate object processed, the optimized expression tree (if successful) and the success status of the optimization run:

ObjectContainerBase.UNOPTIMIZED ("UNOPTIMIZED")

if the predicate could not be optimized and is run in unoptimized mode

ObjectContainerBase.PREOPTIMIZED ("PREOPTIMIZED")

if the predicate already was optimized (due to class file or load time instrumentation)

ObjectContainerBase.DYNOPTIMIZED ("DYNOPTIMIZED")

if the predicate was optimized at query execution time

This revision (10) was last Modified 2007-09-08T14:45:39 by Tetyana.

# Utility Methods

In this chapter we will review utility methods provided by extended ObjectContainer interface. Use API documentation for more information on ExtObjectContainer methods.

More Reading:

- [PeekPersisted](#)
- [IsActive](#)
- [IsStored](#)
- [Descend](#)

This revision (6) was last Modified 2006-11-15T11:33:33 by Tetyana.

# PeekPersisted

Db4o loads each object into reference cache only once in the session, thus ensuring that independently of the way of retrieving, you will always get a reference to the same object. This concept certainly makes things clearer, but in some cases you will want to operate on the copy of an object.

Typical usecases can be:

- comparing object's changes in a running transaction with the original object in a database;
- safely changing an object without making changes to the database;
- modifying an object in several threads independently, writing the changes to the database after conflict resolution.

Db4o helps you with these tasks providing the following method:

Java: `ExtObjectContainer.peekPersisted(object, depth, committed)`

This method creates a copy of a database object in memory instantiating its members up to depth parameter value. The object has no connection to the database.

Committed parameter defines whether committed or set values are to be returned.  
Let's see how you can use it.

We will use 2 threads measuring temperature independently in different parts of the car: somewhere in the cabin (`getCabinTemperature`) and on the conditioner unit (`getConditionerTemperature`). After some period of time the average measured value will be written to the database.

**PeekPersistedExample.java: measureCarTemperature**

```

01 | private static void measureCarTemperature() {
02 |     setObjects();
03 |     ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
04 |     try {
05 |         ObjectSet result = container.query(Car.class);
06 |         if (result.size() > 0) {
07 |             Car car = (Car) result.get(0);
08 |             Car car1 = (Car) container.ext().peekPersisted(car,
09 |                         5, true);
10 |             Change1 ch1 = new Change1();

```

```

11 |     ch1. init(car1);
12 |     Car car2 = (Car) container. ext(). peekPersisted(car,
13 |             5, true);
14 |     Change2 ch2 = new Change2();
15 |     ch2. init(car2);
16 |     try {
17 |         Thread. sleep(300);
18 |     } catch (InterruptedException e) {
19 |     }
20 |     // We can work on the database object at the same time
21 |     car. setModel ("BMW M3Coupe");
22 |     container. set(car);
23 |     ch1. stop();
24 |     ch2. stop();
25 |     System.out. println("car1 saved to the database: "
26 |             + container. ext(). isStored(car1));
27 |     System.out. println("car2 saved to the database: "
28 |             + container. ext(). isStored(car1));
29 |     int temperature = (int) ((car1. getTemperature() + car2
30 |             . getTemperature()) / 2);
31 |     car. setTemperature(temperature);
32 |     container. set(car);
33 | }
34 | } finally {
35 |     container. close();
36 | }
37 | checkCar();
38 |
}

```

peekPersisted method gives you an easy way to work with database objects' clones. Remember that these clones are totally disconnected from the database. If you will try to save such object:

Java: ObjectContainer.set(peekPersistedObject)

you will get a new object in the database.

This revision (8) was last Modified 2006-11-13T16:24:04 by Tetyana.

# IsActive

ExtObjectContainer.isActive method provides you with means to define if the object is active.

UtilityExample.java: checkActive

```
01 ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰⑱⑲
02 |    private static void checkActive() {
03 |        storeSensorPanel();
04 |        ObjectContainer container = Db4o.openFile(DB4O_FILE_NAME);
05 |        try {
06 |            container.ext().configure().activationDepth(2);
07 |            System.out
08 |                .println("Object container activation depth = 2");
09 |            ObjectSet result = container.get(new SensorPanel(1));
10 |            SensorPanel sensor = (SensorPanel) result.get(0);
11 |            SensorPanel next = sensor.next;
12 |            while (next != null) {
13 |                System.out.println("Object " + next + " is active: "
14 |                    + container.ext().isActive(next));
15 |                next = next.next;
16 |            }
17 |        } finally {
18 |            container.close();
19 |        }
20 |    }
```

This method can be useful in applications with deep object hierarchy if you prefer to use manual activation.

This revision (6) was last Modified 2006-11-13T16:14:28 by Tetyana.

# IsStored

ExtObjectContainer#isStored helps you to define if the object is stored in the database. The following example shows how to use it:

UtilityExample.java: checkStored

```
01  private static void checkStored() {  
02 |     // create a linked list with length 10  
03 |     SensorPanel list = new SensorPanel().createList(10);  
04 |     new File(DB40_FILE_NAME).delete();  
05 |     ObjectContainer container = Db4o.openFile(DB40_FILE_NAME);  
06  [|     try {  
07 |         // store all elements with one statement, since all  
08 |         // elements are new  
09 |         container.set(list);  
10 |         Object sensor = (Object) list.sensor;  
11 |         SensorPanel sp5 = list.next.next.next.next;  
12 |         System.out.println("Root element " + list + " isStored: "  
13 |             + container.ext().isStored(list));  
14 |         System.out.println("Simple type " + sensor  
15 |             + " isStored: "  
16 |             + container.ext().isStored(sensor));  
17 |         System.out.println("Descend element " + sp5  
18 |             + " isStored: " + container.ext().isStored(sp5));  
19 |         container.delete(list);  
20 |         System.out.println("Root element " + list + " isStored: "  
21 |             + container.ext().isStored(list));  
22  [|     } finally {  
23 |         container.close();  
24 |     }  
25 | }
```

IsStored

This revision (7) was last Modified 2006-11-13T16:15:56 by Tetyana.

# Descend

ExtObjectContainer#descend method allows you to navigate from a persistent object to it's members without activating or instantiating intermediate objects.

UtilityExample.java: testDescend

```

01  private static void testDescend() {
02      storeSensorPanel();
03      Configuration configuration = Db4o.newConfiguration();
04      configuration.activationDepth(1);
05      ObjectContainer container = Db4o.openFile(configuration, DB4O_FILE_NAME);
06      try {
07          System.out
08              .println("Object container activation depth = 1");
09          ObjectSet result = container.get(new SensorPanel(1));
10          SensorPanel spParent = (SensorPanel) result.get(0);
11          SensorPanel spDescend = (SensorPanel) container.next()
12              .descend(
13                  (Object) spParent,
14                  new String[] { "next", "next", "next",
15                      "next", "next" });
16          container.activate(spDescend, 5);
17          System.out.println(spDescend);
18      } finally {
19          container.close();
20      }
21  }
```

[/filter]

[filter=cs]

UtilityExample.cs: TestDescend

```

01  public static void TestDescend()
02  {
03      StoreSensorPanel();
04      IConfiguration configuration = Db4oFactory.NewConfiguration();
05      configuration.ActivationDepth(1);
06      IObjectContainer db = Db4oFactory.OpenFile(configuration, Db4oFileName);
07      try
```

```
08 | {  
09 |     System.Console.WriteLine("Object container activation depth = 1");  
10 |     IObjectSet result = db.Get(new SensorPanel(1));  
11 |     SensorPanel spParent = (SensorPanel)result[0];  
12 |     SensorPanel spDescend = (SensorPanel)db.Ext().Descend((Object)spParent, new String[] {"_next", "_next", "_next", "_next", "_next"});  
13 |     db.Ext().Activate(spDescend, 5);  
14 |     System.Console.WriteLine(spDescend);  
15 | }  
16 | finally  
17 | {  
18 |     db.Close();  
19 | }  
20 | }
```

Navigating in this way can save you resources on activating only the objects you really need.

This revision (18) was last Modified 2006-11-13T16:09:14 by Tetyana.

# Usage Pitfalls

This topic set contains a collection of the most-common db4o usage pitfalls.

More Reading:

- [Equality Comparison](#)
- [Reference Cache In Client-Server Mode](#)
- [Storing BigDecimal](#)
- [Storing MarshalByRef Objects](#)
- [Storing Timestamp And Date Values](#)
- [Transparent Activation](#)
- [Transparent Persistence](#)
- [Accessing Persistent Classes From Different .NET Applications](#)
- [Activation Depth](#)
- [Classloader And Generic Classes](#)
- [Client-Server Timeouts](#)
- [Dangerous Practices](#)
- [Update Depth](#)
- [Working With Large Amounts Of Data](#)

This revision (2) was last Modified 2007-05-02T15:42:12 by Tetyana.

# Equality Comparison

Db4o uses reference cache for quick access to persistent objects. Each persistent object is guaranteed to have only one instance in the object reference cache independently of whether it was saved or retrieved. You can retrieve the same object several times with different querying methods, but you will still get references to the same object, so that `ref(1) == ref(2) == ... == ref(n)`.

In the same time it means that 2 objects, for example, one created in the runtime and another retrieved from the database, with the same data (field values) won't be equal for db4o.

There are 2 ways to compare db4o objects by data:

- using [QBE](#);
- implementing a suitable `equals` method.

Let's save an object to the database and try the above mentioned methods in practice.

## EqualityExample.java: storePilot

```

01 ①private static void storePilot() {
02 |    ObjectContainer container = database();
03 |    if (container != null) {
04 |        try {
05 |            Pilot pilot = new Pilot("Kimi Rai kkonnen", 100);
06 |            container.set(pilot);
07 |        } catch (Exception ex) {
08 |            System.out.println("System Exception: " + ex.getMessage());
09 |        } finally {
10 |            closeDatabase();
11 |        }
12 |    }
13 }
```

More Reading:

- [QBE](#)

- [Using Equals](#)

This revision (4) was last Modified 2008-01-17T18:26:14 by Tetyana.

# QBE

If we have a prototype and want to find out if there is an object in the database with the same field values, we can simply use QBE:

EqualityExample.java: retrieveEqual

```

01  private static void retrieveEqual() {
02 |     ObjectContainer container = database();
03 |     if (container != null) {
04 |         try {
05 |             ObjectSet result = container.get(new Pilot("Kimi Räikkönen", 100));
06 |             if (result.size() > 0){
07 |                 System.out.println("Found equal object: " + result.next().toString());
08 |             } else {
09 |                 System.out.println("No equal object exist in the database");
10 |             }
11 |         } catch (Exception ex) {
12 |             System.out.println("System Exception: " + ex.getMessage());
13 |         } finally {
14 |             closeDatabase();
15 |         }
16 |     }
17 }
```

This method allows to combine retrieval and comparing in one operation.

This revision (1) was last Modified 2007-09-09T16:06:01 by Tetyana.

# Using Equals

In some cases you can't use QBE as a retrieval method. In these cases you must override the object's `equals` method to allow you to compare objects data. For example:

Pilot.java: `equals`

```
1 public boolean equals(Pilot p) {  
2 |     return name.equals(p.getName()) && points == p.getPoints();  
3 | }
```

Note, that you must implement `hashCode/GetHashCode` method, when you implement `equals`:

Pilot.java: `hashCode`

```
1 public int hashCode() {  
2 |     return name.hashCode() ^ points;  
3 | }
```

Now we can use the `equals` method to compare an object from the database to an object prototype:

EqualityExample.java: `testEquality`

```
01 private static void testEquality() {  
02 |     ObjectContainer container = database();  
03 |     if (container != null) {  
04 |         try {  
05 |             ObjectSet<Pilot> result = container.query(new Predicate<Pilot>() {  
06 |                 public boolean match(Pilot pilot) {  
07 |                     return pilot.getName().equals("Kimi Raikkonen") &&  
08 |                         pilot.getPoints() == 100;  
09 |                 }  
10 |             });  
11 |             Pilot obj = (Pilot)result.next();  
12 |             Pilot pilot = new Pilot("Kimi Raikkonen", 100);
```

```
13 |         String equality = obj.equals(pilot) ? "equal" : "not equal";
14 |         System.out.println("Pilots are " + equality);
15 |     } catch (Exception ex) {
16 |         System.out.println("System Exception: " + ex.getMessage());
17 |     } finally {
18 |         closeDatabase();
19 |     }
20 | }
21 }
```

This revision (1) was last Modified 2007-09-09T16:07:18 by Tetyana.

# Reference Cache In Client-Server Mode

Db4o uses object [reference cache](#) for easy access to persistent objects during one transaction. In client/server mode each client has its own reference cache, which helps to achieve good performance. However it gets complicated, when different clients work on the same object, as this object's cached value is used on each side. It means, that even when the operations go serially, the object's value won't be updated serially unless it is refreshed before each update.

The following example demonstrates this behaviour.

InconsistentGraphExample.java: main

```
1 public static void main(String[] args) throws IOException {
2     new InconsistentGraphExample().run();
3 }
```

InconsistentGraphExample.java: run

```
01 public void run() throws IOException, DatabaseFileLockedException {
02     new File(DB40_FILE_NAME).delete();
03     ObjectServer server = Db4o.openServer(DB40_FILE_NAME, PORT);
04     try {
05         server.grantAccess(USER, PASSWORD);
06
07         ObjectContainer client1 = server.openClient();
08         ObjectContainer client2 = server.openClient();
09
10        if (client1 != null && client2 != null) {
11            try {
12                // wait for the operations to finish
13                waitForCompletion();
14
15                // save pilot with client1
16                Car client1Car = new Car("Ferrari", 2006, new Pilot(
17                    "Schumacher"));
18                client1.set(client1Car);
19                client1.commit();
20                System.out.println("Client1 version initially: " + client1Car);
21                waitForCompletion();
22
23                // retrieve the same pilot with client2
24                Car client2Car = (Car) client2.query(Car.class).next();
25                System.out.println("Client2 version initially: " + client2Car);
```

```

26 |
27 |     // delete the pilot with client1
28 |     Pilot client1Pilot = (Pilot)client1.query(Pilot.class).next();
29 |     client1.delete(client1Pilot);
30 |     // modify the car, add and link a new pilot with client1
31 |     client1Car.setModel(2007);
32 |     client1Car.setPilot(new Pilot("Hakkinen"));
33 |     client1.set(client1Car);
34 |     client1.commit();
35 |
36 |     waitForCompletion();
37 |     client1Car = (Car) client1.query(Car.class).next();
38 |     System.out.println("Client1 version after update: " + client1Car);
39 |
40 |
41 |     System.out.println();
42 |     System.out.println("client2Car still holds the old object graph in its reference
cache");
43 |     client2Car = (Car) client2.query(Car.class).next();
44 |     System.out.println("Client2 version after update: " + client2Car);
45 |     ObjectSet result = client2.query(Pilot.class);
46 |     System.out.println("Though the new Pilot is retrieved by a new query: ");
47 |     listResult(result);
48 |
49 |     waitForCompletion();
50 | } catch (Exception ex) {
51 |     ex.printStackTrace();
52 | } finally {
53 |     closeClient(client1);
54 |     closeClient(client2);
55 | }
56 | }
57 | } catch (Exception ex) {
58 |     ex.printStackTrace();
59 | } finally {
60 |     server.close();
61 | }
62 | }
```

In order to make the objects consistent on each client you must refresh them from the server when they get updated. This can be done by using [Committed Callbacks](#).

This revision (1) was last Modified 2007-09-15T19:10:18 by Tetyana.

# Storing BigDecimal

## Contents

- Problem
- Reason
- Solution

This topic applies to Java version only

## Problem

---

BigDecimal objects can't be stored by db4o with the default configuration:

```
Configuration configuration = Db4o.newConfiguration();
```

### BigDecimalExample.java: storeBigDecimal

```
01  public static void storeBigDecimal(Configuration configuration) {
02 |     new File(DB4O_FILE_NAME).delete();
03 |     ObjectContainer container = database(configuration);
04 |     if (container != null) {
05 |         try {
06 |             BigDecimal d = new BigDecimal("-918.099995e-15");
07 |             System.out.println("BigDecimal value: " + d);
08 |
09 |             container.set(d);
10 |
11 |             container.commit();
12 |         } catch (Exception ex) {
13 |             System.out.println("Exception: " + ex.toString());
14 |         } finally {
15 |             closeDatabase();
16 |         }
17 |     }
}
```

18 }

## BigDecimalExample.java: retrieveBigDecimal

```

01 public static void retrieveBigDecimal(Configuration configuration) {
02     ObjectContainer container = database(configuration);
03     if (container != null) {
04         try {
05             ObjectSet result = container.query(BigDecimal.class);
06             listResult(result);
07         }
08         catch (Exception ex) {
09             System.out.println("Exception: " + ex.toString());
10         } finally {
11             closeDatabase();
12         }
13     }
14 }
```

## Reason

`java.math.BigDecimal` class contains some transient fields, which prevents it from being stored correctly.

## Solution

The problem can be solved by applying the following configuration to the object container:

## BigDecimalExample.java: configure

```

1 private static Configuration configure() {
2     Configuration configuration = Db4o.newConfiguration();
3     //configuration.objectClass(BigDecimal.class).callConstructor(true);
4     configuration.objectClass(BigDecimal.class).storeTransientFields(true);
5     return configuration;
```

6 L }

Try to run `storeBigDecimal` and `retrieveBigDecimal` methods using the new configuration.

This revision (8) was last Modified 2007-08-29T13:19:22 by Tetyana.

# Storing MarshalByRef Objects

## Contents

- [Problem](#)
- [Reason](#)
- [Solution](#)

This topic applies to .NET version only

## Problem

---

MarshalByRef objects from .NET Remoting can not be stored by db4o.

## Reason

---

MarshalByRef objects are not really objects. They are proxy objects to instances, which live on another machine. There are 2 cases to distinguish:

1. if the object instance is local, then the object is storables without an exception unless a remote client has a lifetime lease on the object instance;
2. if the object instance is remote, then you're actually dealing with a DCOM proxy, which cannot be safely stored in db4o, since the remote object lifetime would expire as soon as the object was garbage collected.

## Solution

---

1. Use marshal by value technology for the persistent object. In this case the object implements ISerializable interface and can be stored by db4o without any problems. The object is created on the client and is passed to the server using MarshalByRef proxy.

In the example, TestValue class should be stored to db4o. TestValue implements ISerializable. In order to make TestValue class available to the server and to the client it is declared in a separate class library RemotingClasses.

TestValueServer class extends MarshalByRefObject and plays the role of a carrier between the client and the server. TestValue instance is created on the client and passed to the TestValueServer, which runs as a service on the server. TestValueServer then stores TestValue object to the database and retrieves its value on request.

Download the example solution: [C#](#), [VB](#)

2. If using MarshalByRef object is mandatory for you, use the following configuration:

```
configuration.ObjectClass("System.Runtime.Remoting.ServerIdentity, mscorelbin").  
Translate(new TTransient());  
configuration.ObjectClass("System.Threading.TimerCallback, mscorelbin").  
Translate(new TTransient());
```

The TTransient translator will prevent instances of ServerIdentity and TimerCallback from being stored within a db4o database. These classes are protected internal classes within the .NET Framework. When retrieving your MarshalByRef objects from db4o, you will need to re-marshal them.

In the following example Test object extends MarshalByRefObject. Test object lives on the server and can store itself to db4o when a request is received from the client. The configuration above is used to make it storable with db4o.

Download the example solution: [C#](#), [VB](#).

Note: in many cases using [db4o client-server](#) version can be a better option for a remote persistence.

This revision (7) was last Modified 2007-05-23T09:52:55 by Tetyana.

# Storing Timestamp And Date Values

## Contents

- Problem
- Reason
- Solution

This topic applies to Java version only

## Problem

---

If you've been working with `java.sql.Date`, `java.sql.Timestamp` or other classes extending `java.util.Date`, you should know that they cannot be stored to db4o by default. To see the problem try to run the following methods, using the default configuration:

```
Configuration configuration = Db4o.newConfiguration();
```

### TimestampExample.java: storeTimestamp

```
01 | private static void storeTimestamp(Configuration configuration) {
02 |     new File(DB4O_FILE_NAME).delete();
03 |
04 |     ObjectContainer container = database(configuration);
05 |     if (container != null) {
06 |         try {
07 |             Timestamp timestamp = new Timestamp(System.currentTimeMillis());
08 |             System.out.println("timestamp: " + timestamp);
09 |
10 |             container.set(timestamp);
11 |
12 |             container.commit();
13 |         } catch (Exception ex) {
14 |             System.out.println("Exception: " + ex.toString());
15 |         } finally {
```

```

16 |         closeDatabase();
17 |     }
18 | }
19 L }
```

## TimestampExample.java: retrieveTimestamp

```

01 | private static void retrieveTimestamp(Configuration configuration) {
02 |     ObjectContainer container = database(configuration);
03 |     if (container != null) {
04 |         try {
05 |             ObjectSet result = container.query(Timestamp.class);
06 |             listResult(result);
07 |
08 |         } catch (Exception ex) {
09 |             System.out.println("Exception: " + ex.toString());
10 |         } finally {
11 |             closeDatabase();
12 |         }
13 |     }
14 L }
```

## Reason

`java.sql.Date` or `java.sql.Timestamp` classes extend `java.util.Date` class, which has a transient field, preventing it from being stored. (Note, that using `java.util.Date` as a field in your class does not raise any problem, as this class gets special treatment by db4o.)

## Solution

The problem can be solved by applying the following configuration to the object container:

## TimestampExample.java: configure

```
1 private static Configuration configure() {
2 |   Configuration configuration = Db4o.newConfiguration();
3 |   configuration.objectClass(Date.class).storeTransientFields(true);
4 |   return configuration;
5 }
```

Try to run storeTimestamp and retrieveTimestamp methods using the new configuration.

The same solution can be applied to java.sql.Date and java.util.Date classes.

This revision (7) was last Modified 2007-08-29T13:26:49 by Tetyana.

# Transparent Activation

## Contents

- Not Activatable Objects
  - Problem
  - Solution
- Collections Activation
  - Problem
  - Solution
- Migrating Between Databases
  - Problem
  - Solution
- Instrumentation Limitations
  - Problem
  - Solution
- Debugging Instrumented Classes
  - Problem
  - Solution

[Transparent Activation](#) is a powerful feature that can make development much faster, easier and error-proof. But as any power it can lead to trouble if used in a wrong way. The aim of this chapter is to point you out to typical pitfalls, which can lead to unexpected and undesired results.

## Not Activatable Objects

---

### Problem

---

When TA is enabled Activatable objects are activated transparently on request, whereas not Activatable objects are fully activated. This is done to keep consistency in persistent objects behaviour. However, there is a potential danger of activating too many unnecessary objects and wasting system resources. You will experience lower performance and in the worst case you can end up with OutOfMemory error.

### Solution

---

Make all persistent objects Activatable. This can be done through using load time (Java only) or build time Transparent Activation Enhancement. For more information see [Transparent Activation Framework](#) documentation.

## Collections Activation

---

## Problem

---

Current implementation of TA Framework does not support lazy activation of collection members, i.e. the whole collection will be activated as one object on the first request. However, this only applies to collection object itself, i.e. Activatable members of the collection will be activated in a transparent way.

## Solution

---

Use db4o proprietary collections: com.db4o.collectctions package in Java and Db4objects.Db4o.Collections in .NET

# Migrating Between Databases

---

## Problem

---

Transparent Activation is implemented through Activatable/IActivatable interface, which binds an object to the current object container. In a case when an object is stored to more than one object container, this logic won't work, as only one binding (activator) is allowed per object.

## Solution

---

To allow correct behavior of the object between databases, the object should be unbinded before being stored to the next database. This can be done with the following code:

Java:

```
myObject.bind(null);
```

For more information see an [example](#).

# Instrumentation Limitations

---

## Problem

---

For Java instrumentation instrumenting classloader must know the classes to instrument, i.e. all application classes should be on the classpath.

## Solution

---

Make sure that all classes to be instrumented are available through the classpath

## Debugging Instrumented Classes

---

### Problem

Debugging instrumented classes may not work 100% correct both on Java and .NET. Some of the observed problems:

- In Visual Studio setting a breakpoint in the area effected by instrumentation seems to be not possible
- In Eclipse instrumented bytecode might be not displayed correctly in the debug mode

### Solution

You should be able to debug normally anywhere around instrumented bytecode. If you still think that the problem occurs in the instrumented area, please submit a bug report to [db4o Jira](#).

This revision (6) was last Modified 2008-01-12T11:20:03 by Tetyana.

# Migrating Between Databases

Transparent activation and persistence functionality depends on an association between an object and an object container, which is created when an activator is bound to the object. Each object allows only one activator. Typically this limitation won't show up, however there is a valid use case for it:

- 1) suppose you need to copy one or more objects from one object container to another;
- 2) you will retrieve the object(s) from the first object container using any suitable query syntax;
- 3) optionally you can close the first object container;
- 4) you will now save the object to the second object container.

