Introduction

db4o is an object database management system developed and distributed by Versant Corporation. db4o is an open-source product and is available under dual license.

Overview

- To install db4o read the getting started manual.
- For information about the database operations with db4o read about the basic operations. Use LINQ to query for objects in your database. There are also other alternative query methods: query by example and SODA.
- Take a look at basic db4o concepts like transaction handling and ACID properties and object identity .
- To get an idea when db4o loads object into memory read about the activation concept. And learn you update and delete objects with db4o.
- For good query performance you should create indexes.
- Read about all the db4o configuration options available to you. How you can change the storage system, the activation and update depth and how you can configure an individual class.
- db4o also support a client server mode.
- Take a look at advanced features like session containers, unique constrains, backups, db4o ids and UUIDs and meta-information.
- Callbacks allow you to take action on certain database events.
- You should defragment the database from time to time for the best performance.
- Read about the refactoring support in db4o.
- Take a look at the best practices and pitfalls for db4o.
- db4o supports on many platforms and those have some platform specific limitations. Like the pitfalls in a web environment, and Silverlight .
- For replication between db4o, VOD and Hibernate, read the db4o Replication System documentation.

Join The db4o Community

Join the db4o community for additional help, tips and tricks. Ask for help in the db4o forums at any time. Take a look at additional resources on the community website. If you want to stay informed, subscribe to our blogs.

Product Philosophy

The db4o database is sponsored and supported by Versant Corporation, a publicly-held company (NASD:VSNT) based in Redwood City, California. Versant is a leading developer of object database technology supporting both open source and commercial database initiatives.

Versant's commercial object database technology, targeting extreme scale systems, is powering some of the world's most demanding applications for fortune 1000 companies in industries including:

- Telecommunications: Alcatel-Lucent, Deutsche Telecom, France Telecom, Ericsson, NEC, Nortel, Orange, Samsung, and more.
- Finance: Financial Times, New York Stock Exchange, Dow Jones, Reuters, London Clearing House, Bank of America, and more,
- Transportation: Sabre, GE Railways, BNS Railways, Galileo, and more.
- Defense: Raytheon, Northrop Grumman, Lockheed, and more.
- BioInformatics: Mayo Clinic, St. Jude medical, Eidogen, Science Factory, and more.

db4o has users and customers coming from 170 different countries, from Albania to Zimbabwe, and ranging from world class leaders like Boeing, Bosch, Intel, Ricohand Seagate to a wide range of highly innovative start-up companies.

It is Versant's and the db4o team's mission to give developers a choice differentiated from relational approaches when it comes to object persistence and thus make their life a lot easier. There is no mapping! No mapping annotations or XML mapping meta data. The db4o database is designed to be a universal, affordable product platform that is easy to learn and use. Versant's open source dual-license business model combines the power of an open source development