If both object containers were using transparent activation or persistence - the 4-th step will throw an exception. Let's look at the case in more detail. Typical activatable class contains an `activator` field. When transparent activation functionality is used for the first time an object container activator will be bound to the object:

## SensorPanelTA.java: bind

```

01 /*Bind the class to the specified object container, create the activator*/
02 public void bind(Activator activator) {
03     if (_activator == activator) {
04         return;
05     }
06     if (activator != null && _activator != null) {
07         throw new IllegalStateException();
08     }
09     _activator = activator;
10 }

```

If `bind` method will be re-called with the same object container, activator parameter will always be the same. However, if another object container tries to bind the object (in our case with the `store` call) activator parameter will be different, which will cause an exception. (Exception will be thrown even if the first object container is already closed, as activator object still exists in the memory.) This behaviour is illustrated with the following example ([SensorPanelTA](#) class from Transparent Activation chapter is used):

## TExample.java: testSwitchDatabases

```

01 private static void testSwitchDatabases() {
02     storeSensorPanel();
03

```

```

04 |     ObjectContainer firstDb = Db4o.openFile(configureTA(), FIRST_DB_NAME);
05 |     ObjectContainer secondDb = Db4o.openFile(configureTA(), SECOND_DB_NAME);
06 |     try {
07 |         ObjectSet result = firstDb.queryByExample(new SensorPanelTA(1));
08 |         if (result.size() > 0) {
09 |             SensorPanelTA sensor = (SensorPanelTA) result.get(0);
10 |             firstDb.close();
11 |             // Migrating an object from the first database
12 |             // into a second database
13 |             secondDb.store(sensor);
14 |         }
15 |     } finally {
16 |         firstDb.close();
17 |         secondDb.close();
18 |     }
19 |

```

The solution to this problem is simple: activator should be unbound from the object:

Java:

```
sensor.bind(null);
```

Note, that the object will quit being activatable for the first object container. The following example shows the described behaviour:

#### TAExample.java: testSwitchDatabasesFixed

```

01 |     private static void testSwitchDatabasesFixed() {
02 |         storeSensorPanel();
03 |
04 |         ObjectContainer firstDb = Db4o.openFile(configureTA(), FIRST_DB_NAME);
05 |         ObjectContainer secondDb = Db4o.openFile(configureTA(), SECOND_DB_NAME);
06 |         try {
07 |             ObjectSet result = firstDb.queryByExample(new SensorPanelTA(1));
08 |             if (result.size() > 0) {
09 |                 SensorPanelTA sensor = (SensorPanelTA) result.get(0);
10 |                 // Unbind the object from the first database
11 |                 sensor.bind(null);

```

```
12 |         // Migrating the object into the second database
13 |         secondDb.store(sensor);
14 |
15 |
16 |         System.out.println("Retrieving previous query results from "
17 |             + FIRST_DB_NAME + ":");
18 |         SensorPanelTA next = sensor.getNext();
19 |         while (next != null) {
20 |             System.out.println(next);
21 |             next = next.getNext();
22 |         }
23 |
24 |         System.out.println("Retrieving previous query results from "
25 |             + FIRST_DB_NAME + " with manual activation:");
26 |         firstDb.activate(sensor, Integer.MAX_VALUE);
27 |         next = sensor.getNext();
28 |         while (next != null) {
29 |             System.out.println(next);
30 |             next = next.getNext();
31 |         }
32 |
33 |         System.out.println("Retrieving sensorPanel from " + SECOND_DB_NAME + ":");
34 |         result = secondDb.queryByExample(new SensorPanelTA(1));
35 |         next = sensor.getNext();
36 |         while (next != null) {
37 |             System.out.println(next);
38 |             next = next.getNext();
39 |         }
40 |     }
41 | } finally {
42 |     firstDb.close();
43 |     secondDb.close();
44 | }
45 | }
```

This revision (4) was last Modified 2008-01-12T19:40:37 by Tetyana.

# Transparent Persistence

Transparent Persistence is closely coupled with Transparent Activation, therefore the same [pitfalls](#) apply here. However there are some additional catches:

- [Object Clone](#)
- [Rollback Strategies](#)

This revision (5) was last Modified 2008-03-02T09:13:57 by Tetyana.

# Object Clone

Platform implementations of #clone is not compatible with TP.

Both java and .NET object implementations provide #clone method for default objects, which is enabled by implementing `Cloneable`/`ICloneable` interface. This implementation is a shallow clone, i.e. only the top-level object fields are duplicated, all the referenced(children) objects are only copied as references to the same object in the parent clone. But how it affects Transparent Persistence?

If you remember [Transparent Persistence Implementation](#) you must know that a special `Activator` field is used to bind an object to the object container. Consequently, the default clone will copy this `Activatable` field to the object's duplicate, which will produce ambiguity as the object container won't know which object should be activated for write.

Let's look how it will affect db4o in practice. We will use a usual [Car](#) class and make it cloneable. Use the following code to store a car object and it's clone:

TPCloneExample.java: storeCar

```

01  private static void storeCar() throws CloneNotSupportedException {
02 |     new File(DB40_FILE_NAME).delete();
03 |     ObjectContainer container = database(Db4o.newConfiguration());
04 |     if (container != null) {
05 |         try {
06 |             // create a car
07 |             Car car = new Car("BMW", new Pilot("Rubens Barrichello"));
08 |             container.store(car);
09 |             // clone
10 |             Car car1 = (Car)car.clone();
11 |             container.store(car1);
12 |         } finally {
13 |             closeDatabase();
14 |         }
15 |     }
16 |

```

So it works for the first store, but what if we will clone an object retrieved from the database?

TPCloneExample.java: testClone

```

01  private static void testClone() throws CloneNotSupportedException{
02 |     storeCar();
03 |     Configuration configuration = configureTP();
04 |
05 |     ObjectContainer container = database(configuration);
06 |     if (container != null) {
07 |         try {
08 |             ObjectSet result = container.queryByExample(new Car(null, null));
09 |             listResult(result);
10 |             Car car = null;
11 |             Car car1 = null;
12 |             if (result.size() > 0)
13 |             {
14 |                 car = (Car)result.get(0);
15 |                 System.out.println("Retrieved car: " + car);
16 |                 car1 = (Car)car.clone();
17 |                 System.out.println("Storing cloned car: " + car1);
18 |                 container.store(car1);
19 |             }
20 |         } finally {
21 |             closeDatabase();
22 |         }
23 |     }
24 | }
```

The code above throws an exception when the cloned object is being bound to the object container. Luckily we can easily fix it by overriding #clone method and setting activator to null:

**Car.java: Clone**

```
1 public Object clone() throws CloneNotSupportedException {  
2     Car test = (Car)super.clone();  
3     test._activator = null;  
4     return test;  
5 }
```

This revision (2) was last Modified 2008-03-02T09:14:34 by Tetyana.

# Rollback Strategies

[Transparent Persistence](#) makes development faster and more convenient. However without a clear understanding of what is happening under the hood you can easily get into trouble. Transparent Persistence is triggered by commit call and with rollback the changes are simply discarded. But is it really all that trivial?

More Reading:

- [Rollback And Cache](#)
- [Automatic Deactivation](#)
- [Car](#)
- [Id](#)
- [Pilot](#)

This revision (1) was last Modified 2008-03-02T09:32:01 by Tetyana.

# Rollback And Cache

Suppose we have [Car](#), [Pilot](#) and [Id](#) classes stored in the database. Car class is activatable, others are not. We will modify the car and rollback the transaction:

TPRollback.java: modifyAndRollback

```

01 ①private static void modifyAndRollback() {
02 |     ObjectContainer container = database(configureTP());
03 ③|     if (container != null) {
04 ④|         try {
05 |             // create a car
06 |             Car car = (Car) container.queryByExample(new Car(null, null))
07 |                 .get(0);
08 |             System.out.println("Initial car: " + car + "("
09 |                 + container.ext().getID(car) + ")");
10 |             car.setModel("Ferrari");
11 |             car.setPilot(new Pilot("Michael Schumacher", 123));
12 |             container.rollback();
13 |             System.out.println("Car after rollback: " + car + "("
14 |                 + container.ext().getID(car) + ")");
15 ⑤|         } finally {
16 |             closeDatabase();
17 |         }
18 |     }
19 }
```

If the transaction was going on normally (commit), we would have had the car modified in the database as it is supported by Transparent Persistence. However, as the transaction was rolled back - no modifications should be done to the database. The result that is printed to the screen is taken from the reference cache, so it will show modified objects. That is confusing and should be fixed:

TPRollback.java: modifyRollbackAndCheck

```

01 ①②private static void modifyRollbackAndCheck() {
02 |     ObjectContainer container = database(configureTP());
03 |     if (container != null) {
04 |         try {
05 |             // create a car
06 |             Car car = (Car) container.queryByExample(new Car(null, null))
07 |                 .get(0);
08 |             Pilot pilot = car.getPilot();
09 |             System.out.println("Initial car: " + car + "("
10 |                 + container.ext().getID(car) + ")");
11 |             System.out.println("Initial pilot: " + pilot + "("
12 |                 + container.ext().getID(pilot) + ")");
13 |             car.setModel("Ferrari");
14 |             car.changePilot("Michael Schumacher", 123);
15 |             container.rollback();
16 |             container.deactivate(car, Integer.MAX_VALUE);
17 |             System.out.println("Car after rollback: " + car + "("
18 |                 + container.ext().getID(car) + ")");
19 |             System.out.println("Pilot after rollback: " + pilot + "("
20 |                 + container.ext().getID(pilot) + ")");
21 |         } finally {
22 |             closeDatabase();
23 |         }
24 |     }
25 | }

```

Here we've added a deactivate call for the car object. This call is used to clear the reference cache and its action is reversed to activate.

We've used Integer.MAX\_VALUE/Int32.MaxValue to deactivate car fields to the maximum possible depth. Thus we can be sure that all the car fields will be re-read from the database again (no outdated values from the reference cache), but the trade-off is that all child objects will be deactivated and read from the database too. You can see it on Pilot object. This behaviour is preserved for both activatable and non-activatable objects.

This revision (1) was last Modified 2008-03-02T09:33:06 by Tetyana.

# Automatic Deactivation

The use of depth parameter in deactivate call from the [previous example](#) directly affects performance: the less is the depth the less objects will need to be re-read from the database and the better the performance will be. Ideally we only want to deactivate the objects that were changed in the rolled-back transaction. This can be done by providing a special class for db4o configuration. This class should implement RollbackStrategy/IRollbackStrategy interface and is configured as part of Transparent Persistence support:

TPRollback.java: rollbackDeactivateStrategy

```

1 [+] private static class RollbackDeactivateStrategy implements RollbackStrategy {
2 [+]     public void rollback( ObjectContainer container, Object obj ) {
3 |         container.ext().deactivate( obj );
4 |     }
5 L }
```

TPRollback.java: configureTPForRollback

```

1 [+] private static Configuration configureTPForRollback() {
2 |     Configuration configuration = Db4o.newConfiguration();
3 |     // add TP support and rollback strategy
4 |     configuration.add(new TransparentPersistenceSupport(
5 |         new RollbackDeactivateStrategy()));
6 |     return configuration;
7 L }
```

RollbackDeactivateStrategy#rollback method will be automatically called **once** per each **modified** object after the rollback. Thus you do not have to worry about deactivate depth anymore - all necessary deactivation will happen transparently preserving the best performance possible.

TPRollback.java: modifyWithRollbackStrategy

```

01 [+] private static void modifyWithRollbackStrategy() {
02 |     ObjectContainer container = database(configureTPForRollback());
03 [+]     if (container != null) {
04 [+]         try {
```

```
05 |         // create a car
06 |         Car car = (Car) container.queryByExample(new Car(null, null))
07 |             .get(0);
08 |         Pilot pilot = car.getPilot();
09 |         System.out.println("Initial car: " + car + "("
10 |             + container.ext().getID(car) + ")");
11 |         System.out.println("Initial pilot: " + pilot + "("
12 |             + container.ext().getID(pilot) + ")");
13 |         car.setModel("Ferrari");
14 |         car.changePilot("Michael Schumacher", 123);
15 |         container.rollback();
16 |         System.out.println("Car after rollback: " + car + "("
17 |             + container.ext().getID(car) + ")");
18 |         System.out.println("Pilot after rollback: " + pilot + "("
19 |             + container.ext().getID(pilot) + ")");
20 |     } finally {
21 |         closeDatabase();
22 |     }
23 | }
24 | }
```

Note, that RollbackDeactivateStrategy **only works for activatable** objects. To see the different you can comment out Activatable implementation in Id class (id value will be preserved in the cache).

This revision (1) was last Modified 2008-03-02T09:33:56 by Tetyana.

# Car

Car.java

```
01 /* Copyright (C) 2004 - 2008 db4o objects Inc. http://www.db4o.com */  
02 package com.db4odoc.tp.rollback;  
03  
04 import com.db4o.activation.ActivationPurpose;  
05 import com.db4o.activation.Activator;  
06 import com.db4o.ta.Activatable;  
07  
08 public class Car implements Activatable, Cloneable {  
09     private String model;  
10     private Pilot pilot;  
11     transient Activator _activator;  
12  
13     public Car(String model, Pilot pilot) {  
14         this.model = model;  
15         this.pilot = pilot;  
16     }  
17     // end Car  
18  
19     // Bind the class to an object container  
20     public void bind(Activator activator) {  
21         if (_activator == activator) {  
22             return;  
23         }  
24         if (activator != null && _activator != null) {  
25             throw new IllegalStateException();  
26         }  
27         _activator = activator;  
28     }
```

```
29 | // end bind
30 |
31 | // activate the object fields
32 |⊕| public void activate(ActivationPurpose purpose) {
33 |     if (_activator == null)
34 |         return;
35 |     _activator.activate(purpose);
36 | }
37 | // end activate
38 |
39 |
40 |⊕| public String getModel () {
41 |     activate(ActivationPurpose. READ);
42 |     return model;
43 | }
44 | // end getModel
45 |
46 |⊕| public void setModel (String model) {
47 |     activate(ActivationPurpose. WRITE);
48 |     this.model = model;
49 | }
50 | // end setModel
51 |
52 |⊕| public Pilot getPilot() {
53 |     activate(ActivationPurpose. READ);
54 |     return pilot;
55 | }
56 | // end getPilot
57 |
58 |⊕| public void setPilot(Pilot pilot) {
59 |     activate(ActivationPurpose. WRITE);
60 |     this.pilot = pilot;
61 | }
```

```
62 | // end setPilot
63 |
64 [+] public String toString() {
65 |     activate(ActivationPurpose.READ);
66 |     return model + "[" + pilot + "]";
67 | }
68 | // end toString
69 |
70 [+] public void changePilot(String name, int id) {
71 |     pilot.setName(name);
72 |     pilot.setId(id);
73 | }
74 |
75 }
```

This revision (1) was last Modified 2008-03-02T09:34:55 by Tetyana.

# Id

Id.java

```
01 /* Copyright (C) 2008 db4objects Inc. http://www.db4o.com */  
02  
03 package com.db4odoc.tp.rollback;  
04  
05 import com.db4o.activation.ActivationPurpose;  
06 import com.db4o.activation.Activator;  
07 import com.db4o.ta.Activatable;  
08  
09 public class Id implements Activatable {  
10     int number = 0;  
11  
12     transient Activator _activator;  
13  
14     public Id(int number){  
15         this.number = number;  
16     }  
17  
18     public void bind(Activator activator) {  
19         if (_activator == activator) {  
20             return;  
21         }  
22         if (activator != null && _activator != null) {  
23             throw new IllegalStateException();  
24         }  
25         _activator = activator;  
26     }  
27  
28     public void activate(ActivationPurpose purpose) {
```

```
29 |     if (_activator == null)
30 |         return;
31 |     _activator.activate(purpose);
32 | }
33 |
34 |
35 [⊕] public String toString(){
36 |     activate(ActivationPurpose.READ);
37 |     return String.valueOf(number);
38 | }
39 |
40 [⊕] public void change(int i) {
41 |     activate(ActivationPurpose.WRITE);
42 |     this.number = i;
43 | }
44 }
```

This revision (1) was last Modified 2008-03-02T09:36:09 by Tetyana.

# Pilot

Pilot.java

```
01/* Copyright (C) 2008 db4objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.tp.rollback;
04
05 import com.db4o.activation.ActivationPurpose;
06 import com.db4o.activation.Activator;
07 import com.db4o.ta.Activatable;
08
09public class Pilot implements Activatable {
10
11     private String name;
12     private Id id;
13
14     transient Activator _activator;
15     // Bind the class to an object container
16     public void bind(Activator activator) {
17         if (_activator == activator) {
18             return;
19         }
20         if (activator != null && _activator != null) {
21             throw new IllegalStateException();
22         }
23         _activator = activator;
24     }
25
26     // activate the object fields
27     public void activate(ActivationPurpose purpose) {
28         if (_activator == null)
```

```
29 |         return;
30 |         _activator.activate(purpose);
31 |     }
32 |
33 |
34 |     public Pilot(String name, int id) {
35 |         this.name = name;
36 |         this.id = new Id(id);
37 |     }
38 |
39 |     public String getName() {
40 |         activate(ActivationPurpose.READ);
41 |         return name;
42 |     }
43 |
44 |     public void setName(String name) {
45 |         activate(ActivationPurpose.WRITE);
46 |         this.name = name;
47 |     }
48 |
49 |     public String toString() {
50 |         activate(ActivationPurpose.READ);
51 |         return getName() + "[" + id + "]";
52 |     }
53 |
54 |     public void setId(int i) {
55 |         activate(ActivationPurpose.WRITE);
56 |         this.id.change(i);
57 |     }
58 | }
```

This revision (1) was last Modified 2008-03-02T09:35:36 by Tetyana.

# Car

Car.java

```
01 /* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */
02 package com.db4odoc.clone;
03
04 import com.db4o.activation.ActivationPurpose;
05 import com.db4o.activation.Activator;
06 import com.db4o.ta.Activatable;
07
08 public class Car implements Activatable, Cloneable {
09     private String model;
10    private Pilot pilot;
11    transient Activator _activator;
12
13     public Car(String model, Pilot pilot) {
14         this.model = model;
15         this.pilot = pilot;
16     }
17     // end Car
18
19     public Object clone() throws CloneNotSupportedException {
20         return super.clone();
21     }
22     // end clone
23
24     // Bind the class to an object container
25     public void bind(Activator activator) {
26         if (_activator == activator) {
27             return;
28         }
```

```
29 |         if (activator != null && _activator != null) {  
30 |             throw new IllegalStateException();  
31 |         }  
32 |         _activator = activator;  
33 |     }  
34 | // end bind  
35 |  
36 | // activate the object fields  
37 |         public void activate(ActivationPurpose purpose) {  
38 |             if (_activator == null)  
39 |                 return;  
40 |             _activator.activate(purpose);  
41 |         }  
42 | // end activate  
43 |  
44 |  
45 |         public String getModel() {  
46 |             activate(ActivationPurpose.READ);  
47 |             return model;  
48 |         }  
49 | // end getModel  
50 |  
51 |         public String toString() {  
52 |             activate(ActivationPurpose.READ);  
53 |             return model + "[" + pilot + "]";  
54 |         }  
55 | // end toString  
56 |  
57 | }
```

This revision (2) was last Modified 2008-01-29T14:03:27 by Tetyana.

# Accessing Persistent Classes From Different .NET Applications

## Contents

- [Problem](#)
- [Reason](#)
- [Solution](#)

This topic applies to .NET version only

## Problem

---

Accessing db4o database created and filled in with a .NET application or library from another .NET application or library shows an empty database.

## Reason

---

db4o class name format in db4o consists of the full class name and assembly name:

Namespace.ClassName, AssemblyName

Two different .NET applications (libraries) usually have different assembly names. If you do not use [aliasing](#), the class name will be appended with the current application assembly name.

## Solution

---

In order to access db4o persistent classes from different applications (libraries) you will need to use an [Alias](#). For example:

**Application1.exe ("Application1" assembly):**

```
objectContainer.Set(new MyClasses.Pilot("David Barrichello", 99))  
  
// internally the class is saved as "MyClasses.Pilot, Application1".
```

**Application2.exe ("Application2" assembly):**

For more information see [Class Name Format In .NET](#) and [Aliases](#).

This revision (7) was last Modified 2007-05-09T07:20:53 by Tetyana.

# Activation Depth

In order to work effectively with db4o you must understand the concept of [Activation](#). Activation controls the amount of referenced objects loaded into the memory. There are 2 main pitfalls that you must be aware about:

1. An object retrieved from the database is null.

This happens if the activation level is lower than needed. For example:

class Pilot has field Car:

```
Pilot {  
  
    Car car;  
  
}
```

and is saved to a database. Then `pilot` object us retrieved from the database with the activation depth is set to 0. In this case `pilot.car` will be equal to null and can be incorrectly interpreted.

2. Activation depth is set [globally](#) to a high value or is set to [cascadeOnActivate](#) for a heavily used object with a deep structure. This will result in a huge performance penalty and should be avoided.

The automatic solution of the Activation issues is provided by [Transparent Activation Framework](#). However, understanding of Activation is still important.

For more information on activation see:

- [Global Activation Settings](#)
- [Object-Specific Activation](#)
- [Transparent Activation Framework](#)
- [Activation strategies](#)

This revision (6) was last Modified 2007-11-05T16:30:34 by Tetyana.

# Classloader And Generic Classes

This topic applies to Java version only

db4o uses class information available from the classloader to store and recreate class objects. When a class definition is not available from the classloader db4o resolves to [Generic Objects](#), which represent the class information stored in object arrays. With this approach db4o is ready to function both with and without [class definitions available](#). However, the problem can appear when your application and db4o use different classloaders, because in this case db4o won't match objects in the database to their definitions in the runtime. In order to avoid this:

1. Make sure that your db4o lib is not in JRE or JDK lib folder. Libraries in these folders get a special classloader, which is unaware of your application classes. Instead put db4o library into any other suitable for you location and make it available to your application through CLASSPATH or using IDE provided methods.
2. If your application design does not guarantee that application classes and db4o will be loaded by the same classloader, use

```
Configuration#reflectWith(new JdkReflector(classLoader))
```

where classLoader is the classloader of your application classes.

The above-mentioned cases should be distinguished from a case when Java application uses a db4o database created from a .NET application. In this particular case .NET class definitions should be replaced by Java class definitions with the help of [Aliases](#).

This revision (2) was last Modified 2007-10-14T15:56:42 by Tetyana.

# Client-Server Timeouts

Every client/server application has to face a problem of network communications. Luckily modern protocols screen the end-application from all fixable problems; however there are still physical reasons that can't be fixed by a protocol: disconnections, power failures, crash of a system on the other end of communication channel etc. In these cases it is still the responsibility of the client-server application to exit the connection gracefully, releasing all resources and protecting data.

In order to achieve an efficient client/server communication and handling of connection problems the following requirements were defined for db4o:

- The connection should not be terminated when both client and server are still alive, even if either of the machines is running under heavy load.
- Whenever a client dies, peacefully or with a crash, the server should clean up all resources that were reserved for the client.
- Whenever a server goes offline, it should be possible for the client to detect that there is a problem.
- Since many clients may be connected at the same time, it makes sense to be careful with the resources the server reserves for each client.
- A client can be a very small machine, so it would be good if the client application can work with a single thread.

Unfortunately all the requirements are difficult to achieve for a cross-platform application, as Java and .NET sockets behave differently.

The initial db4o CS implementation used one-thread clients and connection was terminated when there was a timeout and a post-timeout check-up message could not get a response from the other side. However this approach failed for .NET implementation, as .NET sockets upon timeout continue to send and receive messages, but timeout settings are not respected anymore, which in fact leads to a very high CPU usage.

The next approach was to create a separate housekeeping thread on the server, which will send messages to the client regularly, thus checking if the client is still alive and in the same time giving the client information about the server. Unfortunately, the problem described above can occur to the housekeeping thread too.

The current approach tries to keep things as simple as possible: any connection is closed immediately upon a timeout. In order to prevent closing connections when there is no communication between client and server due to reasons different from connection problems a separate timer thread was created to send messages to the server at a regular basis. The server must reply to the thread immediately, if this does not happen the communication channel gets closed.

This approach works effectively for both client and server side, however there are two small downsides:

1. Clients are not single-threaded anymore

2. If an action on the server takes longer than the socket timeout, the connection will be closed.

In the latter case you can adjust the behaviour of db4o with the following configuration settings:

Java:

```
configuration.clientServer().timeoutServerSocket(int milliseconds)
```

```
configuration.clientServer().timeoutClientSocket(int milliseconds)
```

An easy rule of thumb:

- If you experience clients disconnecting, raise the timeout value.
- If you have a system where clients crash frequently or where the network is very unstable, lower the values, so resources for disconnected clients are freed faster.

This revision (2) was last Modified 2007-12-06T15:46:10 by Tetyana.

# Dangerous Practices

Db4o databases are well protected against corruption. However some specific configurations can make your database file vulnerable.

## 1. Configuration#lockDatabaseFile(false)

Java platforms before JDK1.4 do not prevent concurrent access to a file from different JVM. If database file locking is turned off on these platforms, concurrent write access to the same database file from different JVM sessions will corrupt the database file immediately. Do not use this setting unless your application logic guarantees that only one VM session can access your database file at a time. For more information see [No lock file thread](#).

## 2. Db4o.configure().flushFileBuffers(false)

In order to ensure ACID transaction db4o uses a [special strategy](#), which relies on the order of writes to the storage medium. On operating systems that use in-memory file caching, the OS cache may revert the order of writes to optimize file performance. db4o can enforce the correct order by flushing file buffers after every step of transaction commit. Turning this setting off puts you in potential danger of data corruption if a system or hardware failure occurs during commit.

## 3. The following refactorings are incompatible with db4o:

1. Adding classes within a class hierarchy or above a class hierarchy. Example:

### Original

---

```
class A  
class B extends A
```

### Refactored

---

```
class A  
class C extends A  
class B extends C
```

2. Removing a class from the top or within a class hierarchy. Example:

### Original

---

```
class A  
class B extends A  
class C extends B
```

### Refactored

-----  
class A  
class C extends A

3. Changing the type of a field to be an array or back. Example:

Original

-----  
class Foo {  
String bar;  
}

Refactored

-----  
class Foo {  
String [] bar;  
}

If you apply such a refactoring, you will not be able to read existing objects correctly.

More information on refactorings see [Refactoring and Schema Evolution](#)

This revision (1) was last Modified 2007-09-09T12:22:25 by Tetyana.

# Update Depth

db4o update behavior is regulated by [Update Depth](#). Understanding Update Depth will help you to improve performance and avoid unnecessary memory usage.

When Update Depth is set to a big value on objects with a deep reference hierarchy it will cause each update on the top-level object to trigger updates on the lower-level objects, which can impose a huge performance penalty.

The following settings should be used with a special care and only with a good reason to do so:

1. Global deep update depth setting:

Java:

```
configuration.updateDepth(Integer.MaxValue);
```

This setting causes ALL objects in the database to be updated to the lowest possible level on each update.

2. Cascade on update:

Java:

```
configuration.objectClass(Item.class).cascadeOnUpdateDepth(true);
```

This setting will cause an update to the lowest level for Item class only. This can be unnecessary and therefore undesired in order to save system resources.

For more detailed information, please, refer to [Update Depth](#).

This revision (2) was last Modified 2007-09-02T18:02:41 by Tetyana.

# Working With Large Amounts Of Data

## Contents

- [Size of Database Files](#)
- [Performance](#)

db4o is designed to manage large amounts of data. The following paragraphs highlight some information important for using db4o with large data.

## Size of Database Files

---

In the default setting, the maximum database file size is 2GB.

You can increase this value by [configuring the internal db4o block size](#):

Java:

```
configuration.blockSize(blockSize)
```

As a parameter you can specify any value between 1 and 127. The resulting maximum database file size will be a multiple of 2GB. A recommended setting for large database files is 8, since internal pointers have this length.

Using `blockSize` the maximum database file size will be 16GB.

The above method has to be called before an ObjectContainer is opened the first time. During the lifetime of an ObjectContainer the setting will have to stay the same. Since Defragment copies all objects to a new ObjectContainer, it can be used to change the `blockSize` of an existing database:

Java:

```
Defragment.defrag( "filename.db4o" )
```

## Performance

---

Navigation access times to objects and the performance of access by internal IDs remains constant, no matter how large database files are.

Query performance on unindexed objects drops linearly with an increasing number of objects per class.

Query performance on a large number of objects can be dramatically improved by [using indexes](#):

Java:

```
configuration.objectClass(Foo.class).objectField("bar").indexed(true);
```

db4o storage performance is very good. It is recommended to run your own benchmarks with large amounts of data to check the overall performance on your particular class hierarchy.

This revision (3) was last Modified 2007-09-09T17:00:02 by Tetyana.

# Client-Server

Client/Server mode is de-facto standard for any modern database. However there is a big difference between relational and object databases functionality in client-server mode.

With RDBMS everything is pretty straightforward: data is kept on a server and SQL commands generated on a client are used to operate them.

In db4o world SQL is an alien and querying syntax is based on class definitions. Therefore class libraries synchronization between client and server becomes essential.

More Reading:

- [Embedded](#)
- [Networked](#)
- [Native Queries In Client-Server Mode](#)
- [Server Without Persistent Classes Deployed](#)
- [Switching Database Files in CS Mode](#)
- [Semaphores](#)
- [Pluggable Sockets](#)
- [Remote Code Execution](#)
- [Concurrency Control](#)
- [Messaging](#)
- [Batch Mode](#)

This revision (9) was last Modified 2006-11-18T17:51:18 by Tetyana.

# Embedded

From the API side, there's no real difference between transactions executing concurrently within the same VM and transactions executed against a remote server. To use concurrent transactions within a single VM, we just open a db4o server on our database file, directing it to run on port 0, thereby declaring that no networking will take place.

ClientServerExample.java: accessLocalServer

```

01  private static void accessLocalServer() {
02 |     ObjectServer server=Db4o.openServer(DB40_FILE_NAME, 0);
03 |     try {
04 |         ObjectContainer client=server.openClient();
05 |         // Do something with this client, or open more clients
06 |         client.close();
07 |     }
08 |     finally {
09 |         server.close();
10 |     }
11 }
```

Again, we will delegate opening and closing the server to our environment to focus on client interactions.

ClientServerExample.java: queryLocalServer

```

1  private static void queryLocalServer(ObjectServer server) {
2 |     ObjectContainer client=server.openClient();
3 |     listResult(client.get(new Car(null)));
4 |     client.close();
5 }
```

The transaction level in db4o is *read committed*. However, each client container maintains its own weak reference cache of already known objects. To make all changes committed by other clients immediately, we have to explicitly refresh known objects from the server. We will delegate this task to a specialized version of our listResult() method.

ClientServerExample.java: listRefreshedResult

```

1  private static void listRefreshedResult(ObjectContainer container, ObjectSet result, int depth) {
2 |     System.out.println(result.size());
3 |     while(result.hasNext()) {
4 |         Object obj = result.next();
```

```

5 |         container. ext(). refresh(obj , depth);
6 |         System.out.println(obj );
7 |     }
8 | }
```

## ClientServerExample.java: demonstrateLocalReadCommitted

```

01 private static void demonstrateLocalReadCommitted( ObjectServer server) {
02 |     ObjectContainer client1=server.openClient();
03 |     ObjectContainer client2=server.openClient();
04 |     Pilot pilot=new Pilot("David Coulthard", 98);
05 |     ObjectSet result=client1.get(new Car("BMW"));
06 |     Car car=(Car)result.next();
07 |     car.setPilot(pilot);
08 |     client1.set(car);
09 |     listResult(client1.get(new Car(null)));
10 |     listResult(client2.get(new Car(null)));
11 |     client1.commit();
12 |     listResult(client1.get(Car.class));
13 |     listRefreshedResult(client2, client2.get(Car.class), 2);
14 |     client1.close();
15 |     client2.close();
16 | }
```

Simple rollbacks just work as you might expect now.

## ClientServerExample.java: demonstrateLocalRollback

```

01 private static void demonstrateLocalRollback( ObjectServer server) {
02 |     ObjectContainer client1=server.openClient();
03 |     ObjectContainer client2=server.openClient();
04 |     ObjectSet result=client1.get(new Car("BMW"));
05 |     Car car=(Car)result.next();
06 |     car.setPilot(new Pilot("Someone else", 0));
07 |     client1.set(car);
08 |     listResult(client1.get(new Car(null)));
09 |     listResult(client2.get(new Car(null)));
10 |     client1.rollback();
11 |     client1.ext().refresh(car, 2);
12 |     listResult(client1.get(new Car(null)));
```

```
13 |         listResult(client2.get(new Car(null)));
14 |         client1.close();
15 |         client2.close();
16 |     }
```

This revision (2) was last Modified 2006-11-14T11:20:16 by Tetyana.

# Networked

It's only a small step from an [embedded server](#) towards operating db4o over a TCP/IP network.

More Reading:

- [Network Server](#)
- [Out-of-band Signalling](#)
- [Simple db4o Server](#)

This revision (6) was last Modified 2007-08-19T13:56:36 by Tetyana.

# Network Server

In order to make the [embedded server](#) operate over a TCP/IP network, we just need to specify a port number greater than zero and set up one or more accounts for our client(s).

ClientServerExample.java: accessRemoteServer

```

01 private static void accessRemoteServer() throws IOException {
02     ObjectServer server=Db4o.openServer(DB40_FILE_NAME, PORT);
03     server.grantAccess(USER, PASSWORD);
04     try {
05         ObjectContainer client=Db4o.openClient("localhost", PORT, USER, PASSWORD);
06         // Do something with this client, or open more clients
07         client.close();
08     }
09     finally {
10         server.close();
11     }
12 }
```

The client connects providing host, port, user name and password.

ClientServerExample.java: queryRemoteServer

```

1 private static void queryRemoteServer(int port, String user, String password) throws
IOException {
2     ObjectContainer client=Db4o.openClient("localhost", port, user, password);
3     listResult(client.get(new Car(null)));
4     client.close();
5 }
```

Everything else is absolutely identical to the [local server examples](#).

ClientServerExample.java: demonstrateRemoteReadCommitted

```

01 private static void demonstrateRemoteReadCommitted(int port, String user, String password)
throws IOException {
02     ObjectContainer client1=Db4o.openClient("localhost", port, user, password);
03     ObjectContainer client2=Db4o.openClient("localhost", port, user, password);
04     Pilot pilot=new Pilot("Jenson Button", 97);
05     ObjectSet result=client1.get(new Car(null));
06     Car car=(Car)result.next();
```

```

07     car.setPilot(pilot);
08     client1.set(car);
09     listResult(client1.get(new Car(null)));
10     listResult(client2.get(new Car(null)));
11     client1.commit();
12     listResult(client1.get(new Car(null)));
13     listRefreshedResult(client2, client2.get(Car.class), 2);
14     client1.close();
15     client2.close();
16 }

```

## ClientServerExample.java: demonstrateRemoteRollback

```

01 private static void demonstrateRemoteRollback(int port, String user, String password)
throws IOException {
02     ObjectContainer client1=Db4o.openClient("localhost", port, user, password);
03     ObjectContainer client2=Db4o.openClient("localhost", port, user, password);
04     ObjectSet result=client1.get(new Car(null));
05     Car car=(Car)result.next();
06     car.setPilot(new Pilot("Someone else", 0));
07     client1.set(car);
08     listResult(client1.get(new Car(null)));
09     listResult(client2.get(new Car(null)));
10     client1.rollback();
11     client1.ext().refresh(car, 2);
12     listResult(client1.get(new Car(null)));
13     listResult(client2.get(new Car(null)));
14     client1.close();
15     client2.close();
16 }

```

This revision (11) was last Modified 2007-09-15T14:40:38 by Tetyana.

# Out-of-band Signalling

Sometimes a client needs to send a special message to a server in order to tell the server to do something. The server may need to be signalled to perform a defragment or it may need to be signalled to shut itself down gracefully.

This is configured by calling `setMessageRecipient()`, passing the object that will process client-initiated messages.

StartServer.java: runServer

```
01  /*
02 |   * opens the ObjectServer, and waits forever until close() is called
03 |   * or a StopServer message is being received.
04 |   */
05  public void runServer() {
06      synchronized(this) {
07          ObjectServer db4oServer = Db4o.openServer(FILE, PORT);
08          db4oServer.grantAccess(USER, PASS);
09
10          // Using the messaging functionality to redirect all
11          // messages to this.processMessage
12          db4oServer.ext().configure().clientServer().setMessageRecipient(this);
13
14          // to identify the thread in a debugger
15  Thread.currentThread().setName(this.getClass().getName());
16
17          // We only need low priority since the db4o server has
18          // its own thread.
19  Thread.currentThread().setPriority(Thread.MIN_PRIORITY);
20          try {
21              if(! stop){
22                  // wait forever for notify() from close()
23                  this.wait(Long.MAX_VALUE);
24              }
25          } catch (Exception e) {
```



```
26 |         e.printStackTrace();  
27 |     }  
28 |     db4oServer.close();  
29 | }  
30 }
```

This revision (4) was last Modified 2007-08-19T13:54:30 by Tetyana.

# Simple db4o Server

Let's implement a simple standalone db4o server with a special client that can tell the server to shut itself down gracefully on demand.

First, both the client and the server need some shared configuration information. We will provide this using an interface:

ServerConfiguration.java

```
01  * Copyright (C) 2007 db4o objects Inc. http://www.db4o.com */
02  package com.db4odoc.clientserver;
03
04  /**
05  * Configuration used for {@link StartServer} and {@link StopServer}.
06  */
07  public interface ServerConfiguration {
08  |
09  /**
10  * the host to be used.
11  * <br>If you want to run the client server examples on two computers,
12  * enter the computer name of the one that you want to use as server.
13  */
14  public String HOST = "localhost";
15
16  /**
17  * the database file to be used by the server.
18  */
19  public String FILE = "reference.db4o";
20
21  /**
22  * the port to be used by the server.
23  */
24  public int PORT = 0xdb40;
25
26  /**
27  * the user name for access control.
28  */
29  public String USER = "db4o";
30}
```

```

31  /***
32  * the password for access control.
33  */
34  public String PASS = "db4o";
35 }

```

Now we'll create the server:

### StartServer.java

```

01  /* Copyright (C) 2007 db4o objects Inc. http://www.db4o.com */
02  package com.db4odoc.clientserver;
03
04  import com.db4o.*;
05  import com.db4o.messaging.*;
06
07  /**
08  * starts a db4o server with the settings from {@link ServerConfiguration}.
09  * <br><br>This is a typical setup for a long running server.
10  * <br><br>The Server may be stopped from a remote location by running
11  * StopServer. The StartServer instance is used as a MessageRecipient and
12  * reacts to receiving an instance of a StopServer object.
13  * <br><br>Note that all user classes need to be present on the server
14  * side and that all possible Db4o.configure() calls to alter the db4o
15  * configuration need to be executed on the client and on the server.
16 */
17  public class StartServer
18  implements ServerConfiguration, MessageRecipient {
19
20  /**
21  * setting the value to true denotes that the server should be closed
22  */
23  private boolean stop = false;
24
25  /**
26  * starts a db4o server using the configuration from
27  * {@link ServerConfiguration}.
28  */
29  public static void main(String[] arguments) {

```

```
30 |     new StartServer().runServer();
31 | }
32 | // end main
33 |
34 | /**
35 | * opens the ObjectServer, and waits forever until close() is called
36 | * or a StopServer message is being received.
37 | */
38 | public void runServer(){
39 |     synchronized(this){
40 |         ObjectServer db4oServer = Db4o.openServer(FILE, PORT);
41 |         db4oServer.grantAccess(USER, PASS);
42 |
43 |         // Using the messaging functionality to redirect all
44 |         // messages to this.processMessage
45 |         db4oServer.ext().configure().clientServer().setMessageRecipient(this);
46 |
47 |         // to identify the thread in a debugger
48 |         Thread.currentThread().setName(this.getClass().getName());
49 |
50 |         // We only need low priority since the db4o server has
51 |         // its own thread.
52 |         Thread.currentThread().setPriority(Thread.MIN_PRIORITY);
53 |         try {
54 |             if(!stop){
55 |                 // wait forever for notify() from close()
56 |                 this.wait(Long.MAX_VALUE);
57 |             }
58 |             } catch (Exception e) {
59 |                 e.printStackTrace();
60 |             }
61 |             db4oServer.close();
62 |         }
63 |     }
64 | // end runServer
65 |
66 | /**
67 | * messaging callback
```

```

68 |     * @see com.db4o.messaging.MessageRecipient#processMessage(ObjectContainer, Object)
69 | */
70 |     public void processMessage(MessageContext context, Object message) {
71 |         if(message instanceof StopServer){
72 |             close();
73 |         }
74 |     }
75 | // end processMessage
76 |
77 |     /**
78 |      * closes this server.
79 |     */
80 |     public void close(){
81 |         synchronized(this){
82 |             stop = true;
83 |             this.notify();
84 |         }
85 |     }
86 | // end close
87 | }

```

And last but not least, the client that stops the server.

#### StopServer.java

```

01 /* Copyright (C) 2007 db4o objects Inc. http://www.db4o.com */
02 package com.db4odoc.clientserver;
03
04 import com.db4o.*;
05 import com.db4o.messaging.*;
06
07 /**
08 * stops the db4o Server started with {@link StartServer}.
09 * <br><br>This is done by opening a client connection
10 * to the server and by sending a StopServer object as
11 * a message. {@link StartServer} will react in it's
12 * processMessage method.
13 */
14 public class StopServer implements ServerConfiguration {

```

```
15 |
16 |     /**
17 |     * stops a db4o Server started with StartServer.
18 |     * @throws Exception
19 |     */
20 |     public static void main(String[] args) {
21 |         ObjectContainer objectContainer = null;
22 |         try {
23 |             // connect to the server
24 |             objectContainer = Db4o.openClient(HOST, PORT, USER, PASS);
25 |
26 |
27 |         } catch (Exception e) {
28 |             e.printStackTrace();
29 |         }
30 |
31 |         if(objectContainer != null){
32 |
33 |             // get the messageSender for the ObjectContainer
34 |             MessageSender messageSender = objectContainer.ext()
35 |                 .configure().clientServer().getMessageSender();
36 |
37 |             // send an instance of a StopServer object
38 |             messageSender.send(new StopServer());
39 |
40 |             // close the ObjectContainer
41 |             objectContainer.close();
42 |         }
43 |     }
44 |     // end main
45 | }
```

Keywords:

start remote instance

This revision (4) was last Modified 2007-12-12T00:45:12 by German Viscuso.

# Native Queries In Client-Server Mode

This topic applies to java version only

A quite subtle problem may occur if you're using Native Queries as anonymous (or just non-static) inner classes in Client/Server mode. Anonymous/non-static inner class instances carry a synthetic field referencing their outer class instance - this is just Java's way of implementing inner class access to fields or final method locals of the outer class without introducing any notion of inner classes at all at the bytecode level. If such a non-static inner class predicate cannot be converted to S.O.D.A. form on the client, it will be wrapped into an evaluation and passed to the server, introducing the same problem already mentioned in the [evaluation chapter](#): db4o will try to transfer the evaluation, using the standard platform serialization mechanism. If this fails, it will just try to pass it to the server via db4o marshalling. However, this may fail, too, for example if the outer class references any local db4o objects like ObjectContainer, etc., resulting in an ObjectNotStorableException.

So to be on the safe side with NQs in C/S mode, you should declare Predicates as top-level or static inner classes only. Alternatively, you could either make sure that the outer classes containing Predicate definitions could principally be persisted to db4o, too, and don't add significant overhead to the predicate (the appropriate value for 'significant' being your choice) or ensure during development that all predicates used actually can be optimized to S.O.D.A. form.

This revision (12) was last Modified 2007-05-09T07:30:32 by Tetyana.

# Server Without Persistent Classes Deployed

In an ordinary Client/Server setup persistent classes are present on both client and server side. However this condition is not mandatory and a server side can work without persistent classes deployed utilizing db4o [GenericReflector](#) functionality.

How it works?

When classes are unknown GenericReflector creates generic objects, which hold simulated "field values" in an array of objects. This is done automatically by db4o engine and does not require any additional settings from your side: you can save, retrieve and modify objects just as usual. An example of this functionality is db4o [Object Manager](#).

Unfortunately in a server without persistent classes mode there are still some limitations:

- evaluation queries won't work
- native queries will only work if they can be optimized
- multidimensional arrays can not be stored.

The following topics provide examples of a server without persistent classes usage. We will use a server from the [ClientServer](#) example, a separate project for a client and deploy persistent classes only on the client side. In .NET you will need to test the examples on separate machines as the assembly information is loaded in CLR and shared.

More Reading:

- [Saving Objects](#)
- [QBE](#)
- [SODA](#)
- [Native Queries](#)
- [Evaluations](#)
- [Multidimensional Arrays](#)

This revision (5) was last Modified 2007-05-09T07:40:14 by Tetyana.

# Saving Objects

Client.java: savePilots

```
01 private static void savePilots() throws IOException {
02 |     System.out.println("Saving Pilot objects without Pilot class on the server");
03 |     ObjectContainer oc = Db4o.openClient("localhost", 0xdb40, "db4o",
04 |                                         "db4o");
05 |     try {
06 |         for (int i = 0; i < COUNT; i++) {
07 |             oc.set(new Pilot("Pilot #" + i, i));
08 |         }
09 |
10 |     } finally {
11 |         oc.close();
12 |     }
13 }
```

You can check the saved objects using ObjectManager or the querying examples.

This revision (1) was last Modified 2006-12-30T08:29:17 by Tetyana.

# QBE

Client.java: getPilotsQBE

```
01  private static void getPilotsQBE() throws IOException {
02 |     System.out.println("Retrieving Pilot objects: QBE");
03 |     ObjectContainer oc = Db4o.openClient("localhost", 0xdb40, "db4o", "db4o");
04  [| try {
05 |     ObjectSet result = oc.get(new Pilot(null, 0));
06 |     listResult(result);
07  [| } finally {
08 |     oc.close();
09 | }
10 | }
```

This revision (2) was last Modified 2006-12-30T08:32:52 by Tetyana.

# SODA

Client.java: getPilotsSODA

```
01|private static void getPilotsSODA() throws IOException {  
02|    System.out.println("Retrieving Pilot objects: SODA");  
03|    ObjectContainer oc = Db4o.openClient("localhost", 0xdb40, "db4o", "db4o");  
04|    try {  
05|        Query query = oc.query();  
06|  
07|        query.constrain(Pilot.class);  
08|        query.descend("points").constrain(5);  
09|  
10|        ObjectSet result = query.execute();  
11|        listResult(result);  
12|    } finally {  
13|        oc.close();  
14|    }  
15|}
```

This revision (1) was last Modified 2006-12-30T08:34:50 by Tetyana.

# Native Queries

We have checked in the SODA topic that SODA queries can work with a server without application classes deployed. So we can expect optimized Native Queries (converted to SODA under the hood) to work as well:

Client.java: getPilotsNative

```

01  private static void getPilotsNative() throws IOException {
02 |     System.out.println("Retrieving Pilot objects: Native Query");
03 |     ObjectContainer oc = Db4o.openClient("localhost", 0xdb40, "db4o", "db4o");
04  |     try {
05  |         List<Pilot> result = oc.query(new Predicate<Pilot>() {
06  |             public boolean match(Pilot pilot) {
07 |                 return pilot.getPoints() == 5;
08 |             }
09 |         });
10 |         listResult(result);
11  |     } finally {
12 |         oc.close();
13 |     }
14 }
```

Unfortunately, if the query cannot be optimized to SODA, the server needs to instantiate the classes to run the query. This is not possible if the class is not available

This query won't work:

Client.java: getPilotsNativeUnoptimized

```

01  private static void getPilotsNativeUnoptimized() throws IOException {
02 |     System.out.println("Retrieving Pilot objects: Native Query Unoptimized");
03 |     ObjectContainer oc = Db4o.openClient("localhost", 0xdb40, "db4o", "db4o");
04  |     try {
05  |         List<Pilot> result = oc.query(new Predicate<Pilot>() {
06  |             public boolean match(Pilot pilot) {
07 |                 return pilot.getPoints() % 2 == 0;
```

```
08 |         }
09 |     });
10 |     listResult(result);
11 | } finally {
12 |     oc.close();
13 | }
14 | }
```

This revision (2) was last Modified 2006-12-30T11:00:53 by Tetyana.

# Evaluations

Evaluations need to retrieve the actual object instance to be evaluated. That is why they do not work on a server without persistent classes:

Client.java: getPilotsEvaluation

```
01 ①②private static void getPilotsEvaluation() throws IOException {
02 |     System.out.println("Retrieving Pilot objects: Evaluation");
03 |     ObjectContainer oc = Db4o.openClient("localhost", 0xdb40, "db4o", "db4o");
04 ③④    try {
05 |         Query query = oc.query();
06 |
07 |         query.constrain(Pilot.class);
08 ⑤⑥        query.constrain(new Evaluation() {
09 ⑦⑧            public void evaluate(Candidate candidate) {
10 |                Pilot pilot = (Pilot) candidate.getObject();
11 |                candidate.include(pilot.getPoints() % 2 == 0);
12 |            }
13 |        });
14 |        ObjectSet result = query.execute();
15 |        listResult(result);
16 ⑨⑩    } finally {
17 |        oc.close();
18 |    }
19 }
```

This revision (3) was last Modified 2006-12-30T11:09:58 by Tetyana.

# Multidimensional Arrays

Currently you can't use multidimensional array fields in a server without persistent classes setup.

RecordBook.java

```

01 /* Copyright (C) 2004 - 2006 db4o objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.noclasses.client;
04
05
06 public class RecordBook {
07 |   private String[][] notes;
08 |   private int recordCounter;
09 |
10 |
11     public RecordBook(){
12 |      notes = new String[20][3];
13 |      recordCounter = 0;
14 |    }
15 |
16     public void addRecord(String period, String pilotName, String note){
17 |      String[] tempArray = {period, pilotName, note};
18 |      notes[recordCounter] = tempArray;
19 |      recordCounter++;
20 |    }
21 |
22     public String toString(){
23 |      String temp;
24 |      temp = "Record book: \n";
25 |      for (int i=0; i<recordCounter; i++) {
26 |        temp = temp + notes[i][0] + "/" + notes[i][1] + "/" + notes[i][2] + "\n";
27 |      }
28 |      return temp;
29 |    }
30 }
```

Client.java: saveMultiArray

```

01 private static void saveMultiArray() throws IOException {
```

```

02 |     System.out.println("Testing saving an object with multidimensional array field");
03 |     ObjectContainer oc = Db4o.openClient("localhost", 0xdb40, "db4o", "db4o");
04 |     try {
05 |         RecordBook recordBook = new RecordBook();
06 |         recordBook.addRecord("September 2006", "Michael Schumacher", "last race");
07 |         recordBook.addRecord("September 2006", "Kimi Raikkonen", "no notes");
08 |         oc.set(recordBook);
09 |     } finally {
10 |         oc.close();
11 |     }
12 |

```

## Client.java: getMultiArray

```

01 |     private static void getMultiArray() throws IOException {
02 |         System.out.println("Testing retrieving an object with multidimensional array field");
03 |         ObjectContainer oc = Db4o.openClient("localhost", 0xdb40, "db4o", "db4o");
04 |         try {
05 |             ObjectSet result = oc.get(new RecordBook());
06 |             listResult(result);
07 |         } finally {
08 |             oc.close();
09 |         }
10 |

```

This revision (2) was last Modified 2006-12-30T10:58:43 by Tetyana.

# Switching Database Files in CS Mode

## Contents

- [Switching Database Files](#)
- [Multiple Database Usecase](#)

ExtClient is an extended interface, which can be used to access database from the client side. ExtClient object can be simply cast from the ObjectContainer, returned by openClient method:

```
Java: ExtClient clientExt = (ExtClient) server.openClient()
```

Please, note that this cast is only possible in embedded server mode.

ExtClient is extending ExtObjectContainer interface. In addition ExtClient provides 3 client-specific methods:

- `switchToFile(String fileName)/SwitchToFile(string fileName)`  
allows to switch your client database connection to another file using the same server connection.
- `switchToMainFile()/SwitchToMainFile()`  
switches the connection back to the original database file.
- `boolean isAlive()/IsAlive()`  
checks if the client is currently connected to a server, returns true if the client is alive.

## Switching Database Files

---

In some special cases you may want to store different objects to different files.

This approach has the following advantages:

- easier rights management
- easier backup
- possible later load balancing to multiple servers
- better performance of smaller individual database files
- special debugging database files can be used

To make database switch handling easier you should consider using the `ExtClient#switchToFile(fileName)` and `ExtClient#switchToMainFile()` methods. This allows you to use single connection to the server for all database files, thus avoiding overhead of opening and managing new connections (a server starts a new thread for each connection). All you have to do is just point out which database file to use.

<b>ExtClientExample.java: switchExtClients</b>
--

```

01  private static void switchClients() {
02      new File(DB40_FILE_NAME).delete();
03      new File(EXTFILENAME).delete();
04      ObjectServer server=Db4o.openServer(DB40_FILE_NAME, 0);
05      try {
06          ObjectContainer client=server.openClient();
07          deleteAll(client); // added to solve sticking objects in
doctor
08          Car car = new Car("BMW");
09          client.set(car);
10         System.out.println("Objects in the main database file: ");
11         retrieveAll(client);
12
13         System.out.println("Switching to additional database: ");
14         ExtClient clientExt = (ExtClient)client;
15         clientExt.switchToFile(EXTFILENAME);
16         car = new Car("Ferrari");
17         clientExt.set(car);
18         retrieveAll(clientExt);
19         System.out.println("Main database file again: ");
20         clientExt.switchToMainFile();
21         retrieveAll(clientExt);
22         clientExt.close();
23     }
24     finally {
25         server.close();
26     }
27 }
```

## Multiple Database Usecase

The following can be an example usecase of multiple database usage.

The main database file is used for login, user and rights management only. Multiple satellite database files are used for different applications or multiple user circles. User authorization to the satellite databases is not

checked.

Only one single db4o server session needs to be run.

This revision (8) was last Modified 2007-09-01T14:19:29 by Tetyana.

# Semaphores

## Contents

- [OO Languages Semaphores](#)
- [db4o Semaphore](#)

db4o Semaphores are named flags that can only be owned by one client/transaction at one time. They are supplied to be used as locks for exclusive access to code blocks in applications and to signal states from one client to the server and to all other clients.

The naming of semaphores is up to the application. Any string can be used.

Semaphores are freed automatically when a client disconnects correctly or when a clients presence is no longer detected by the server, that constantly monitors all client connections.

Semaphores can be set and released with the following two methods:

Java:

```
ExtObjectContainer#setSemaphore(String name, int waitMilliSeconds);  
ExtObjectContainer#releaseSemaphore(String name);
```

## OO Languages Semaphores

The concept of db4o semaphores is very similar to the concept of synchronization in OO programming languages:

Java:

```
synchronized(monitorObject){  
    // exclusive code block here  
}
```

## db4o Semaphore

Java:

```
if(objectContainer.ext().setSemaphore(semaphoreName, 1000){
```

```
// exclusive code block here  
objectContainer.ext().releaseSemaphore(semaphoreName);  
}
```

Although the principle of semaphores is very simple they are powerful enough for a wide range of usecases.

More Reading:

- [Locking Objects](#)
- [Ensuring Singletons](#)
- [Limiting the Number of Users](#)
- [Controlling Login Information](#)

This revision (10) was last Modified 2006-12-14T19:01:06 by Tetyana.

# Locking Objects

LockManager.java

```
01/* Copyright (C) 2004 - 2007 db4objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.semaphores;
04
05 import com.db4o.*;
06 import com.db4o.ext.*;
07
08/**
09 * This class demonstrates a very rudimentary implementation of
10 * virtual "locks" on objects with db4o. All code that is intended to
11 * obey these locks will have to call lock() and unlock().
12 */
13
14
15 private final String SEMAPHORE_NAME = "locked: ";
16
17 private final int WAIT_FOR_AVAILABILITY = 300; // 300
18 // milliseconds
19
20 private final ExtObjectContainer _objectContainer;
21
22 public LockManager(ObjectContainer objectContainer) {
23     _objectContainer = objectContainer.ext();
24 }
25
26 public boolean lock(Object obj) {
27     long id = _objectContainer.getID(obj);
28     return _objectContainer.setSemaphore(SEMAPHORE_NAME + id,
```

```
29 |     WAIT_FOR_AVAILABILITY) ;  
30 | }  
31 |  
32 | public void unlock(Object obj) {  
33 |     long id = _objectContainer.getID(obj);  
34 |     _objectContainer.releaseSemaphore(SEMAPHORE_NAME + id);  
35 | }  
36 |}
```

This revision (5) was last Modified 2006-11-14T12:02:27 by Tetyana.

# Ensuring Singletons

Singleton.java

```
01/* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */  
02  
03 package com.db4odoc.semaphores;  
04  
05 import com.db4o.*;  
06 import com.db4o.query.*;  
07  
08/**  
09 * This class demonstrates the use of a semaphore to ensure that only  
10 * one instance of a certain class is stored to an ObjectContainer.  
11 *  
12 * Caution !!! The getSingleton method contains a commit() call.  
13 */  
14public class Singleton {  
15  
16    /**  
17     * returns a singleton object of one class for an ObjectContainer.  
18     * <br>  
19     * <b>Caution !!!</b> This method contains a commit() call. </b>  
20     */  
21     public static Object getSingleton(  
22         ObjectContainer objectContainer, Class clazz) {  
23  
24         Object obj = queryForSingletonClass(objectContainer, clazz);  
25         if (obj != null) {  
26             return obj;  
27         }  
28     }
```

```
29 |     String semaphore = "Singleton#getSingleton_"
30 |         + clazz.getName();
31 |
32 |         if (!objectContainer.ext().setSemaphore(semaphore, 10000)) {
33 |             throw new RuntimeException("Blocked semaphore "
34 |                 + semaphore);
35 |         }
36 |
37 |         obj = queryForSingletonClass(objectContainer, clazz);
38 |
39 |         if (obj == null) {
40 |
41 |             try {
42 |                 obj = clazz.newInstance();
43 |             } catch (InstantiationException e) {
44 |                 e.printStackTrace();
45 |             } catch (IllegalAccessException e) {
46 |                 e.printStackTrace();
47 |             }
48 |
49 |             objectContainer.set(obj);
50 |
51 |             /*
52 |             * !!! CAUTION !!! There is a commit call here.
53 |             *
54 |             * The commit call is necessary, so other transactions can
55 |             * see the new inserted object.
56 |             */
57 |             objectContainer.commit();
58 |
59 |         }
60 |
61 |         objectContainer.ext().releaseSemaphore(semaphore);
```

```
62 |
63 |     return obj;
64 | }
65 |
66 |     private static Object queryForSingletonClass(
67 |         ObjectContainer objectContainer, Class clazz) {
68 |         Query q = objectContainer.query();
69 |         q.constrain(clazz);
70 |         ObjectSet objectSet = q.execute();
71 |         if (objectSet.size() == 1) {
72 |             return objectSet.next();
73 |         }
74 |         if (objectSet.size() > 1) {
75 |             throw new RuntimeException(
76 |                 "Singleton problem. Multiple instances of: "
77 |                 + clazz.getName());
78 |         }
79 |         return null;
80 |     }
81 |
82 }
```

This revision (5) was last Modified 2006-11-14T12:00:51 by Tetyana.

# Limiting the Number of Users

LimitLogins.java

```
01 /* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.semaphores;
04
05 import com.db4o.*;
06
07 /**
08 * This class demonstrates the use of semaphores to limit the number
09 * of logins to a server.
10 */
11 public class LimitLogins {
12
13     static final String HOST = "localhost";
14
15     static final int PORT = 4455;
16
17     static final String USER = "db4o";
18
19     static final String PASSWORD = "db4o";
20
21     static final int MAXIMUM_USERS = 10;
22
23     public static ObjectContainer login() {
24
25         ObjectContainer objectContainer;
26         try {
27             objectContainer = Db4o.openClient(HOST, PORT, USER,
28                 PASSWORD);
```

```
29 } catch (Exception e) {  
30 |     return null;  
31 | }  
32 |  
33 |     boolean allowedToLogin = false;  
34 |  
35 |     for (int i = 0; i < MAXIMUM_USERS; i++) {  
36 |         if (objectContainer.ext().setSemaphore(  
37 |             "max_user_check_" + i, 0)) {  
38 |             allowedToLogin = true;  
39 |             break;  
40 |         }  
41 |     }  
42 |  
43 |     if (!allowedToLogin) {  
44 |         objectContainer.close();  
45 |         return null;  
46 |     }  
47 |  
48 |     return objectContainer;  
49 | }  
50 }
```

This revision (5) was last Modified 2006-11-14T12:01:39 by Tetyana.

# Controlling Login Information

ConnectedUser.java

```
001 /* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */
002
003 package com.db4odoc.semaphores;
004
005 import java.util.*;
006 import com.db4o.*;
007 import com.db4o.config.*;
008 import com.db4o.ext.*;
009 import com.db4o.query.*;
010
011 /**
012 * This class demonstrates how semaphores can be used to rule out race
013 * conditions when providing exact and up-to-date information about
014 * all connected clients on a server. The class also can be used to
015 * make sure that only one login is possible with a give user name and
016 * ipAddress combination.
017 */
018 public class ConnectedUser {
019
020     static final String SEMAPHORE_CONNECTED = "ConnectedUser_";
021
022     static final String SEMAPHORE_LOCK_ACCESS = "ConnectedUser_Lock_";
023
024     static final int TIMEOUT = 10000; // concurrent access timeout
025
026     // 10 seconds
027
028     String userName;
```

```
029 |
030 |     String ipAddress;
031 |
032 |     public ConnectedUser(String userName, String ipAddress) {
033 |         this.userName = userName;
034 |         this.ipAddress = ipAddress;
035 |     }
036 |
037 |     // make sure to call this on the server before opening the
038 |     // database
039 |     // to improve querying speed
040 |     public static void configure() {
041 |         ObjectClass objectClass = Db4o.configure().objectClass(
042 |             ConnectedUser.class);
043 |         objectClass.objectField("userName").indexed(true);
044 |         objectClass.objectField("ipAddress").indexed(true);
045 |     }
046 |
047 |     // call this on the client to ensure to have a ConnectedUser
048 |     // record
049 |     // in the database file and the semaphore set
050 |     public static void login(ObjectContainer client, String userName,
051 |         String ipAddress) {
052 |         if (!client.ext()
053 |             .setSemaphore(SEMAPHORE_LOCK_ACCESS, TIMEOUT)) {
054 |             throw new RuntimeException(
055 |                 "Timeout trying to get access to ConnectedUser lock");
056 |         }
057 |         Query q = client.query();
058 |         q.constrain(ConnectedUser.class);
059 |         q.descend("userName").constrain(userName);
060 |         q.descend("ipAddress").constrain(ipAddress);
061 |         if (q.execute().size() == 0) {
```

```
062 |     client.set(new ConnectedUser(userName, ipAddress));
063 |     client.commit();
064 | }
065 | String connectedSemaphoreName = SEMAPHORE_CONNECTED
066 |     + userName + ipAddress;
067 | boolean unique = client.ext().setSemaphore(
068 |     connectedSemaphoreName, 0);
069 | client.ext().releaseSemaphore(SEMAPHORE_LOCK_ACCESS);
070 | if (!unique) {
071 |     throw new RuntimeException(
072 |         "Two clients with same userName and ipAddress");
073 | }
074 | }
075 |
076 | // here is your list of all connected users, callable on the
077 | // server
078 | public static List connectedUsers(ObjectServer server) {
079 |     ExtObjectContainer serverObjectContainer = server.ext()
080 |         .objectContainer().ext();
081 |     if (serverObjectContainer.setSemaphore(SEMAPHORE_LOCK_ACCESS,
082 |         TIMEOUT)) {
083 |         throw new RuntimeException(
084 |             "Timeout trying to get access to ConnectedUser lock");
085 |     }
086 |     List list = new ArrayList();
087 |     Query q = serverObjectContainer.query();
088 |     q.constrain(ConnectedUser.class);
089 |     ObjectSet objectSet = q.execute();
090 |     while (objectSet.hasNext()) {
091 |         ConnectedUser connectedUser = (ConnectedUser) objectSet
092 |             .next();
093 |         String connectedSemaphoreName = SEMAPHORE_CONNECTED
094 |             + connectedUser.userName
```

```
095 |         + connectedUser.ipAddress;
096 |     if (serverObj ectContainer.setSemaphore(
097 |         connectedSemaphoreName, TIMEOUT)) {
098 |         serverObj ectContainer.delete(connectedUser);
099 |     } else {
100 |         list.add(connectedUser);
101 |     }
102 | }
103 |     serverObj ectContainer.commit();
104 |     serverObj ectContainer.releaseSemaphore(SEMAPHORE_LOCK_ACCESS);
105 |     return list;
106 | }
107 | }
```

This revision (5) was last Modified 2006-11-14T12:00:13 by Tetyana.

# Pluggable Sockets

db4o allows to customize client-server communication by using pluggable socket implementations:

Java:

```
public static ObjectServer openServer(Configuration config, String  
databaseFileName, int port, NativeSocketFactory socketFactory)
```

```
public static ObjectContainer OpenClient(Configuration config, String  
hostName, int port, String user, String password, NativeSocketFactory  
socketFactory)
```

Here NativeSocketFactory interface should be used to provide client and server sockets, which may implement any custom way of communication.

An example of such customary implementation can be [encrypted communication](#).

Note, that this API is in the development stage and is subject to future changes.

This revision (2) was last Modified 2007-12-23T20:04:14 by Tetyana.

# Using SSL For Client-Server Communication

This topic applies to Java version only

With the default settings db4o client-server communication is not encrypted and thus can potentially be a dangerous security hole. This can be fixed with a [new pluggable socket](#) client/server implementation. Let's look at a simple example - we will use SSL protocol to protect our communication channel.

Basically the task is to create a NativeSocketFactory implementation that will create sockets able to communicate through an encrypted channel. In Java these would be SSLSocket and SSLServerSocket.

SecureSocketFactory.java

```

01  /* Copyright (C) 2007 db4o objects Inc. http://www.db4o.com */
02  package com.db4odoc.ssl;
03
04  import java.io.IOException;
05  import java.net.ServerSocket;
06  import java.net.Socket;
07
08  import javax.net.ssl.SSLContext;
09
10  import com.db4o.config.NativeSocketFactory;
11
12  public class SecureSocketFactory implements NativeSocketFactory {
13
14      private SSLContext _context;
15
16      public SecureSocketFactory(SSLContext context) {
17          _context = context;
18      }
19
20      public ServerSocket createServerSocket(int port) throws IOException {
21          System.out.println("SERVER on " + port);
22          return _context.getServerSocketFactory().createServerSocket(port);
23      }
24
25      public Socket createSocket(String hostName, int port) throws IOException {
26          System.out.println("CLIENT on " + port);
27          return _context.getSocketFactory().createSocket(hostName, port);
28      }
29

```

```

30 |     public Object deepClone(Object context) {
31 |         return this;
32 |     }
33 |
34 |}

```

In order for this class to work correctly we need to provide a correctly initialized SSLContext. For those who are not familiar with Java SSL implementation it is recommended to get acquainted with Java documentation for SSLContext, TrustManager, KeyStore and KeyManagerFactory APIs.

For encryption purposes we will need to create a new keystore. This can be done with the following command:

```
keytool -genkey -keystore SSLcert -storepass password
```

This command creates a file SSLcert, which contains a keystore protected by "password" password. For a test keystore you can provide any answers to the keytool questions and leave the key password the same as the keystore password. For easy access copy the SSLcert file into your projects directory.

Now we are ready to create a SecureSocketFactory

SSL.SocketsExample.java: createSecureSocketFactory

```

01 | private static SecureSocketFactory createSecureSocketFactory() throws Exception{
02 |     SSLContext sc;
03 |
04 |     //Create a trust manager that does not validate certificate chains
05 |     TrustManager[] trustAllCerts = createTrustManager();
06 |
07 |     // Install the all-trusting trust manager
08 |     sc = SSLContext.getInstance("SSLv3");
09 |     KeyStore ks = KeyStore.getInstance(KEYSTORE_ID);
10 |     ks.load(new FileInputStream(KEYSTORE_PATH), null);
11 |     KeyManagerFactory kmf = KeyManagerFactory.getInstance( KeyManagerFactory.
getDefaultAlgorithm());
12 |     kmf.init(ks, KEYSTORE_PASSWORD.toCharArray());
13 |
14 |     sc.init(kmf.getKeyManagers(), trustAllCerts, new java.security.SecureRandom());
15 |
16 |     return new SecureSocketFactory(sc);
}

```

SSL.SocketsExample.java: createTrustManager

```

01 | private static TrustManager[] createTrustManager(){
02 |     return new TrustManager[]{}
}

```

```

03  new X509TrustManager() {
04      public java.security.cert.X509Certificate[] getAcceptedIssuers() {
05          return null;
06      }
07      public void checkClientTrusted(
08          java.security.cert.X509Certificate[] certs, String authType) {
09      }
10      public void checkServerTrusted(
11          java.security.cert.X509Certificate[] certs, String authType) {
12      }
13  }
14 }
15 }

```

Starting a server and opening client connections with the new socket factory is as simple as usual:

SSL.SocketsExample.java: main

```

01  public static void main(String[] args) throws Exception {
02
03      // Create a SecureSocketFactory for the SSL context
04      socketFactory = createSecureSocketFactory();
05
06      Configuration config = Db4o.newConfiguration();
07      ObjectServer db4oServer = Db4o.openServer(config, FILE, PORT,
08          socketFactory);
09      db4oServer.grantAccess(USER, PASSWORD);
10     try {
11         storeObjectsRemotely(HOST, PORT, USER, PASSWORD);
12         queryRemoteServer(HOST, PORT, USER, PASSWORD);
13     } finally {
14         db4oServer.close();
15     }
16 }

```

SSL.SocketsExample.java: storeObjectsRemotely

```

1  private static void storeObjectsRemotely(String host, int port, String user, String
password) throws IOException {
2      Configuration config = Db4o.newConfiguration();
3      ObjectContainer client=Db4o.openClient(config, "localhost", port, user, password,

```

```
socketFactory);  
4 |         Pilot pilot = new Pilot("Fernando Alonso", 89);  
5 |         client.set(pilot);  
6 |         client.close();  
7 |     }
```

## SSLSocketsExample.java: queryRemoteServer

```
1 private static void queryRemoteServer(String host, int port, String user, String password)  
throws IOException {  
2 |     Configuration config = Db4o.newConfiguration();  
3 |     ObjectContainer client=Db4o.openClient(config, "localhost", port, user, password,  
socketFactory);  
4 |     listResult(client.get(new Pilot(null, 0)));  
5 |     client.close();  
6 | }
```

This revision (2) was last Modified 2007-12-28T04:09:45 by German Viscuso.

# Remote Code Execution

Sometimes you will need your client to update many objects in a similar way on the server. The solution that is sought in this case is to give a server a criteria for selecting objects and instructions on what to do with them. In this way you will avoid the overhead of getting the objects over network and sending the updated set back.

More Reading:

- [Remote Execution Through Evaluation API](#)
- [Using Messaging API For Remote Code Execution](#)

This revision (4) was last Modified 2006-11-15T11:50:26 by Tetyana.

# Remote Execution Through Evaluation API

One of the ways to do that is using evaluation API. Evaluation/candidate classes are serialized and sent to the server. That means that if we will put the selection criteria and update code inside evaluation class, we will have that code on the server and executing a query using evaluation on the client side will run the update code on the server side.

For easier query execution we can use database singleton - a class that have only one instance saved in the database. That actually can be the class calling the query itself.

Let's fill up our server database:

**RemoteExample.java: setObjects**

```

01  private static void setObjects() {
02 |     new File(DB40_FILE_NAME).delete();
03 |     ObjectContainer container = Db4o.openFile(DB40_FILE_NAME);
04  |     try {
05  |         for (int i = 0; i < 5; i++) {
06 |             Car car = new Car("car" + i);
07 |             container.set(car);
08 |         }
09 |         container.set(new RemoteExample());
10  |     } finally {
11 |         container.close();
12 |     }
13 |     checkCars();
14 }
```

This method has its pros and cons.

Pros:

- any arbitrary code can be executed;
- the code is executed on the server side;

Cons:

- the code will have to be serialized and sent over a network connection;
- changing the code will require update of all clients

This revision (5) was last Modified 2006-11-14T11:50:58 by Tetyana.

# Using Messaging API For Remote Code Execution

Messaging API gives you an easy and powerful tool for remote code execution. The short overview of the API is in [Messaging chapter](#).

All you will need to do is to define specific message class or classes (should be shared between client and server).

The client side can issue messages using:

Java: `MessageSender#send(message)`

The server side should register a message listener:

Java: `Configuration#setMessageRecipient(MessageRecipient)`

Message recipient should define a response to the different messages received in

Java: `processMessage(ObjectContainer objectContainer, Object message)`

method. `ObjectContainer` parameter gives full control over the database.

Let's reset our database and try updating using special `UpdateServer` message.

`RemoteExample.java: setObjects`

```

01 ①②private static void setObjects() {
02 |     new File(DB40_FILE_NAME).delete();
03 |     ObjectContainer container = Db4o.openFile(DB40_FILE_NAME);
04 ④⑤    try {
05 ④⑥        for (int i = 0; i < 5; i++) {
06 |            Car car = new Car("car" + i);
07 |            container.set(car);
08 |        }
09 |        container.set(new RemoteExample());
10 ④⑦    } finally {
11 |        container.close();
12 |    }
13 |    checkCars();
14 }
```

`RemoteExample.java: updateCarsWithMessage`

```

01 ①②private static void updateCarsWithMessage() {
```

```

02 |     Configuration configuration = Db4o.newConfiguration();
03 |     configuration.messageLevel(0);
04 |     ObjectServer server = Db4o.openServer(configuration, DB40_FILE_NAME, 0xdb40);
05 |     server.grantAccess("user", "password");
06 |     // create message handler on the server
07 |     server.ext().configure().clientServer().setMessageRecipient(
08 |         new MessageRecipient() {
09 |             public void processMessage(
10 |                 MessageContext context,
11 |                 Object message) {
12 |                     // message type defines the code to be
13 |                     // executed
14 |                     if (message instanceof UpdateServer) {
15 |                         Query q = context.container().query();
16 |                         q.constrain(Car.class);
17 |                         ObjectSet objectSet = q.execute();
18 |                         while (objectSet.hasNext()) {
19 |                             Car car = (Car) objectSet.next();
20 |                             car.setModel ("Updated2-"
21 |                                 + car.getModel ());
22 |                             context.container().set(car);
23 |                         }
24 |                         context.container().commit();
25 |                     }
26 |                 }
27 |             });
28 |         try {
29 |             ObjectContainer client = Db4o.openClient("localhost", 0xdb40, "user", "password");
30 |             // send message object to the server
31 |             MessageSender sender = client.ext().configure()
32 |                 .clientServer().getMessageSender();
33 |             sender.send(new UpdateServer());
34 |             client.close();
35 |         } finally {
36 |             server.close();
37 |         }
38 |         checkCars();
39 |     }

```

You can also put some information in the message being sent (UpdateServer class).

The advantage of this method is that having predefined message types you can make any changes to the handling code only on the server side. In the situations with many distributed clients it can really save you lots of time on support.

This revision (10) was last Modified 2007-05-09T07:45:01 by Tetyana.

# Concurrency Control

As soon as you will start using db4o with multiple client connections you will recognize the necessity of implementing a concurrency control system. Db4o itself works as an overly optimistic scheme, i.e. an object is locked for read and write, but no collision check is made. This approach makes db4o very flexible and gives you an opportunity to organize a concurrency control system, which will suit your needs the best. Your main tools to build your concurrency control system would be:

- [semaphores](#)
- [callbacks](#)

The articles in this topic set will show you how to implement locking and will give you an advice which strategy to use in different situations.

More Reading:

- [Optimistic Locking](#)
- [Pessimistic Locking](#)

This revision (6) was last Modified 2007-05-09T07:46:13 by Tetyana.

# Optimistic Locking

In optimistic locking system no locks are used to prevent collision: any user can read an object into the memory and work on it at any time. However, before the client can save its modifications back to the database, a check should take place verifying that the item did not change since the time of initial read (no collision occurred). If a collision is detected it should be resolved according to your application logic. Typical solutions are:

- Rollback
- Display the problem and let the user decide
- Merge the changes
- Log the problem so someone can decide later
- Ignore the collision and overwrite

Let's look at an example realization.

We will use a db4o database containing objects of Pilot class and a separate thread to create a client connection to the database, retrieve and modify objects.

OptimisticThread.java: run

```

01  public void run() {
02    try {
03      ObjectSet result = _container.get(Pilot.class);
04      while (result.hasNext()) {
05        Pilot pilot = (Pilot) result.next();
06        /* We will need to set a lock to make sure that the
07         * object version corresponds to the object retrieved.
08         * (Prevent other client committing changes
09         * at the time between object retrieval and version
10         * retrieval )
11        */
12        if (!_container.ext().setSemaphore("LOCK_"+_container.ext().getID(pilot), 3000))
13          System.out.println("Error. The object is locked");
14        continue;
15      }
16      long objVersion = _container.ext().getObjectInfo(pilot).getVersion();
17      _container.ext().refresh(pilot, Integer.MAX_VALUE);
18      _container.ext().releaseSemaphore("LOCK_"+_container.ext().getID(pilot));
19
20      /* save object version into _idVersions collection
21       * This will be needed to make sure that the version
22       * originally retrieved is the same in the database
23       * at the time of modification
24      */

```

```

25 |         long id = _container.ext().getID(pilot);
26 |         _idVersions.put(id, objVersion);
27 |
28 |         System.out.println(getName() + "Updating pilot: " + pilot + " version:
29 | " + objVersion);
30 |         pilot.addPoints(1);
31 |         _updateSuccess = false;
32 |         randomWait();
33 |         if (!_container.ext().setSemaphore("LOCK_" + _container.ext().getID(pilot),
34 | 3000)) {
35 |             System.out.println("Error. The object is locked");
36 |             continue;
37 |         }
38 |         _container.set(pilot);
39 |         /* The changes should be committed to be
40 |          * visible to the other clients
41 |          */
42 |         _container.commit();
43 |         _container.ext().releaseSemaphore("LOCK_" + _container.ext().getID(pilot));
44 |         if (_updateSuccess) {
45 |             System.out.println(getName() + "Updated pilot: " + pilot);
46 |         }
47 |         System.out.println();
48 |         /* The object version is not valid after commit
49 |          * - should be removed
50 |          */
51 |
52 |         _idVersions.remove(id);
53 |     }
54 | }
55 |

```

A semaphore is used for locking the object before saving and the lock is released after commit when the changes become visible to the other clients. The semaphore is assigned a name based on object ID to make sure that only the modified object will be locked and the other clients can work with the other objects of the same class simultaneously.

Locking the object for the update only ensures that no changes will be made to the object from the other clients during update. However the object might be already changed since the time when the current thread retrieved it. In order to check this we will need to implement an event handler for the updating event:

OptimisticThread.java: registerCallbacks

```

01  public void registerCallbacks() {
02 |     EventRegistry registry = EventRegistryFactory.forObjectContainer(_container);
03 |     // register an event handler to check collisions on update
04 |     registry.updating().addListener(new EventListener4() {
05 |         public void onEvent(Event4 e, EventArgs args) {
06 |             CancelableObjectEventArgs queryArgs = ((CancelableObjectEventArgs) args);
07 |             Object obj = queryArgs.object();
08 |             // retrieve the object version from the database
09 |             long currentVersion = _container.ext().getObjectInfo(obj).getVersion();
10 |             long id = _container.ext().getID(obj);
11 |             // get the version saved at the object retrieval
12 |             long initialVersion = ((Long)_idVersions.get(id)).longValue();
13 |             if (initialVersion != currentVersion) {
14 |                 System.out.println(getName() + "Collision: ");
15 |                 System.out.println(getName() + "Stored object: version: " + currentVersion);
16 |                 System.out.println(getName() + "New object: " + obj + " version: " +
initialVersion);
17 |                 queryArgs.cancel();
18 |             } else {
19 |                 _updateSuccess = true;
20 |             }
21 |         }
22 |     });
23 | }

```

In the above case the changes are discarded and a message is sent to the user if the object is already modified from another thread. You can replace it with your own strategy of collision handling.

Note: the supplied example has random delays to make the collision happen. You can experiment with the delay values to see different behavior.

#### OptimisticThread.java: randomWait

```

1  private void randomWait() {
2 |     try {
3 |         Thread.sleep((long)(5000*Math.random()));
4 |     } catch(InterruptedException e) {
5 |         System.out.println("Interrupted!");
6 |     }
7 |
}

```

This revision (1) was last Modified 2006-12-18T14:39:30 by Tetyana.

# Pessimistic Locking

Pessimistic locking is an approach when an entity is locked in the database for the entire time that it is in application memory. This means that an object should be locked as soon as it is retrieved from the database and released after commit.

PessimisticThread.java: run

```

01  public void run() {
02    try {
03      ObjectSet result = _container.get(Pilot.class);
04      while (result.hasNext()) {
05        Pilot pilot = (Pilot) result.next();
06        /* with pessimistic approach the object is locked as soon
07        * as we get it
08        */
09        if (!_container.ext().setSemaphore("LOCK_" + _container.ext().getID(pilot), 0)) {
10          System.out.println("Error. The object is locked");
11        }
12
13        System.out.println(getName() + "Updating pilot: " + pilot);
14        pilot.addPoints(1);
15        _container.set(pilot);
16        /* The changes should be committed to be
17        * visible to the other clients
18        */
19        _container.commit();
20        _container.ext().releaseSemaphore("LOCK_" + _container.ext().getID(pilot));
21        System.out.println(getName() + "Updated pilot: " + pilot);
22        System.out.println();
23      }
24    } finally {
25      _container.close();
26    }
27  }

```

As you see this approach is considerably easier to implement. Another advantage is that it guarantees that your changes to the database are made consistently and safely.

The main disadvantage is the lack of scalability. Time waiting for the lock to be released can become unacceptable for a system with many users or long transactions. This limits the practical implementations of pessimistic locking.

You may want to select pessimistic locking in cases when the cost of loosing the transaction results due to a collision is too high.

This revision (2) was last Modified 2006-12-18T14:50:20 by Tetyana.

# Messaging

In client/server mode the TCP connection between the client and the server can also be used to send messages from the client to the server.

Possible usecases could be:

- shutting down and restarting the server
- triggering server backup
- using a customized login strategy to restrict the number of allowed client connections

Here is some example code how this can be done.

First we need to decide on a class that we want to use as the message. Any object that is storables in db4o can be used as a message, but strings and other simple types will not be carried (as they are not storables in db4o on their own). Let's create a dedicated class:

## MyClientServerMessage.java

```

01  /* Copyright (C) 2004 - 2007 db4objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.messaging;
04
05
06 class MyClientServerMessage {
07 |
08     private String info;
09
10    public MyClientServerMessage(String info) {
11        this.info = info;
12    }
13
14    public String toString() {
15        return "MyClientServerMessage: " + info;
16    }
17
18 }
```

Now we have to add some code to the server to react to arriving messages. This can be done by configuring a MessageRecipient object on the server. Let's simply print out all objects that arrive as messages. For the following we assume that we already have an ObjectServer object, opened with [Db4o.openServer\(\)](#).

## MessagingExample.java

```
01 /* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */  
02  
03 package com.db4odoc.messaging;  
04  
05 import com.db4o.Db4o;  
06 import com.db4o.ObjectContainer;  
07 import com.db4o.ObjectServer;  
08 import com.db4o.config.Configuration;  
09 import com.db4o.messaging.MessageContext;  
10 import com.db4o.messaging.MessageRecipient;  
11 import com.db4o.messaging.MessageSender;  
12  
13  
14 public class MessagingExample {  
15     private final static String DB4O_FILE_NAME="reference.db4o";  
16  
17     public static void configureServer() {  
18         Configuration configuration = Db4o.newConfiguration();  
19         configuration.clientServer().setMessageRecipient(new MessageRecipient() {  
20             public void processMessage(MessageContext context,  
21                 Object message) {  
22                 // message objects will arrive in this code block  
23                 System.out.println(message);  
24             }  
25         });  
26         ObjectServer objectServer = Db4o.openServer(configuration, DB4O_FILE_NAME, 0);  
27  
28         try {  
29             ObjectContainer clientObjectContainer = objectServer.openClient();  
30             // Here is what we would do on the client to send the message  
31             MessageSender sender = clientObjectContainer.ext().configure().clientServer().  
getMessageSender();  
32  
33             sender.send(new MyClientServerMessage("Hello from client."));  
34             clientObjectContainer.close();  
35         } finally {
```

```
36 |         objectServer.close();
37 |     }
38 | }
39 | // end configureServer
40 |
41 }
```

The MessageSender object on the client can be reused to send multiple messages.

This revision (8) was last Modified 2008-04-14T16:05:00 by Tetyana.

# Batch Mode

Client-server batch messaging mode was introduced in db4o version 6.1. This mode allows to increase the performance by reducing client/server communication.

Java:

```
container.ext().configure().clientServer().batchMessages(true);
```

How it works? Db4o client communicates with the server by means of messaging. In the default mode (`batchMessages(false)`) db4o client sends a message with an instruction to the server and waits for the response. This might be quite inefficient when there are many small messages to be sent (like bulk inserts, updates, deletes): network communication becomes a bottleneck. Batch messaging mode solves this problem by caching the client messages on the client before sending them to the server.

The advantages of the batch messaging mode are:

- reduced network load;
- increased performance for bulk operations.

The downside is:

- increased memory consumption on both the client and the server.

More Reading:

- [Controlling Memory Consumption](#)
- [Batch Messaging Example](#)

This revision (5) was last Modified 2007-03-13T19:42:31 by Tetyana.

# Controlling Memory Consumption

In order to control the increasing memory usage in batch messaging mode the following configuration method is provided:

Java:

```
container.ext().configure().clientServer().maxBatchQueueSize(size);
```

When this configuration method is used, client messages are flushed to the server when the message queue reaches the specified size in bytes.

This revision (2) was last Modified 2007-03-13T19:46:10 by Tetyana.

# Batch Messaging Example

Let's create a small example to test batch messaging mode behavior. We will use bulk insert with and without batch messaging configuration.

BatchExample.java: fillUpDb

```

01  private static void fillUpDb(Configuration configuration) throws IOException {
02 |     System.out.println("Testing inserts");
03 |     ObjectContainer container = Db4o.openClient(HOST, PORT, USER,
04 |                                         PASS);
05 |     try {
06 |         long t1 = System.currentTimeMillis();
07 |         for (int i = 0; i < NO_OF_OBJECTS; i++) {
08 |             Pilot pilot = new Pilot("pilot #" + i, i);
09 |             container.set(pilot);
10 |         }
11 |         long t2 = System.currentTimeMillis();
12 |         long diff = t2 - t1;
13 |         System.out.println("Operation time: " + diff + " ms.");
14 |     } finally {
15 |         container.close();
16 |     }
17 | }
```

Let's configure the server and run the insert operation first without batch messages, then with batch messages:

BatchExample.java: main

```

01  public static void main(String[] args) throws IOException {
02 |     ObjectServer db4oServer = Db4o.openServer(FILE, PORT);
03 |     try {
04 |         db4oServer.grantAccess(USER, PASS);
05 |         Configuration configuration = Db4o.newConfiguration();
06 |         fillUpDb(configuration);
07 |         configuration.clientServer().batchMessages(true);
08 |         fillUpDb(configuration);
```

```
09 } finally {  
10 |     db4oServer.close();  
11 | }  
12 }
```

You can try different values of NO\_OF\_OBJECTS constant to see the difference.

If the value of NO\_OF\_OBJECTS is high (>1,000,000) you may notice that the memory consumption increases a lot.  
In order to decrease it, try using:

Java:

```
container.ext().configure().clientServer().maxBatchQueueSize(size);
```

Specify the size parameter according to the desirable memory consumption limit.

This revision (3) was last Modified 2007-03-13T19:41:00 by Tetyana.

# db4o Replication System (dRS)

## Contents

- [Java Platform](#)

db4o Replication System (dRS) provides functionality to periodically synchronize databases that work disconnected from each other, such as remote autonomous servers or handheld devices synchronizing with central servers.

Before you start, please make sure that you have downloaded the latest dRS distribution from the [download area](#) of the [db4objects website](#).

The list of currently supported replication features currently include:

## Java Platform

---

- db4o to db4o replication
- db4o to Hibernate replication
- Hibernate to Hibernate replication
- Array Replication
- Inheritance hierarchies
- Standard Primitive Types
- One-To-One Relations
- One-To-Many Relations
- Many-To-One Relations
- UUID and Version generation for db4o and Hibernate
- Queries for changed objects
- Parent-to-Child traversal along changed objects
- Collection Replication
- Replication Event System
- Replication of deleted objects

For the list of unsupported and planned features, please, see dRS project in our [Jira tracking system](#).

We invite you to join the db4o community in the public [db4o forums](#) to ask for help at any time. You may also find the db4o [knowledgebase](#) helpful for keyword searches.

More Reading:

- [Getting Started](#)
- [Db4o Databases Replication](#)
- [Replication With RDBMS](#)
- [Advanced Replication Strategies](#)
- [Deployment Instructions](#)
- [License](#)

This revision (10) was last Modified 2007-07-08T13:56:57 by Tetyana.

# Getting Started

## Contents

- [Download Contents](#)
  - [Java Platform](#)
- [Requirements](#)
  - [Java version](#)
  - [.NET version](#)
- [Installation](#)
  - [Java Platform Installation](#)

This topic will give you some initial information about dRS and will help to get your environment ready to work with dRS.

## Download Contents

---

## Java Platform

---

The distribution comes as a zip archive. The contents need to be extracted to any folder before you start to use the dRS.

You can find the offline version of this documentation in **/doc/reference** folder.

The API documentation is located in **/doc/api** folder.

db4o and [dRS online](#) documentation resources are located in the [reference documentation](#).

## Requirements

---

### Java version

---

dRS is dependent on the JDK 5 version of db4o object database. You will also need a corresponding db4o version (compatible db4o and dRS have the same version number).

To use the Hibernate Replication functionality, the Hibernate core version 3.1 files are required. These files and their dependencies have been included in the /lib folder for your convenience. If you require more complicated Hibernate mapping configurations, you may wish to download the full Hibernate documentation from the [Hibernate project website](#).

To run the provided build scripts dRS requires [Apache Ant](#) 1.6 or later.

## .NET version

---

dRS requires Microsoft .NET framework version 2.0 to work. A compatible version of db4o database should be used (compatible db4o and dRS have the same version number).

To work with the source code and examples you will need Microsoft Visual Studio 2005.

## Installation

---

### Java Platform Installation

---

dRS functionality is delivered in a single jar file. If you add dRS.jar to your CLASSPATH, dRS is installed. For using db4o replication to another database (RDBMS) you will also need hibernate library, which is included in the /lib folder in the distribution.

In case you work with an integrated development environment like [Eclipse](#) you can copy the dRS jar to a / lib/ folder under your project and add dRS to your project as a library.

Here is how to add the dRS library to an Eclipse project

- create a folder named "lib" under your project directory, if it doesn't exist yet
- copy dRS.jar to this folder
- Right-click on your project in the Package Explorer and choose "refresh"
- Right-click on your project in the Package Explorer again and choose "properties"
- select "Java Build Path" in the treeview on the left
- select the "Libraries" tabpage.
- click "Add Jar"
- the "lib" folder should appear below your project
- choose dRS.jar in this folder
- hit OK twice

You might need to repeat the same for hibernate3.jar

This revision (7) was last Modified 2007-07-10T10:00:41 by Tetyana.

# Db4o Databases Replication

db4o-to-db4o replication is the simplest replication and is supported by both java and .NET versions.

In the following examples we will use Pilot class from the [db4o documentation](#).

Pilot.java

```
01 /* Copyright (C) 2004 - 2007 db4o objects Inc. http://www.db4o.com */
02
03 package com.db4odoc.replication;
04
05 public class Pilot {
06     private String name;
07
08     public Pilot(String name) {
09         this.name=name;
10    }
11
12     public String getName() {
13         return name;
14    }
15
16     public String toString() {
17         return name;
18    }
19 }
```

More Reading:

- [Initial Setup](#)
- [Simple Example](#)
- [Bi-Directional Replication](#)
- [Selective Replication](#)

This revision (2) was last Modified 2007-07-08T15:31:32 by Tetyana.

# Initial Setup

When replicating objects to and from a db4o database, we need to enable UUIDs and VersionNumbers.

UUIDs are object IDs that are unique across all databases created with db4o. That is achieved by having the database creation timestamp as part of its objects UUIDs. The db4o UUID contains two parts. The first part contains an object ID. The second part identifies the database that originally created this ID. More information on the UUIDs can be found in the [db4o reference documentation](#).

The replication system will use the version number to invisibly tell when an object was last replicated, and if any changes have been made to the object since it was last replicated. An object's version number indicates the last time an object was modified. It is the database version at the moment of the modification.

ReplicationExample.java: configureReplication

```
1 public static void configureReplication() {  
2     Db4o.configure().generateUUIDs(ConfigScope.GLOBALLY);  
3     Db4o.configure().generateVersionNumbers(ConfigScope.GLOBALLY);  
4 }
```

The above settings can also be applied to a specific class object, which needs to be replicated. This can help to improve the performance if only selected classes need to be replicated:

ReplicationExample.java: configureReplicationPilot

```
1 public static void configureReplicationPilot() {  
2     Db4o.configure().objectClass(Pilot.class).generateUUIDs(true);  
3     Db4o.configure().objectClass(Pilot.class).generateVersionNumbers(true);  
4 }
```

This revision (1) was last Modified 2007-07-08T21:16:16 by Tetyana.

# Simple Example

The following example does a simple replication from a handheld database to the desktop database:

ReplicationExample.java: replicate

```

01  public static void replicate() {
02 |     ObjectContainer desktop=Db4o.openFile(DTFILENAME);
03 |     ObjectContainer handheld=Db4o.openFile(HHFILENAME);
04 |         //      Setup a replication session
05 |     ReplicationSession replication = Replication.begin(handheld, desktop);
06 |
07 |     /*
08 |      * There is no need to replicate all the objects each time.
09 |      * objectsChangedSinceLastReplication methods gives us
10 |      * a list of modified objects
11 |      */
12 |     ObjectSet changed = replication.providerA().objectsChangedSinceLastReplication();
13 |
14 |     while (changed.hasNext()) {
15 |         replication.replicate(changed.next());
16 |     }
17 |
18 |     replication.commit();
19 }
```

We start by opening two ObjectContainers. The next line creates the ReplicationSession. This object contains all of the replication-related logic.

After creating the session, there is an interesting line:

Java:

```
ObjectSet changed = replication.providerA().objectsChangedSinceLastReplication();
```

This line of code will get the provider associated with the first of our sources (the handheld ObjectContainer in this case). Then it finds all of the objects that have been updated or created. The new/modified objects will be returned in an enumerable ObjectSet.

After that comes a simple loop where the resulting objects are replicated one at a time.

The `replication.commit()` call at the end is important. This line will save all of the changes we have made, and end any needed transactions. Forgetting to make this call will result in your replication changes being discarded when your application ends, or your ObjectContainers are closed.

## Simple Example

The `#commit()` calls also mark all objects as replicated. Therefore, changed/new objects that are not replicated in this session will be marked as replicated.

This revision (2) was last Modified 2007-07-08T21:19:24 by Tetyana.

# Bi-Directional Replication

The previous example copied all new or modified objects from the handheld device to the desktop database. What if we want to go the other way? Well, we only have to add one more loop:

ReplicationExample.java: replicateBiDirectional

```

01  public static void replicateBiDirectional() {
02 |     ObjectContainer desktop=Db4o.openFile(DTFILENAME);
03 |     ObjectContainer handheld=Db4o.openFile(HHFILENAME);
04 |     ReplicationSession replication = Replication.begin(handheld, desktop);
05 |     ObjectSet changed = replication.providerA().objectsChangedSinceLastReplication();
06  ↘|     while (changed.hasNext()) {
07 |         replication.replicate(changed.next());
08 |     }
09 |     // Add one more loop for bi-directional replication
10 |     changed = replication.providerB().objectsChangedSinceLastReplication();
11  ↘|     while(changed.hasNext()) {
12 |         replication.replicate(changed.next());
13 |     }
14 |
15 |     replication.commit();
16 }
```

Now our handheld contains all of the new and modified records from our desktop.

Easy, isn't it? Now, if there had been any modifications made to the destination database, the two are now both in sync with each other.

This revision (1) was last Modified 2007-07-08T21:23:26 by Tetyana.

# Selective Replication

What if the handheld doesn't have enough memory to store a complete set of all of the data objects? Well, then we should check, which objects are to be replicated:

ReplicationExample.java: replicatePilots

```

01  public static void replicatePilots() {
02 |     ObjectContainer desktop=Db4o.openFile(DTFILENAME);
03 |     ObjectContainer handheld=Db4o.openFile(HHFILENAME);
04 |     ReplicationSession replication = Replication.begin(handheld, desktop);
05 |     ObjectSet changed = replication.providerB().objectsChangedSinceLastReplication();
06 |
07 |     /* Iterate through the changed objects,
08 |      * check if the name starts with "S" and replicate only those items
09 |      */
10 |     while (changed.hasNext()) {
11 |         if (changed instanceof Pilot) {
12 |             if (((Pilot)changed).getName().startsWith("S")){
13 |                 replication.replicate(changed.next());
14 |             }
15 |         }
16 |     }
17 |
18 |     replication.commit();
19 }
```

Now, only Pilots whose name starts with "S" will be replicated to the handheld database.

This revision (1) was last Modified 2007-07-08T21:24:22 by Tetyana.

# Replication With RDBMS

This topic applies to Java version only

db4o replication can be used to synchronize db4o database with an RDBMS. This is achieved by using [Hibernate](#).

The following topics will show how to configure and run Hibernate enabled replication.

More Reading:

- [Configuration](#)
- [Running DRS](#)
- [Replication Examples](#)
- [Hibernate Replication Internals](#)

This revision (2) was last Modified 2007-07-08T21:38:25 by Tetyana.

# Configuration

## Contents

- hibernate.cfg.xml
- Manual Creation of dRS Tables And Columns
- Hibernate Mapping Files

This topic applies to Java version only

## hibernate.cfg.xml

Hibernate requires a xml configuration file (hibernate.cfg.xml) to run. In order to run dRS with Hibernate, the user has to set some parameters in the configuration file.

### hibernate.cfg.xml

```

01 <hi bernate- configuration>      <sessi on- factory>
02      <!-- Database connection settings -->
03      <property name="hi bernate. connection. driver_ class">oracle.jdbc.driver. OracleDriver</
property>
04      <property name="hi bernate. connection. url ">jdbc: oracle: thin:@ws-peter: 1521: websys</
property>
05      <property name="hi bernate. connection. username">db4o</property>
06      <property name="hi bernate. connection. password">db4o</property>
07      <!-- JDBC connection pool (use the built-in) -->
08      <property name="hi bernate. connection. pool_size">1</property>
09      <!-- SQL dialect -->
10      <property name="hi bernate. dialect">org. hi bernate. dialect. OracleDialect</property>
11      <!-- Echo all executed SQL to stdout -->
12      <property name="hi bernate. show_sql ">false</property>
13      <!-- Update the database schema if out of date -->
14      <property name="hi bernate. hbm2ddl. auto">update</property>
15
16      <property name="hi bernate. jdbc. batch_si ze">0</property>
17  </sessi on- factory>
18 </hi bernate- configuration>
```

For the property "hibernate.connection.pool\_size", dRS requires only 1 connection to the RDBMS, increasing it will not have any effect. "hibernate.jdbc.batch\_size" is set to 0 is for easier debugging. You may increase it to batch SQL statements to potentially increase performance.

It is a MUST that "hibernate.hbm2ddl.auto" be set to "update" because the system will create some extra tables to store the metadata for replication to work properly.

```
<property name="hibernate.hbm2ddl.auto">update</property>
```

## Manual Creation of dRS Tables And Columns

---

In some situations, you may not have the privilege to create or alter tables. You may need to ask your DBA to create the tables for you before using dRS.

You can turn off the automatic creation of dRS tables and columns by changing the "hibernate.hbm2ddl.auto" property to "validate" in hibernate.cfg.xml. By doing so, dRS will not create or alter any tables. Rather, it will check the existence of the dRS tables before starting replication.

If the required dRS tables do not exist, dRS will throw a RuntimeException notifying the user and the replication will halt.

## Hibernate Mapping Files

---

Persisted Objects are objects that the user wants to store to the database, e.g. cars, pilots, purchase orders.

For dRS to operate properly, for each persisted object, the user MUST declare the primary key column of the database table in the hbm.xml mapping file as follow:

```
<id column="typed_id" type="long">
    <generator class="native"/>
</id>
```

- column - The name of the primary key column. The value can be well-formed string . "typed\_id" is recommended.
- type - MUST be "long"
- class - MUST be "native"

The "typed\_id" column stores the id used by Hibernate. It allows dRS to identify a persisted object by invoking org.hibernate.Session#getIdentifier(Object object).

If you do not define getter/setter for property, make default-access="field". default-lazy="false" default-cascade="save-update" is required for replication to work properly. Note, you should not set the cascade style to "delete", otherwise deletion replication will not work.

This revision (2) was last Modified 2007-07-09T14:55:46 by Tetyana.

# Running DRS

This topic applies to Java version only

This section describes how to configure and run dRS. It is crucial to follow the sequence of actions so that dRS can detect new/changed objects and replicate them during replication sessions.

```
// Read or create the Configuration as usual
Configuration cfg = new Configuration().configure("your-hibernate.cfg.xml");
// Let the ReplicationConfigurator adjust the configuration
ReplicationConfigurator.configure(cfg);
// Create the SessionFactory as usual
SessionFactory sessionFactory = cfg.buildSessionFactory();
// Create the Session as usual
Session session = sessionFactory.openSession();
// Let the ReplicationConfigurator install the listeners to the Session
ReplicationConfigurator.install(session, cfg);
//Update objects as usual
Transaction tx = session.beginTransaction();
Pilot john = (Pilot) session.createCriteria(Pilot.class)
```

Some precautions to take into consideration:

1. Do not open more than one dRS replication session against the same RDBMS concurrently. Otherwise data corruption will occur.
2. When dRS is in progress, do not modify the data in the RDBMS by using SQL or Hibernate. Otherwise data corruption will occur.

This revision (2) was last Modified 2007-07-09T14:56:21 by Tetyana.

# Replication Examples

This topic applies to Java version only

The following topics represent some typical replication patterns.

More Reading:

- [Simple Example](#)
- [Collections](#)

This revision (2) was last Modified 2007-07-09T14:57:08 by Tetyana.

# Simple Example

This topic applies to Java version only

This is a one-to-one association example.

The following persistent classes are used:

Helmet.java

```

1 package f1.one_to_one;
2
3 public class Helmet {
4     String model;
5 }
```

Pilot.java

```

1 package f1.one_to_one;
2
3 public class Pilot {
4     String name;
5     Helmet helmet;
6 }
```

A one-to-one association to another persistent class is declared using a one-to-one element:

Pilot.hbm.xml

```

01 <?xml version="1.0"?>
02
03 <!DOCTYPE hibernate-mapping PUBLIC
04     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
05     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
06
07 <hibernate-mapping default-access="field" default-lazy="false"
08                 default-cascade="save-update">
09     <class name="f1.one_to_one.Pilot">
10         <id column="typed_id" type="long">
11             <generator class="foreign">
12                 <param name="property">helmet</param>
```

```

13      </generator>
14    </id>
15    <property name="name"/>
16    <one-to-one name="helmet"/>
17  </class>
18 </hibernate-mapping>

```

Remember to add mappings in hibernate.cfg.xml:

```

<mapping resource="f1/one_to_one/Pilot.hbm.xml"/>

<mapping resource="f1/one_to_one/Helmet.hbm.xml"/>

```

The code to run the replication is provided below:

OneToOneExample.java: main

```

01  public class OneToOneExample {
02    public static void main(String[] args) {
03      new File("OneToOneExample.db4o").delete();
04
05      System.out.println("Running OneToOneExample example.");
06
07      ExtDb4o.configure().generateUUIDs(Integer.MAX_VALUE);
08      ExtDb4o.configure().generateVersionNumbers(Integer.MAX_VALUE);
09
10      ObjectContainer objectContainer = Db4o.openFile("OneToOneExample.db4o");
11
12      //create and save the pilot. Helmet is saved automatically.
13      Helmet helmet = new Helmet();
14      helmet.model = "Robuster";
15
16      Pilot pilot = new Pilot();
17      pilot.name = "John";
18      pilot.helmet = helmet;
19
20      objectContainer.set(pilot);
21      objectContainer.commit();
22
23      // Perform the replication
24      Configuration config = new Configuration().configure("f1/one_to_one/hibernate.cfg.xml");
25
26      ReplicationSession replication = HibernateReplication.begin(objectContainer, config);

```

```
27 |     ObjectSet changed = replication.providerA().objectsChangedSinceLastReplication();
28 |
29 |     // Here helmet is cascaded from pilot and is replicated automatically.
30 |     while (changed.hasNext())
31 |         replication.replicate(changed.next());
32 |
33 |     replication.commit();
34 |     replication.close();
35 |     objectContainer.close();
36 |
37 |     new File("OneToOneExample.db4o").delete();
38 | }
```

This revision (4) was last Modified 2007-07-09T14:58:27 by Tetyana.

# Collections

This topic applies to Java version only

This section covers examples on Collection, including array, Set, List and Map.

As an experienced db4o user, you may know that db4o treats Collection as first class object, which means it assigns unique UUID to each Collection. Hence a Collection can be shared among many owners. This is different to Hibernate's approach, where Collection does not have a unique ID and they cannot be shared among objects.

To bridge this gap, dRS treats Collections as second class objects and does not assign UUIDs to them. When a shared Collection is replicated from db4o to Hibernate using dRS, it is automatically cloned. Each owner of the Collection receives a copy of the Collection. Further modifications to the db4o copy will not be replicated to cloned copies. Therefore, you cannot share Collections if you want to perform RDBMS replications with dRS.

In the following examples, we will use Car as the element in the following examples.

**Car.java**

```

1 package f1.collection;
2
3 public class Car {
4     public String brand;
5     public String model;
6 }
```

**Car.hbm.xml**

```

01 <?xml version="1.0"?>
02
03 <!DOCTYPE hibernate-mapping PUBLIC
04      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
05      "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
06
07 <hibernate-mapping default-access="field" default-lazy="false"
```

```
08          default-cascade="save-update">
09      <class name="f1. collection. Car">
10          <id column="typed_id" type="long">
11              <generator class="native"/>
12          </id>
13          <property name="brand"/>
14          <property name="model"/>
15      </class>
16  </hibernate-mapping>
```

More Reading:

- [Array](#)
- [List](#)
- [Set](#)
- [Map](#)

This revision (2) was last Modified 2007-07-09T14:59:13 by Tetyana.

# Array

This topic applies to Java version only

Hibernate version used with dRS supports one dimensional arrays but does not support multidimensional arrays.

Pilot.java

```

1 package f1.collection.array;
2
3 import f1.collection.Car;
4
5 public class Pilot {
6     String name;
7     Car[] cars;
8 }
```

Pilot.hbm.xml

```

01 <?xml version="1.0"?>
02
03 <!DOCTYPE hibernate-mapping PUBLIC
04      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
05      "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
06
07 <hibernate-mapping default-access="field" default-lazy="false" default-cascade="save-
update">
08   <class name="f1.collection.array.Pilot">
09     <id column="typed_id" type="long">
10       <generator class="native"/>
11     </id>
12
13     <property name="name"/>
14
15     <array name="cars" table="cars">
16       <key column="pilotId"/>
17       <list-index column="sortOrder"/>
18       <one-to-many class="f1.collection.Car"/>
19     </array>
20   </class>
21 </hibernate-mapping>
```

Add Pilot and Car to hibernate.cfg.xml

```
<mapping resource="f1/collection/array/Pilot.hbm.xml"/>

<mapping resource="f1/collection/Car.hbm.xml"/>
```

Save the pilot as usual and start replication:

ArrayExample.java: main

```
01  public class ArrayExample {
02      public static void main(String[] args) {
03          new File("ArrayExample.yap").delete();
04
05          System.out.println("Running Array example.");
06
07          ExtDb4o.configure().generateUUIDs(Integer.MAX_VALUE);
08          ExtDb4o.configure().generateVersionNumbers(Integer.MAX_VALUE);
09
10         ObjectContainer objectContainer = Db4o.openFile("ArrayExample.yap");
11
12         Pilot pilot = new Pilot();
13         pilot.name = "John";
14
15         Car car1 = new Car();
16         car1.brand = "BMW";
17         car1.model = "M3";
18
19         Car car2 = new Car();
20         car2.brand = "Mercedes Benz";
21         car2.model = "S600SL";
22
23         pilot.cars = new Car[]{car1, car2};
24
25         objectContainer.set(pilot);
26         objectContainer.commit();
27
28         Configuration config = new Configuration().configure("f1/collection/array/hibernate.cfg.xml");
29
30         ReplicationSession replication = HibernateReplication.begin(objectContainer, config);
31
32         ObjectSet changed = replication.providerA().objectsChangedSinceLastReplication();
```

```
33 |
34 |     while (changed.hasNext())
35 |         replication.replicate(changed.next());
36 |
37 |     replication.commit();
38 |     replication.close();
39 |     objectContainer.close();
40 |
41 |     new File("ArrayExample.yap").delete();
42 }
```

This revision (3) was last Modified 2007-07-09T15:01:03 by Tetyana.

# List

This topic applies to Java version only

Similar to array, you can replicate a List of Cars.

Pilot.java

```

1 package f1.collection.list;
2
3 import java.util.List;
4
5 public class Pilot {
6     String name;
7     List cars;
8 }
```

Map the List using the list tag in Pilot.hbm.xml

Pilot.hbm.xml

```

01 <?xml version="1.0"?>
02
03 <!DOCTYPE hibernate-mapping PUBLIC
04      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
05      "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
06
07 <hibernate-mapping default-access="field" default-lazy="false" default-cascade="save-
update">
08   <class name="f1.collection.list.Pilot">
09     <id column="typed_id" type="long">
10       <generator class="native"/>
11     </id>
12
13     <property name="name"/>
14
15     <list name="cars" table="cars">
16       <key column="pilotId"/>
17       <list-index column="sortOrder"/>
18       <one-to-many class="f1.collection.Car"/>
19     </list>
20   </class>
```

## 21 &lt;/hibernate-mapping&gt;

Replicate the pilot:

ListExample.java: main

```

01  public class ListExample {
02      public static void main(String[] args) {
03          new File("ListExample.yap").delete();
04
05          System.out.println("Running List example.");
06
07          ExtDb4o.configure().generateUUIDs(Integer.MAX_VALUE);
08          ExtDb4o.configure().generateVersionNumbers(Integer.MAX_VALUE);
09
10          ObjectContainer objectContainer = Db4o.openFile("ListExample.yap");
11
12          Pilot pilot = new Pilot();
13          pilot.name = "John";
14
15          Car car1 = new Car();
16          car1.brand = "BMW";
17          car1.model = "M3";
18
19          Car car2 = new Car();
20          car2.brand = "Mercedes Benz";
21          car2.model = "S600SL";
22
23          pilot.cars = new ArrayList();
24          pilot.cars.add(car1);
25          pilot.cars.add(car2);
26
27          objectContainer.set(pilot);
28          objectContainer.commit();
29
30          Configuration config = new Configuration().configure("f1/collection/list/hibernate.cfg.xml");
31
32          ReplicationSession replication = HibernateReplication.begin(objectContainer, config);
33
34          ObjectSet changed = replication.provider().objectsChangedSinceLastReplication();

```

```
35 |
36 |     while (changed.hasNext())
37 |         replication.replicate(changed.next());
38 |
39 |     replication.commit();
40 |     replication.close();
41 |     objectContainer.close();
42 |
43 |     new File("ListExample.yap").delete();
44 }
```

This revision (2) was last Modified 2007-07-09T15:02:04 by Tetyana.

# Set

This topic applies to Java version only

Replicating a Set of objects is simple and is similar to the [List](#) example.

Pilot.java

```

1 package f1.collection.set;
2
3 import java.util.Set;
4
5 public class Pilot {
6     String name;
7     Set cars;
8 }
```

Use the set tag.

Pilot.hbm.xml

```

01 <?xml version="1.0"?>
02
03 <!DOCTYPE hibernate-mapping PUBLIC
04     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
05     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
06
07 <hibernate-mapping default-access="field" default-lazy="false" default-cascade="save-
update">
08     <class name="f1.collection.set.Pilot">
09         <id column="typed_id" type="long">
10             <generator class="native"/>
11         </id>
12
13         <property name="name"/>
14
15         <set name="cars" table="cars">
16             <key column="pilotId"/>
17             <one-to-many class="f1.collection.Car"/>
18         </set>
19     </class>
```

## 20 </hibernate-mapping>

Do the replication.

SetExample.java: main

```
01 public class SetExample {
02     public static void main(String[] args) {
03         new File("SetExample.yap").delete();
04
05         System.out.println("Running Set example.");
06
07         ExtDb4o.configure().generateUUIDs(Integer.MAX_VALUE);
08         ExtDb4o.configure().generateVersionNumbers(Integer.MAX_VALUE);
09
10         ObjectContainer objectContainer = Db4o.openFile("SetExample.yap");
11
12         Pilot pilot = new Pilot();
13         pilot.name = "John";
14
15         Car car1 = new Car();
16         car1.brand = "BMW";
17         car1.model = "M3";
18
19         Car car2 = new Car();
20         car2.brand = "Mercedes Benz";
21         car2.model = "S600SL";
22
23         pilot.cars = new HashSet();
24         pilot.cars.add(car1);
25         pilot.cars.add(car2);
26
27         objectContainer.set(pilot);
28         objectContainer.commit();
29
30         Configuration config = new Configuration().configure("f1/collection/set/hibernate.cfg.xml");
31
32         ReplicationSession replication = HibernateReplication.begin(objectContainer, config);
33         ObjectSet changed = replication.providerA().objectsChangedSinceLastReplication();
34 }
```

```
35 |     while (changed.hasNext())  
36 |         replication.replicate(changed.next());  
37 |  
38 |     replication.commit();  
39 |     replication.close();  
40 |     objectContainer.close();  
41 |  
42 |     new File("SetExample.yap").delete();  
43 | }
```

This revision (2) was last Modified 2007-07-09T15:02:50 by Tetyana.

# Map

This topic applies to Java version only

Replication supports replicating a Map of objects.

## Pilot.java

```

1 package f1. collection.map;
2
3 import java.util.Map;
4
5 public class Pilot {
6     String name;
7     Map cars;
8 }
```

Use the map element to define the Map in the hbm file:

## Pilot.hbm.xml

```

01 <?xml version="1.0"?>
02
03 <!DOCTYPE hibernate-mapping PUBLIC
04     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
05     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
06
07 <hibernate-mapping default-access="field" default-lazy="false" default-cascade="save-
update">
08     <class name="f1. collection.map.Pilot">
09         <id column="typed_id" type="long">
10             <generator class="native"/>
11         </id>
12
13         <property name="name"/>
14
15         <map name="cars" table="cars">
16             <key column="pilotId"/>
17             <map-key type="string"/>
18             <one-to-many class="f1. collection.Car"/>
19         </map>
```

```
20 </class>
21 </hibernate-mapping>
```

Do the replication:

MapExample.java: main

```
01 public class MapExample {
02     public static void main(String[] args) {
03         new File("MapExample.yap").delete();
04
05         System.out.println("Running Map example.");
06
07         ExtDb4o.configure().generateUUIDs(Integer.MAX_VALUE);
08         ExtDb4o.configure().generateVersionNumbers(Integer.MAX_VALUE);
09
10         ObjectContainer objectContainer = Db4o.openFile("MapExample.yap");
11
12         Pilot pilot = new Pilot();
13         pilot.name = "John";
14
15         Car car1 = new Car();
16         car1.brand = "BMW";
17         car1.model = "M3";
18
19         Car car2 = new Car();
20         car2.brand = "Mercedes Benz";
21         car2.model = "S600SL";
22
23         pilot.cars = new HashMap();
24         pilot.cars.put("car1", car1);
25         pilot.cars.put("car2", car2);
26
27         objectContainer.set(pilot);
28         objectContainer.commit();
29
30         Configuration config = new Configuration().configure("f1/collection/map/hibernate.cfg.xml");
31
32         ReplicationSession replication = HibernateReplication.begin(objectContainer, config);
33         ObjectSet changed = replication.providerA().objectsChangedSinceLastReplication();
```

```
34 |
35 |     while (changed.hasNext())
36 |         replication.replicate(changed.next());
37 |
38 |     replication.commit();
39 |     replication.close();
40 |     objectContainer.close();
41 |
42 |     new File("MapExample.yap").delete();
43 }
```

This revision (3) was last Modified 2007-07-09T15:04:55 by Tetyana.

# Hibernate Replication Internals

## Contents

- [ProviderSignature, MySignature and PeerSignature](#)
- [Record](#)
- [UUID](#)
- [ObjectReference](#)
- [List of dRS tables](#)
- [drs\\_providers](#)
- [drs\\_history](#)
- [drs\\_objects](#)

This topic applies to Java version only

So far we have seen that dRS allows you to replicate objects between db4o and relational database. You maybe curious about how dRS keeps track of the identity of objects in relational database and how dRS knows which objects are changed since the last round of replication. Read on and you will see how dRS does that.

dRS internal objects keep information used by replication. Each internal object is associated with a Hibernate mapping file (.hbm.xml). Hibernate reads these files and understands how to store / retrieve these internal objects to / from the RDBMS. Each type of internal object maps to one table in RDBMS. If such table does not exist, Hibernate creates it automatically.

## ProviderSignature, MySignature and PeerSignature

---

ProviderSignature uniquely identifies a ReplicationProvider. MySignature and PeerSignature are the subclasses of ProviderSignature. A HibernateReplicationProvider has a MySignature to serve as its own identity. PeerSignature identifies the peer ReplicationProvider during a ReplicationSession.

## Record

---

Record contains the version of the RDBMS during a ReplicationSession. Near the end of a ReplicationSession, two ReplicationProviders synchronize their versions.

Record allows dRS to detect changed objects. dRS detects changed objects by comparing the version of an object (v) with the maximum version of all Records (m). An object is updated if  $v > m$ .

## UUID

---

UUID uniquely identifies a persisted object in dRS.

Each persisted object is identified by a "typed\_id" in Hibernate. This "typed\_id" is unique only with its type of that object (i.e. A car has an "typed\_id" of 1534, a Pilot can also have an "typed\_id" of 1534) and within the current RDBMS.

How do we identify "a Pilot that is originated from Oracle instance pi2763" ? To do so, we need two parameters:

1. an id that is unique across types
2. association between this id and the ProviderSignature of the RDBMS (The RDBMS that owns this object)

```
class UUID {
    long longPart;
    ProviderSignature provider;
}
```

Collectively, 1 and 2 forms the "UUID".

## ObjectReference

---

ObjectReference contains the UUID and the version of a persisted object. It also contains the className and the typed\_id of that persisted object.

UUID forms an 1 to 1 relationship with {className, typedId}.

```
class ObjectReference {
    String className;
    long typedId;
    Uuid uuid;
    long version;
}
```

## List of dRS tables

---

### drs\_providers

---

Column	Type	Function
id	long	synthetic, auto-increment primary key
is_my_sig	char(1)	't' if MySignature, 'f' if PeerSignature
signature	binary	holds the unique identifier - byte array
created	long	legacy field used by pre-dRS db4o replication code

## drs\_history

---

Column	Type	Function
provider_id	long	primary key, same as the PK of a PeerSignature
time	long	the version of the RDBMS during a ReplicationSession

## drs\_objects

---

Column	Type	Function
id	long	synthetic, auto-increment primary key
created	long	the UUID long part of this ObjectReference
provider_id	long	specifies the originating ReplicationProvider of this ObjectReference
class_name	varchar	the type of the referenced object
typed_id	long	the id used by Hibernate, which is only unique within its type
modified	long	the version of the referenced object

This revision (3) was last Modified 2007-07-09T14:57:51 by Tetyana.

# Advanced Replication Strategies

This chapter contains information on some advanced dRS strategies.

More Reading:

- [Events](#)
- [Deleted Objects Replication](#)

This revision (1) was last Modified 2007-07-09T07:56:49 by Tetyana.

# Events

Replication events provide you with an ability to add your custom logic into the replication process. Below you will find some of the usage examples. For more information see ReplicationEvent API.

More Reading:

- [Conflict Resolution](#)

This revision (1) was last Modified 2007-07-09T08:12:23 by Tetyana.

# Conflict Resolution

The most popular usage of replication callback events is conflict resolution. When an object was changed in both replicating databases you must have a strategy to decide, which value must be final. In the simplest case one of the databases is made "dominant" and the changes in this database always override other changes. The following example demonstrates this behavior:

EventsExample.java: conflictResolutionExample

```

01  private static void conflictResolutionExample() {
02 |     Db4o. configure(). generateUUIDs(ConfigScope.GLOBALLY);
03 |     Db4o. configure(). generateVersionNumbers(ConfigScope.GLOBALLY);
04 |
05 |     // Open databases
06 |     ObjectContainer desktop = Db4o.openFile("desktop.db4o");
07 |     ObjectContainer handheld = Db4o.openFile("handheld.db4o");
08 |
09 |     Pilot pilot = new Pilot("Scott Felton", 200);
10 |     handheld.set(pilot);
11 |     handheld.commit();
12 |     /* Clean the reference cache to make sure that objects in memory
13 |      * won't interfere
14 |      */
15 |     handheld.ext().refresh(Pilot.class, Integer.MAX_VALUE);
16 |
17 |     /* Replicate changes from handheld to desktop
18 |      * Note, that only objects replicated from one database to another will
19 |      * be treated as the same. If you will create an object and save it to both
20 |      * databases, dRS will count them as 2 different objects with identical
21 |      * fields.
22 |      */
23 |     ReplicationSession replication = Replication.begin(handheld, desktop);
24 |     ObjectSet changedObjects = replication.providerA().objectsChangedSinceLastReplication();
25 |     while (changedObjects.hasNext())
26 |         replication.replicate(changedObjects.next());
27 |     replication.commit();
28 |
29 |     // change object on the handheld
30 |     pilot = (Pilot) handheld.query(Pilot.class).next();
31 |     pilot.setName("S. Felton");

```

```

32 |     handheld.set(pilot);
33 |     handheld.commit();
34 |
35 |     // change object on the desktop
36 |     pilot = (Pilot)desktop.query(Pilot.class).next();
37 |     pilot.setName("Scott");
38 |     desktop.set(pilot);
39 |     desktop.commit();
40 |
41 |     /* The replication will face a conflict: Pilot object was changed on the
42 |      * handheld and on the desktop.
43 |      * To resolve this conflict we will add an event handler, which makes
44 |      * desktop changes dominating.
45 |      */
46 |     ReplicationEventListener listener;
47 |     listener = new ReplicationEventListener() {
48 |         public void onReplicate(ReplicationEvent event) {
49 |             if (event.isConflict()) {
50 |                 ObjectState chosenObjectState = event.stateInProviderB();
51 |                 event.overrideWith(chosenObjectState);
52 |             }
53 |         }
54 |     };
55 |
56 |     replication = Replication.begin(handheld, desktop, listener);
57 |
58 |     //The state of the desktop after the replication should not change, as it dominates
59 |     changedObjects = replication.providerA().objectsChangedSinceLastReplication();
60 |     while (changedObjects.hasNext())
61 |         replication.replicate(changedObjects.next());
62 |
63 |     //Commit
64 |     replication.commit();
65 |     replication.close();
66 |
67 |     // Check what we've got on the desktop
68 |     ObjectSet result = desktop.query(Pilot.class);
69 |     System.out.println(result.size());
70 |     while (result.hasNext()) {
71 |         System.out.println(result.next());
72 |     }

```

```
73 |     handheld.close();  
74 |     desktop.close();  
75 |  
76 |     new File("handheld.db4o").delete();  
77 |     new File("desktop.db4o").delete();  
78 |  
79 L }
```

This revision (5) was last Modified 2007-07-09T19:39:39 by Tetyana.

# Deleted Objects Replication

In addition to replicating changed/new objects, dRS is able to replicate deletions of objects. When an object is deleted since last replication in one database and you would like to replicate these changes to another database you can use the following method to do this:

Java:

```
replication .replicateDeletions(Car.class);
```

```
[/filter]
```

dRS traverses every Car object in both providers. For instance, if a deletion is found in one provider, the deletion will be replicated to the other provider. During the traversal replication events will be generated and can be used as usual. By default, in a case of a conflict the deletion will prevail. You can choose the counterpart of the deleted object to prevail using the event.

Note, that the deletions of a Parent will not be cascaded to child objects. For example, if a Car contains a child object, e.g. Pilot, Pilot will not be traversed and the deletions of Pilot will not be replicated.

This revision (3) was last Modified 2007-07-09T08:31:56 by Tetyana.

# Deployment Instructions

## Contents

- [dRS-Java Deployment](#)

When you deploy dRS with your application, you must first ensure that the [requirements for db4o deployment](#) are met.

## dRS-Java Deployment

---

In order to deploy dRS in the most simple db4o-to-db4o replication scenario you will only need **dRS-x.x-core.jar**.

For support of replication to relational databases you will need **hibernate3.jar** and its dependencies:

- antlr-2.7.6.jar
- asm.jar
- asm-attrs.jar
- cglib-2.1.3.jar
- commons-collections-2.1.1.jar
- commons-logging-1.0.4.jar
- dom4j-1.6.1.jar
- ehcache-1.2.jar
- jta.jar
- log4j-1.2.13.jar
- xerces-2.6.2.jar

Please, refer to [hibernate](#) documentation for an explanation of these dependencies.

If you are going to use dRS with HSQLDB you will need hsqldb-1.8.0.7.jar in addition to hibernate.

This revision (4) was last Modified 2007-10-01T19:03:25 by Tetyana.

# License

## Contents

- [dRS](#)
- [Bundled 3rd Party Licenses](#)

## dRS

---

[db4objects Inc.](#) supplies the db4o Replication System (dRS) under the General Public License (GPL).

Under the GPL dRS is free to be used:

- for development,
- in-house as long as no deployment to third parties takes place,
- together with works that are placed under the GPL themselves.

You should have received a copy of the GPL in the file dRS.license.html together with the dRS distribution.

## Bundled 3rd Party Licenses

---

The dRS distribution comes with several 3rd party libraries, located in the /lib/ subdirectory together with the respective license files.

This revision (1) was last Modified 2007-07-08T22:10:46 by Tetyana.

# Db4o Testing Framework

db4ounit is a minimal xUnit (JUnit, NUnit) style testing framework. Db4ounit framework was created to fulfill the following requirements:

- the core tests should be run against JDK1.1
- it should be possible to automatically convert test cases from Java to .NET.

db4ounit design deviates from vanilla xUnit in some respect, but if you know xUnit, db4ounit should look very familiar.

db4ounit itself is completely agnostic of db4o, but there is the db4ounit.extensions module which provides a base class for db4o specific test cases with different fixtures, etc.

Db4ounit and db4ounit.extensions are supplied as a source code for both java and .NET. Java version also comes with a compiled library: db4o-6.0-db4ounit.jar, which allows you to run your tests from a separate package.

If you've found a bug and want to supply a test case to help db4o to fix the issue quickly, the best option would be to supply your code in the java db4ounit format. This format allows very easy integration of a new test case into db4o test suite: only copy/paste is required to put your test class code into the framework using Eclipse.

More Reading:

- [Db4ounit Methods](#)
- [Creating A Sample Test](#)
- [Running The Tests](#)

This revision (2) was last Modified 2007-01-29T16:51:18 by Tetyana.

# Db4ounit Methods

## Contents

- [AbstractDb4oTestCase](#)
  - [Methods for working with a database:](#)
  - [Various methods to work with persistent objects:](#)
  - [Methods for running tests:](#)
- [Db4ounit.Assert](#)
- [FrameworkTestCase](#)

Let's look through the basic API , which will help you to build your own test. This document is not a complete API reference and its intention is to give you a general idea of the methods usage and availability.

## AbstractDb4oTestCase

---

AbstractDb4oTestCase is a base class for creating test cases.

private transient Db4oFixture \_fixture; - determines an environment for the test execution and gives an access to the test database. The environment can be local (derived from AbstractSoloDb4oFixture) or client/server (AbstractClientServerDb4oFixture).

You can always access the fixture from your test class using

Java:

```
public void fixture(Db4oFixture fixture)
```

## Methods for working with a database:

---

Java:

```
public ExtObjectContainer db()
```

Returns an instanse of object container for the current environment.

Java:

```
protected void reopen() throws Exception
```

This function will close the database and open it again. It also performs an implicit commit on close.

Java:

```
protected Reflector reflector()
```

Returns current reflector.

Java:

```
protected Transaction trans()
```

```
protected Transaction systemTrans()
```

```
protected Transaction newTransaction()
```

Methods to get transaction object for the current environment.

## Various methods to work with persistent objects:

---

Java:

```
protected Query newQuery()
```

```
protected Query newQuery(Class clazz)
```

```
protected Query newQuery(Transaction transaction, Class clazz)
```

```
protected Query newQuery(Transaction transaction)
```

Checks if only one object of a class is stored in the database

Java:

```
protected int countOccurrences(Class clazz)
```

Returns the amount of objects of the specified class in the database.

Java:

```
protected void foreach(Class clazz, Visitor4 visitor)
```

This method goes through the ObjectSet of the specified class objects in the database calling Visitor4.visit() method. Visitor4 is an interface specifiing a visit method:

Java:

```
public void visit(Object obj);
```

Java:

```
protected void deleteAll(Class clazz)
```

Deletes all the instances of the specified class in the database.

Java:

```
protected ReflectClass reflectClass(Class clazz)
```

Returns a ReflectClass instance for the specified class.

Java:

```
protected void indexField(Configuration config, Class clazz, String fieldName)
```

Adds field index into specified configuration.

Java:

```
public final void setUp() throws Exception
```

This method:

- deletes the used database;
- configures and opens a new one (see Configure method).
- Calls db4oSetupBeforeStore
- Calls store()
- Commits and reopens the database

- Calls db4oSetupAfterStore

More details about the mentioned above methods:

Java:

```
protected void configure(Configuration config)
```

Use this method to create your custom configuration for a test. Config parameter is the current default test configuration, which can be modified.

Java:

```
protected void db4oSetupBeforeStore() throws Exception
```

This method is a placeholder for any custom setup actions that need to be taken before filling up the database with objects.

Java:

```
protected void store() throws Exception { }
```

This method is supplied for creating and storing the objects, which you are going to use for your test.

Java:

```
protected void db4oSetupAfterStore() throws Exception
```

This method is a placeholder for any custom setup actions that need to be taken after the database is filled up with objects.

## Methods for running tests:

---

Java:

```
public int runSoloAndClientServer()
```

Will run the test in both modes

Java:

```
public int runSolo()
```

Only local mode.

Java:

```
public int runClientServer()
```

## Db4ounit.Assert

---

Db4ounit.Assert class - provides a variety of methods for controlling code execution. Some of the methods are presented below. For more information please refer to the source code.

Java:

```
public static void expect(Class exception, CodeBlock block)
```

This method runs a specified method (block parameter) and throws an exception if the block runs without any exception.

Java:

```
public static void isTrue(boolean condition)
```

```
public static void isTrue(boolean condition, String msg)
```

This method checks the condition and throws an exception if the condition is false. Msg parameter can be used to customize exception message.

Java:

```
public static void areEqual(Object expected, Object actual)
```

Checks if the supplied parameters are equal and throws an exception otherwise.

Similar methods are provided for null, lesser, greater and other checkings, please refer to Assert class code for full information.

# FrameworkTestCase

---

FrameworkTestCase class provides methods to run your test suite and check if its results.

Java:

```
public static void runTestAndExpect(Test test, int expFailures)
```

This method will run the test specified and throw an exception if the number of expected failures (expFailures parameter) is not equal to the number of experienced failures.

For more information please refer to the source code of FrameworkTestCase class.

This revision (11) was last Modified 2007-01-29T18:30:55 by Tetyana.

# Creating A Sample Test

Let's create a simple test case to check if the cascade on update setting is working as expected.

We will use a simple linked list class:

CascadeOnUpdate.java: Atom

```

01  public static class Atom {
02  |
03  |     public Atom child;
04  |     public String name;
05  |
06  |     public Atom() {
07  |         }
08  |
09  |     public Atom(Atom child) {
10 |         this.child = child;
11 |     }
12  |
13  |     public Atom(String name) {
14 |         this.name = name;
15 |     }
16  |
17  |     public Atom(Atom child, String name) {
18 |         this(child);
19 |         this.name = name;
20 |     }
21 }
```

Custom configuration should be added to configure method:

CascadeOnUpdate.java: configure

```

1   protected void configure(Configuration conf) {
2 |     conf.objectClass(Atom.class).cascadeOnUpdate(false);
3  }
```

To prepare the database we will need to store some Atom objects:

CascadeOnUpdate.java: store

```

1   protected void store() {
2 |     Atom atom = new Atom(new Atom(new Atom("storedGrandChild"), "storedChild"), "parent");
3 |     db().set(atom);
4 |     db().commit();
5  }
```

Now we are ready to test the expected behavior:

CascadeOnUpdate.java: test

```

01   public void test() throws Exception {
02 |     Query q = newQuery(Atom.class);
03 |     q.descend("name").constraint("parent");
04 |     ObjectSet objectSet = q.execute();
05 |     Atom atom = null;
06   while (objectSet.hasNext()){
07 |         // update child objects
08 |         atom = (Atom) objectSet.next();
09 |         ((Atom) atom.child).name = "updated";
10 |         ((Atom) atom.child).child.name = "notUpdated";
11 |         // store the parent object
12 |         db().set(atom);
13 |     }
14 |
15 |     // commit and refresh to make sure that the changes are saved
16 |     // and reference cash is refreshed
```

```
17 |     db().commit();
18 |     db().refresh(atom, Integer.MAX_VALUE);
19 |
20 |
21 |     q = newQuery(Atom.class);
22 |     q.descend("name").constraint("parent");
23 |     objectSet = q.execute();
24 |     while(objectSet.hasNext()){
25 |         atom = (Atom) objectSet.next();
26 |         Atom child = (Atom) atom.child;
27 |         // check if the child objects were updated
28 |         Assert.AreEqual("updated", child.name);
29 |         Assert.AreNotEqual("updated", child.child.name);
30 |     }
31 }
```

This revision (1) was last Modified 2007-01-29T18:36:18 by Tetyana.

# Running The Tests

The TestSuite can be built:

- from the list of the arguments supplied to Db4oUnitTestMain class (only for Java version) .
- using Db4oTestSuiteBuilder to specify test classes to be run.

If you are using java version you can test your class immediately (assuming the class is created in TestDb4oUnit package):

```
java -cp db4o-6.0-db4ounit.jar;your_class_folder db4ounit.UnitTestingMain your_class_package.AssertTestCase
```

The second option is to list the test class in Db4oTestSuiteBuilder argument. You can use db4ounit.extensions.tests.AllTests class. Just add the following method:

```
AllTests.java: testCascadeOnUpdate
```

```
1 public void testCascadeOnUpdate() {  
2     Db4oFixture fixture = new ExcludingInMemoryFixture(new IndependentConfigurationSource());  
3     TestSuite suite = new Db4oTestSuiteBuilder(fixture, new Class[] { CascadeOnUpdate.  
class }).build();  
4     FrameworkTestCase.runTestAndExpect(suite, 0);  
5 }
```

Now running the AllTests class will include your custom class too.  
This revision (3) was last Modified 2007-01-29T18:48:54 by Tetyana.

# Object Manager For db4o

The db4o Object Manager is a GUI tool to browse and query the contents of any db4o database file. Object Manager currently provides the following features:

- Powerful ad-hoc queries
- Efficient handling of large data sets
- Table view for quick scrolling through large data sets
- Tree view for drilling down through an object graph
- Access to internal database information including stored classes, data size and indexes
- Management functions including Backup and Defragment

You can check the project development flow in [OM Jira-project](#). Please, report there if any problems are encountered.

[Online version](#) of this document can be found on the [developer website](#).

More Reading:

- [Installation](#)
- [Object Manager Specifics](#)
- [Object Manager Tour](#)

This revision (7) was last Modified 2007-08-11T14:55:25 by Tetyana.

# Installation

Object Manager has to be downloaded separately from the main db4o distributions. Please visit the [db4o Download Center](#) and choose the installation appropriate for the version of db4o you are using (since db4o version 6.0, db4o and ObjectManager versions are synchronized, i.e. you should use ObjectManager 6.1 with db4o 6.1). Object Manager runs on Windows, Linux and Mac OS X.

Once you have downloaded the appropriate Object Manager build, create a folder called Object Manager in any location of your choice and unpack the downloaded zip file there.

The following instructions explain how to start Object Manager in different environments. It is assumed that you already have a Sun Java Virtual Machine 1.5 or later installed and that your PATH variable lists the directory containing the java binary. This should be set when you've first installed the Java Runtime Environment. You can download the latest Java Runtime Environment here: <http://java.com/getjava>

## Windows

From Windows Explorer: Double-click the objectmanager.jar

From Command Line: Run "java -jar objectmanager.jar"

## Linux

From UI: Double click the objectmanager.jar

From Command Line: Run "java -jar objectmanager.jar"

This revision (4) was last Modified 2007-04-06T06:58:19 by Tetyana.

# Object Manager Specifics

## Contents

- [Java enumerations](#)

Object Manager can operate on any java or .NET db4o database, because it does not need class definitions to instantiate database objects. Instead of real classes ObjectManager in fact instantiates [Generic Objects](#), which hold a database object information in an array of fields. This approach is very powerful, however it introduces some limitations.

## Java enumerations

---

Java enumerations (`java.lang.Enum`) hold the information in `name` and `ordinal` fields. However, when an enumeration is used in a persistent object it is not really necessary to save the whole enumeration to the database, as it will be available from the classloader in runtime. That is why db4o only saves the actual enumeration values together with the object, and the values are bind to the enumeration when the object is instantiated. For more information about the enumerations see [Static Fields And Enums](#).

This approach allows to save database space and proves to be efficient. However, if you are looking at an enumeration object using ObjectManager, the enumeration will be constructed as a generic object, therefore the binding procedure won't be triggered. This will affect the following:

- enumeration type fields will show the class name instead of the actual values, for example "com.examples.Colors" instead of "Red", "Green" etc.;
- you won't be able to constrain queries on the enumeration field values;
- you won't see "name" and "ordinal" field values querying for the enumeration type data.

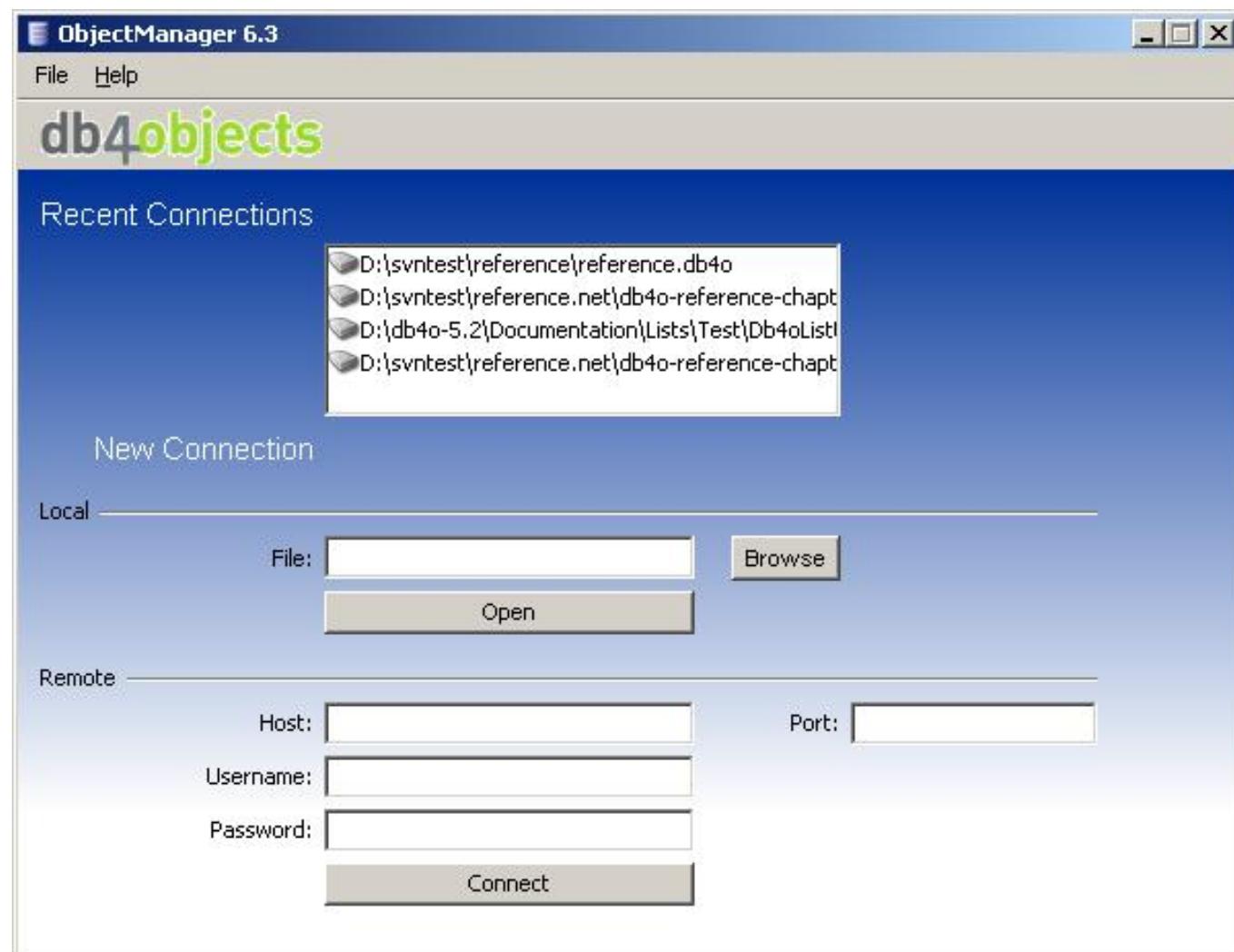
This revision (1) was last Modified 2007-08-11T14:57:01 by Tetyana.

# Object Manager Tour

## Contents

- [Stored Classes Tree](#)
- [Querying for objects](#)
  - [Query Language](#)
  - [Query Results](#)
- [Database Management](#)

Upon starting Object Manager, you will see the Connection Manager:



At this point, you can either:

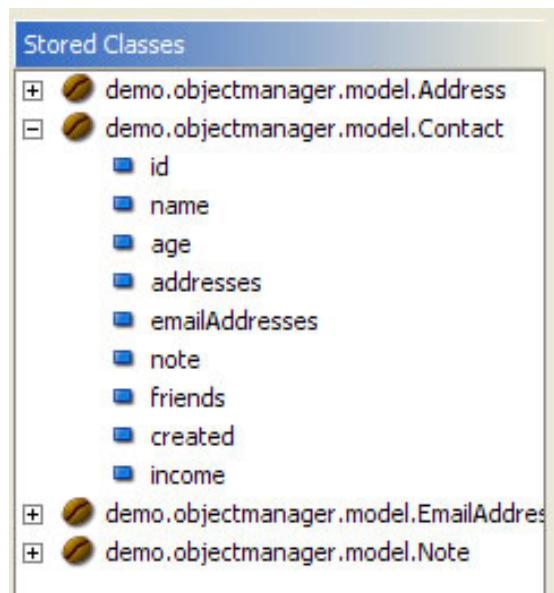
- Open a db4o database file
- Open a connection to a db4o database server

In order to open a db4o database file, simply click Browse and choose the db4o database file to open, then click Connect.

If you just want to try ObjectManager, you can create a demo database by clicking File - Create Demo Db. This will create and open a database for you to use.

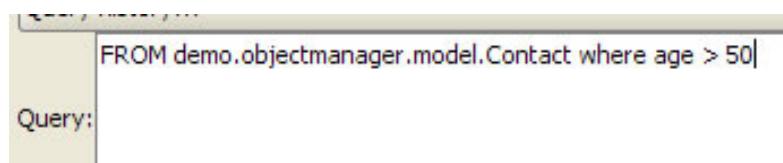
## Stored Classes Tree

Once you click connect, the main ObjectManager window will open. All of the classes that have been stored will be displayed on the left in a tree:



## Querying for objects

To view the objects stored in your db4o database, you must query for it. The query editor is at the top of the main window and looks like this:



To quickly see all of the objects of a certain class, just double click the class name in the stored classes tree. This will automatically build your query for you, and you just have to click the Submit button to the right of the query text area.

## Query Language

The Object Manager query language, Sql4o, is based on the SQL query language, so if you know SQL, you should have no problems with Sql4o.

Syntax:

[SELECT *select\_expr*, ...]

FROM *object\_type*

[WHERE *where\_condition*]

For example, to query all Contacts in the Demo Db:

FROM demo.objectmanager.model.Contact

To query all contacts with age 50.

FROM demo.objectmanager.model.Contact where age 50

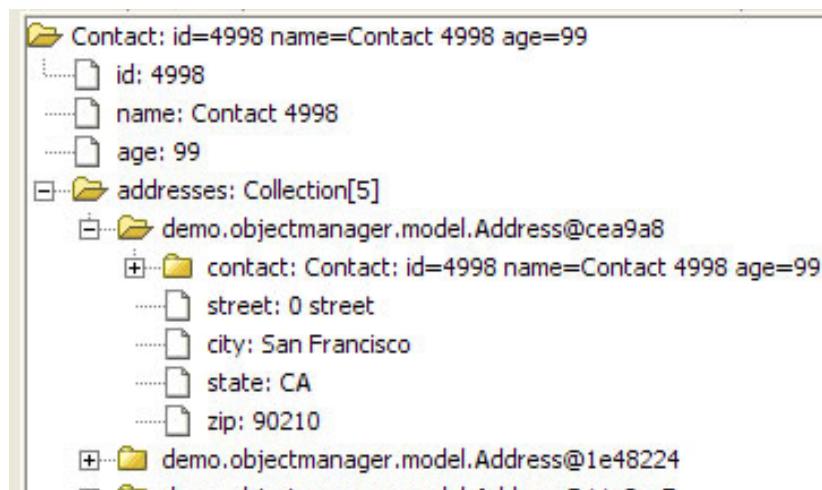
## Query Results

After executing a query, you will see the query results in table:

Row	id	name	age	addresses	emailAddresses	no
0	4999	Contact 4999		100 Collection: 5 items	Collection: 10 items	demo.obje
1	4998	Contact 4998		99 Collection: 5 items	Collection: 10 items	demo.obje
2	4997	Contact 4997		98 Collection: 5 items	Collection: 10 items	demo.obje
3	4996	Contact 4996		97 Collection: 5 items	Collection: 10 items	demo.obje
4	4995	Contact 4995		96 Collection: 5 items	Collection: 10 items	demo.obje
5	4994	Contact 4994		95 Collection: 5 items	Collection: 10 items	demo.obje

From this table, you can edit all primitive fields, Dates, and Strings by double clicking on a cell.

You can also open an object up in a tree view so that you can descend further into the object:



You can also edit fields from this view.

## Database Management

The following management features are currently available:

- Backup
- Defragment
- Refactoring (Schema Evolution)

Back-up is available from "Manage-Backup..." menu. It will ask you for a location to store the back-up, once that is chosen, the back-up copy of your database will be created. The backed-up copy is a fully functional database and can be used by

replacing your current database.

Defragment is available from "Manage-Defragment" menu or through "Ctrl-Shift-D" shortcut. This will execute the db4o defragmentation process on your database. For more information about the defragment see [Defragment](#).

[Refactoring](#) allows you to change the structure of persistent classes in a db4o database. Object Manager helps you to automate the simplest refactoring: renaming of class fields.

In order to rename a class field double-click the class in a "Stored Classes" tree - class view will open on the right panel. Click "Edit" button at the bottom and change field names as necessary.

Fields			
Name	Type	Indexed?	
id	java.lang.Integer	<input type="checkbox"/>	▲
name	java.lang.String	<input type="checkbox"/>	
age	int	<input type="checkbox"/>	
addresses	java.util.List	<input type="checkbox"/>	
emailAddresses	java.util.List	<input type="checkbox"/>	
noteNew	demo.objectmanager.model.Note	<input checked="" type="checkbox"/>	
friends	java.util.List	<input type="checkbox"/>	
created	java.util.Date	<input type="checkbox"/>	
income	double	<input type="checkbox"/>	
birthDate	java.util.Date	<input type="checkbox"/>	
gender	char	<input type="checkbox"/>	▼

**Stop Editing**

When the renaming is done, click "Stop Editing" button at the bottom. Remember to change your application classes accordingly.

This revision (12) was last Modified 2007-08-11T14:52:50 by Tetyana.

# Working With Source Code

db4o is an open-source project. The source is available for reviewing, modifying for own needs or contributing your modifications. You can use the [source code from the downloaded distribution](#) package or you may use our [SVN repository](#) to get the latest modifications.

SVN repository can be found at: <https://source.db4o.com/db4o/trunk/>

More Reading:

- [Using The Sources From Db4o Distribution](#)
- [Using The Repository](#)
- [Db4o Directory Structure](#)
- [Configuring Java System Libraries](#)
- [Building Java Version](#)
- [Building .NET Version](#)
- [Testing Db4o](#)
- [Object Manager Sources](#)
- [Patch Submission](#)
- [Project Dependencies](#)
- [Coding Style](#)

This revision (11) was last Modified 2008-01-09T04:33:28 by German Viscuso.

# Using The Sources From Db4o Distribution

The source code is available in /src folder in your db4o distribution.

In Java version the following projects are available:

## bloat

[Byte-code optimization library](#)

## db4o.instrumentation

Bytecode instrumentation layer on top of bloat

## db4o osgi

db4o OSGI bundle.

## db4o osgi test

Tests for db4o OSGI bundle.

## db4oj

db4o core.

## db4oj.tests

collection of db4o regression tests.

## db4ojdk1.2

db4o sources for JDK1.2-1.4

## db4ojdk5

db4o sources for JDK 5-6

## db4otaj

Transparent Activation instrumentation library

## db4otools

Bytecode instrumentation tools

## db4onqopt

db4o native query optimization library

## db4ounit

[unit test framework](#) for db4o.

## db4ounit.extensions

Db4oUnit extensions for [testing db4o](#).

We recommend [Eclipse](#) to work with the sources.

First you will need to [configure Java System Libraries](#). Having done that use the following steps to import the source project into the development environment:

- open a workspace;
- select File/New/Java Project;
- type in a project name (for example "db4o");
- select "Create project from existing source" and browse to the src/db4oj folder in your installation (you may want to copy it to another location first);
- click "Next" and "Finish".

Further reading:

[Db4o Directory Structure](#)

[Building Java Version](#)

[Building .NET Version](#)

This revision (14) was last Modified 2007-11-28T15:32:05 by Tetyana.

# Using The Repository

If you enjoy being on the "cutting edge" and want to follow up with the development process, you can use our SVN repository to get the most up-to-date db4o source code.

If you are using [Eclipse](#) it may be convenient for you to use [Subversive plugin](#) as the Subversion client.

Access to the public projects on our Subversion server is available under the following public URL. No login is required.

<https://source.db4o.com/db4o/trunk/>

The following projects are currently available.

Projects may be under constant development.  
Source code is not guaranteed to be stable.

Most top-level modules in svn directly map to Eclipse projects, i.e. the root folder contains the Eclipse project metadata. If a top-level module acts as a container for related projects rather than as a standalone project, its name should be suffixed with **-projects** by convention, indicating that you'll have to look inside to find the actual project folders to check out.

## bloat

Bytecode instrumentation library

## cruisecontrol

Cruisecontrol project for db4o build

## dashboard

db4o eclipse utility to watch db4o cruisecontrol build status

## db4o-update-site

db4o OSGI plugin installation and update site

## db4o.archives

Collection of db4o versions.

## **db4o.net**

db4o for .NET sources including dRS

## **db4oME**

db4o for J2ME CLDC (spike project, to be integrated with regular production)

## **db4obuild**

Resources and scripts to build db4o

## **db4oclassedit**

Class edit layer on top of bloat

## **db4odebugutils**

Debug utilities for db4o

## **db4oj**

db4o for Java, compilable against JDK 1.1

## **db4oj.tests**

Collection of db4o regression tests

## **db4ojdk1.2**

db4o for Java, sources for JDKs 1.2 to 1.4

## **db4ojdk5**

db4o for Java, sources for JDK 5

## **db4onqopt**

Native query optimizer for Java

## **db4opolepos**

[Poleposition](#) benchmark for db4o

## **db4otaj**

Instrumentation for db4o TA

## **db4otesteclipse**

db4ounit integration in Eclipse

## **db4otools**

Instrumentation tools for db4o

## **db4ounit**

[Unit-test framework for db4o](#) (no db4o dependencies)

## **db4ounit.extensions**

db4o-specific extensions for [db4o testing framework](#)

## **docWiki**

Offline version of [db4o reference documentation](#)

## **doctor**

Internal db4o documentation system

## **eiffel**

Tests with Eiffel for .NET

## **javatocsharp**

**javatocsharp.core** - java to c# converter

**javatocsharp.ui** - Eclipse UI java to c# converter extension

## **objectmanager**

Old Object Manager sources

## **objectmanager-api**

Object Manager API

## **objectmanager-swing**

Object Manager (database browser)

## **osgi-projects**

**osgi\_db4o** - OSGi db4o bundle with factory service

**osgi\_db4o\_test** - bundle to run db4ojdk1.2 test suite in OSGi context

## **reference**

Collection of java examples for the reference documentation

## **reference.net**

Collection of c# examples for the reference documentation

## **reference.vb.net**

Collection of VB.NET examples for the reference documentation

## **sandbox**

Sandbox

## **spikes**

Placeholder for db4o spike projects

## **sync4o**

Client and Server sync sources for the Funambol sync4j system.

## **tutorial**

Tutorial sources

## **version6converter**

Converter for .NET legacy projects to db4o version 6 .NET conventions

The latest development source code is available from TRUNK.

For branches, tags and versions we recommend downloading the distributions from our [download center](#).

Further reading:

[Db4o Directory Structure](#)

[Building Java Version](#)

[Building .NET Version](#)

This revision (13) was last Modified 2008-04-14T16:40:48 by Tetyana.

# Db4o Directory Structure

This topic will explain the directory structure of the db4o project.

Note: Java version of db4o is separated into 3 projects:

db4oj - db4o version for JDK1.1

db4ojdk1.2 - db4o version for JDK1.2-1.4, depends on db4oj

db4ojdk5 - db4o version for JDK5-6, depends on db4oj

db4ojdk1.2 and db4ojdk5 contain only those folders, where functionality is different from db4oj.

The following folders are used within the db4o core project.

**Activation** - Transparent Activation support classes.

**Cluster** - contains cluster source code, which allows queries against several databases. Work in progress.

**Config** - contains configuration interface and other classes and interfaces used for db4o tuning and configuration.

**Constraints** -unique constraints code.

**Defragment** - defragmentation code and related service classes and interfaces.

**Diagnostic** - diagnostics classes and interfaces.

**Events** - external events implementation.

**Ext** - extended db4o functionality .

**Foundation** - db4o base classes and interfaces.

**Io** - file system related code.

**Network** - classes for network communications (buffer, socket).

**Internal** - internal db4o logic.

**Btree** - b-Tree implementation. Used for indexing, freespace, defragment etc.

**Callbacks** - callbacks definitions.

**ClassIndex** - class index implementation.

**Cluster** - internal cluster code.

**Collections** - fast collections development classes.

**Convert** - version converter.

**CS** - client/server code

**Diagnostic** - diagnostic processor.

**Events** - internal events implementation.

**Fieldindex** - field index logic

**Fileheader** - classes for handling db4o file header.

**Freespace** - freespace management code.

**Handlers** - different type handlers (array, byte, char etc).

**Ix** - old indexing logic.

**Mapping** - internal mapping implementation for defragment.

**Marshall** - different type marshallers.

**Query** - query logic implementation.

**Replication** - deprecated replication logic.

**Slots** - classes dealing with slots in db4o file.

**IO** - db4o IoAdapter implementations.

**Marshal** - marshalling interfaces.

**Messaging** - messaging interfaces for client/server communications.

**Query** - classes and interfaces for different query types.

**Reflect** - reflection interfaces, generic reflector implementation.

**Core** - abstract reflect classes.

**Generic** - generic reflector implementation.

**Jdk** - wrapper classes to JDK reflection.

**Self** - reflector implementations for JDK platforms without reflection support.

**Replication** - db4o replication code

**Ta** - transparent activation code

**Types** - db4o specific types

This revision (3) was last Modified 2007-10-28T10:33:47 by Tetyana.

# Configuring Java System Libraries

This topic applies to Java version only.

This topic explains how to configure System Library names used by db4o in Eclipse.

You will need JREs with the following names configured in Eclipse: "1.3", "5" In order to set these up, you do not need to install the corresponding Java JREs on your machine. It is sufficient to have the above named Eclipse JREs point to the JDK5 "JRE home directory". Here is an example how to install a JRE with the name "1.3" in Eclipse, pointing to JDK5:

- open Window + Preferences + Java + Installed JREs;
- click "Add";
- enter the name (e.g. "1.3");
- enter the path to the JRE (e.g. "C:\Program Files\Java\jdk1.5.0\_01").

This revision (2) was last Modified 2007-10-14T16:56:51 by Tetyana.

# Building Java Version

## Contents

- [Projects Required](#)
- [machine.properties](#)
- [Running the Build](#)

This topic applies to Java version only

## Projects Required

---

In order to build db4o you must have access to db4obuild project, which contains build scripts for creating a full db4o java distribution. You will need to check out the following projects.

db4obuild - db4o build tool

bloat - bytecode optimization library, required for db4o NQ optimizer

db4o-osgi - db4o OSGI library

db4o-osgi-tests - tests for db4o OSGI

db4oj - db4o core

db4ojdk1.2 - db4o core for JDK1.2-1.4

db4ojdk5 - db4o core for JDK5

db4otaj - TA instrumentation

db4otools - bytecode instrumentation for db4o

db4onqopt - db4o NQ optimizer

db4ounit - db4o unit-test framework

db4ounit.extensions - db4o unit-test framework extensions

doctor - documentation generation tool, required for building interactive tutorial

tutorial - interactive tutorial

docWiki - db4o reference documentation

## machine.properties

---

You will need to create machine.properties file in db4obuild folder. The contents of the file can be copied from build.xml (see the comments at the beginning of the file). Modify the paths where applicable to set the build variables for your environment.

If you do not have a JDK 1.3 installed and you want to use JDK 5 to build db4o for Java 1.1, you will need this line:

```
file.compiler.jdk1.3.args.optional=-source 1.3
```

## Running the Build

---

You can start the build by right-clicking build.xml file and selecting "Run As/Ant Build". You will run "buildjava" target. (You can't use "buildall" or "buildnet" as they require javatocsharp project, which is not public.)

The ready distribution can be found in /dist folder of db4obuild project

Further reading:

[Testing Db4o](#)

[Patch Submission](#)

This revision (9) was last Modified 2007-12-23T08:39:48 by Tetyana.

# Building .NET Version

This topic applies to .NET version only

Further reading:

[Testing Db4o](#)

[Patch Submission](#)

This revision (13) was last Modified 2007-11-28T15:31:10 by Tetyana.

# Testing Db4o

After you've modified the sources and built the library it is always a good idea to test your new version thoroughly. For this purpose, you can use db4o regression tests.

In order to run regression tests in java you will need the following projects:

bloat

db4oj

db4ojdk1.2

db4ojdk5

db4oj.tests

db4o osgi

db4o osgi test

db4otaj

db4otools

db4ounit

db4ounit.extensions

The test suite is started from com.db4o.db4ounit.jre5.AllTestsDb4oUnitJdk5 class in db4ojdk5 project. You can check the output of the tests in the console ("GREEN" - for successful test completion).

For more information see [Db4o Testing Framework](#).

This revision (4) was last Modified 2007-10-28T11:19:09 by Tetyana.

# Object Manager Sources

[Object Manager](#) is a GUI tool for browsing, querying and managing db4o database. Object Manager sources are available from the following projects in our [SVN](#):

- objectmanager-swing - contains Object Manager GUI application
- objectmanager-api - contains Object Manager API

Object Manager application depends on db4o Java library. In order to build Object Manager you will need to check out and build db4o java distribution. Note, that Object Manager sources and db4o sources should be checked out in the same folder.

You can build Object Manager by running build.xml in objectmanager-swing folder. The ready distribution package will appear in the same folder on completion.

This revision (3) was last Modified 2007-09-09T18:37:12 by Tetyana.

# Patch Submission

If you want to contribute to the core code, you must follow our [contribution policy](#).

This topic explains how to prepare your patch.

Before writing a patch, please, familiarize yourself with our [Coding Style](#) conventions.

You can create a patch using "Create Patch" SVN command.

If you are using [Subversive](#) plugin you can use the following steps:

- select the project in Package Explorer;
- right-click and select Team/Create Patch;
- select "Save In File System" and choose the file name;
- click "Next" and "Finish";
- check unversioned resources that should be included in the patch;
- click "OK"

Once the patch is created you should register the functionality provided with our [Jira tracking system](#) by creating new issue, submitting the description, patch and [test case](#) if applicable.

This revision (4) was last Modified 2007-12-06T04:13:53 by German Viscuso.

# Project Dependencies

Project	Dependencies
db4oj	JDK1.3 (can be run on JDK1.1)
db4ojdk1.2	JDK1.2 (can be built on JDK1.2). db4oj db4oj.tests db4ounit db4ounit.extensions
db4ojdk5	JDK5 db4oj db4oj.tests db4ounit db4ounit.extensions
db4o_osgi	JDK5 bloat-1.0.jar db4o-x.x-java1.2.jar db4o-x.x-nqopt.jar
db4o osgi test	JDK5 db4o-x.x-db4ounit.jar db4o-tests.jar db4o-tests-nq.jar
db4otaj	JDK1.3 bloat db4oj.tests db4ojdk1.3 db4otools db4ounit db4ounit.extensions
db4otools	JDK1.3 bloat db4ojdk1.2 db4ounit db4ounit.extensions

## Project Dependencies

<b>db4o.net</b>	.NET1.1 .NET2.0 .NET1.1.CF .NET2.0.CF
<b>db4oj.tests</b>	JDK1.3 db4oj db4ounit db4ounit.extensions
<b>db4oME</b>	cldcapi1.1.jar jsr75.jar midpapi20.jar
<b>db4onqopt</b>	JRE5 bloat db4oj db4oj.tests db4ounit db4ounit.extensions
<b>db4opolepos</b>	db4oj db4ojdk1.2 db4ojdk5 db4onqopt polepos
<b>db4ounit</b>	JRE1.1
<b>db4ounit.extensions</b>	JRE1.3 db4oj db4ounit
<b>doctor</b>	JRE1.4 ant.jar itext.jar junit.jar
<b>docWiki</b>	None
<b>eiffel</b>	N/a
<b>objectmanager-swing</b>	db4ojdk5

<b>sync4o</b>	JRE5 commons-lang-2.1.jar funambol-admin-dev.jar funambol-clientframework.jar funambol-framework.jar funambol-server.jar com-funambol-admin.jar commons-codec-1.3.jar sc-api-j2se.jar sync4j-clientframework.jar db4o-5.5-java1.2.jar
<b>drs</b>	db4oj db4ojdk5 db4ounit ojdbc14.jar antlr-2.7.6.jar asm.jar asm-attrs.jar cglib-2.1.3.jar commons-collections-2.1.1.jar commons-logging-1.0.4.jar dom4j-1.6.1.jar ehcache-1.2.jar hibernate3.jar jta.jar log4j-1.2.13.jar xerces-2.6.2.jar ant.jar sqljdbc.jar doctor.jar iText.jar junit.jar hsqldb-1.8.0.7.jar postgresql-8.1-407.jdbc3.jar mysql-connector-java-5.0.4-bin.jar db2jcc.jar db2jcc_license_cu.jar derby.jar

## Project Dependencies

<b>drspolepos</b>	JDK1.5 db4oj db4ojdk1.2 db4ojdk5 db4oj.tests db4ounit db4ounit.extensions drs polepos
<b>drsquickstarts</b>	.NET2.0
<b>reference</b>	JDK5 db4ojdk5
<b>reference.net</b>	.NET2.0
<b>reference.vb.net</b>	.NET2.0
<b>spikes</b>	Depends on project
<b>Tutorial</b>	JRE1.4 db4ojdk1.2

This revision (5) was last Modified 2007-10-28T11:25:27 by Tetyana.

# Coding Style

## Contents

- File Header
- Naming Conventions
  - General Naming
  - Package naming
  - Class Naming
  - Methods Naming
  - Abbreviations and Acronyms
  - Type Naming
  - Variable Naming
  - Constants
  - Getters/Setters
  - Boolean Methods And Variables
  - Initialize
  - Complementary Names
  - Abbreviations
  - Named Constants
- Code Organization
  - Package Structure
  - Classes and Interfaces
  - Methods
  - Wait & Notify
  - Blank Lines
  - Import Declarations
  - Initialization
- Exception Handling
- Comments

Coding conventions proved to be very important for producing maintainable and reliable code. In db4o production cycle, coding conventions have a special value, as the code ownership is spread over all the members of the team and any developer is able to work with any piece of code.

In general, we follow [Code Conventions for the Java Programming Language](#), however there are some specifics, which can be useful to know for db4o users and core contributors.

This document is supposed to emphasize some of the java coding style recommendations used by db4o and explain db4o specific coding style requirements.

## File Header

---

All code files must have copyright. The normal db4o copyright notice should be created as the standard template in your Eclipse workspace for db4o development.

Window + Preferences + Java + Code Style + Code Templates + Code + New Java Files

/\* Copyright (C) 2004 - 2007 db4objects Inc. http://www.db4o.com \*/

\${package\_declaration}

\${typecomment}

\${type\_declaration}

## Naming Conventions

---

### General Naming

---

All names should be written in English.

English is the preferred language for international development.

### Package naming

---

Package names should be in all lower case.

com.db4o.reflection

### Class Naming

---

Class names should be nouns and written in mixed case starting with upper case.

ObjectContainer, Configuration

### Methods Naming

---

Method names must be verbs and written in mixed case starting with lower case.

isReadOnly(), rename()

### Abbreviations and Acronyms

---

Abbreviations and acronyms should not be uppercase when used as name.

IoAdapter(); // NOT: IOAdapter

Using all uppercase for the base name will give conflicts with the naming conventions given above and will reduce the readability.

## Type Naming

---

Type names must be nouns and written in mixed case starting with upper case.

Db4oList, TransientClass

## Variable Naming

---

Variable names must be in mixed case starting with lower case. Variables should have full sensible name, reflecting their purpose.

listener, objectContainer

- Private Variables

Private class variables should have underscore prefix.

```
public abstract class IoAdapter {  
  
    private int _blockSize;  
  
    ....  
}
```

Underscore prefix will help a programmer to distinguish private class variables from local scratch variables.

- Scratch Variables

Scratch variables used for iterations or indices should be kept short. Common practice is to use i, j, k, m, n for numbers and c, d for characters.

## Constants

---

Constants names (final variables) must be all uppercase using underscore to separate words.

ACTIVATION\_DEPTH, READ\_ONLY

It is a good practice to add methods to retrieve constant values for user interface:

```
boolean isReadOnly() {
    return _config.getAsBoolean(READ_ONLY);
}
```

## Getters/Setters

---

Normally we do not use *get/set* prefix for methods accessing attributes directly, unless such usage adds valuable information:

```
public void setStateDirty() {}
```

In other cases feel free to access attributes by names:

```
clientServer(), configuration()
```

## Boolean Methods And Variables

---

*is(can, has, should)* prefix should be used for boolean variables and methods.

```
isDirty(), canHold(reflectClass)
```

Using the *is(can, has, should)* prevents choosing bad names like *status* or *flag*. *isStatus* or *isFlag* simply doesn't fit, and the programmer is forced to choose more meaningful names.

Setter methods for boolean variables must have *set* prefix as in:

```
public void setStateDirty() {}
```

## Initialize

---

The term *initialize* can be used where an object or a concept is established. `classIndex.initialize(_targetDb);`

Abbreviations like *init* must be avoided.

## Complementary Names

---

Complementary names must be used for complementary entities:

get/set, add/remove, create/destroy, start/stop, insert/delete, increment/decrement, old/new, begin/end, first/last, up/down, min/max, next/previous, old/new, open/close, show/hide, suspend/resume, etc. For example:

```
startServer();
```

```
stopServer();
```

This convention helps to distinguish the borders of a logical operation and to recognize opposite action methods.

## Abbreviations

---

Abbreviations in names should be avoided.

```
copyIdentity(); // NOT: cpIdentity, NOT:copyId
```

However some well established and commonly used acronyms or abbreviations must be preferred to full names:

```
html // NOT: HypertextMarkupLanguage  
cpu // NOT: CentralProcessingUnit
```

## Named Constants

---

Named constants should be used instead of:

- magic numbers:

```
if (blockSize > MAX_BLOCK_SIZE) // NOT: blockSize > 256
```

- fixed phrases:

```
if (fieldname == CREATIONTIME_FIELD) // NOT if (fieldname ==  
"i_uuid")
```

This convention gives a programmer an idea about the meaning of the constant value. At the same time, it makes it easier to change the constant value: the change must be made only in one place.

## Code Organization

---

## Package Structure

---

Internal class implementations should be placed in com.db4o.internal package. This helps to keep the top-level API smaller and more understandable.

com.db4o.query.Evaluation

has implementation in

com.db4o.internal.query.PredicateEvaluation

## Classes and Interfaces

---

Class and Interface declarations should be organized in the following manner:

1. Class/Interface documentation.
2. class or interface statement.
3. Class (static) variables in the order public, protected, package(no access modifier), private.
4. Instance variables in the order public, protected, package(no access modifier), private.
5. Constructors.
6. Methods.

## Methods

---

Group class methods by functionality rather than by scope or accessibility. For example, a private class method can be in between two public instance methods. The goal is to make reading and understanding of the code easier.

## Wait & Notify

---

Use Lock4#snooze(), #awake() instead of Object#wait directly for CF reason.

## Blank Lines

---

Use blank lines to separate methods, class variable declarations of different scope, and logical units within a block of code. Consider extracting logical blocks of code into separate methods

## Import Declarations

---

Import declarations should only include package name::

```
import java.util.*; // NOT: import java.util.List;
```

Modern IDEs, such as [Eclipse](#), provide an automated way to create correct import statements (see "Source/Organize Imports" command in Eclipse).

## Initialization

---

Local variables should appear at the beginning of a code block. (A block is any code surrounded by curly braces "{" and "}".) Try to initialize the variable immediately to prevent using uninitialized values.

The exception to the rule is indexes of `for` loops, which in Java can be declared in the `for` statement:

```
for (int i = 0; i < maxLoops; i++) { ... }
```

Avoid local declarations that hide declarations at higher levels. For example, do not declare the same variable name in an inner block:

```
int count;  
  
...  
  
myMethod() {  
  
    if (condition) {  
  
        int count = 0;      // AVOID!  
  
        ...  
  
    }  
  
    ...  
  
}
```

## Exception Handling

---

Never use `exception#printStackTrace()` in db4o productive code.

There are three possible choices that can be used:

- throw (the base type is Db4oException)
- swallow silently
- swallow with a com.db4o.diagnostic message to the user

## Comments

---

Supply your code with javadoc comments. All public classes and public and protected functions within public classes should be documented. This makes it easy to keep up-to-date online code documentation. If a class or a method is not part of public API use @exclude tag. For further details, see "[How to Write Doc Comments for Javadoc](#)"

All comments should be written in English. In an international environment, English is the preferred language.

Avoid using comments to explain tricky code, rather rewrite it to make it self-explanatory.

For more information see: [Code Conventions for the Java Programming Language](#).

This document was compiled based on db4o team coding practices and the following documents:

<http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html>

<http://geosoft.no/development/javastyle.html>

<http://java.sun.com/j2se/javadoc/writingdoccomments/index.html>

This revision (12) was last Modified 2007-12-20T05:20:22 by German Viscuso.

# License

## Contents

- Licensing the db4o Engine
  - General Public License (GPL) Version 2
  - Commercial License
  - db4o Opensource Compatibility License (dOCL)
- 3rd Party Licenses
  - In java versions

## Licensing the db4o Engine

---

[db4objects Inc.](#) offers three difference license options for the db4o object database engine db4o:

### General Public License (GPL) Version 2

---

db4o is free under the [GPL](#), where it can be used:

- for development
- in-house as long as no deployment to third parties takes place
- together with works that are placed under the GPL themselves

You receive a copy of the GPL in the file db4o.license.txt together with the db4o distribution.

If you have questions whether the GPL is the right license for you, please read:

- db4objects and the GPL - [frequently asked questions FAQ](#)
- the free whitepaper [db4objects and the Dual Licensing Model](#)
- [db4objects' GPL interpretation policy](#) for further clarification

### Commercial License

---

For incorporation into own commercial products and for use together with redistributed software that is not placed under the GPL, db4o is also available under a commercial license.

Visit the [commercial information on db4o website](#) for licensing terms and pricing.

### db4o Opensource Compatibility License (dOCL)

---

The db4o Opensource Compatibility License (dOCL) is designed for free/open source projects that want to embed db4o but do not want to (or are not able to) license their derivative work under the GPL in its entirety. This initiative aims to proliferate db4o into many more open source projects by providing compatibility for projects licensed under Apache, LGPL, BSD, EPL, and others, as required by our users.

The terms of this license are available here: ["dOCL" agreement](#).

## 3rd Party Licenses

---

When you download the db4o distribution, you receive the following 3rd party libraries:

### In java versions

---

- [Apache Ant](#)(Apache Software License)

Files: lib/ant.jar, lib/ant.license.txt

Ant can be used as a make tool for class file based optimization of native queries at compile time.

This product includes software developed by the Apache Software Foundation(<http://www.apache.org/>).

- [BLOAT](#)(GNU LGPL)

Files: lib/bloat-1.0.jar, lib/bloat.license.txt

Bloat is used for bytecode analysis during native queries optimization. It needs to be on the classpath during runtime at load time or query execution time for just-in-time optimization. Preoptimized class files are not dependent on BLOAT at runtime.

These products are not part of db4o's licensed core offer and for development purposes. You receive and license those products directly from their respective owners.

This revision (14) was last Modified 2007-11-27T19:18:32 by Tetyana.

# Contacts

## db4objects Inc.

1900 South Norfolk Street

San Mateo, CA, 94403

USA

## Phone

+1 (650) 577-2340

## Fax

+1 (650) 240-0431

## Sales

Fill out our [sales contact form](#) on the db4o website

or mail to [sales@db4o.com](mailto:sales@db4o.com)

## Support

Visit our [free Community Forums](#)

or log into your [Customer Portal](#) (dDN Members Only).

## Careers

[career@db4o.com](mailto:career@db4o.com)

## Partnering

[partner@db4o.com](mailto:partner@db4o.com)

This revision (5) was last Modified 2008-04-14T16:50:20 by Tetyana.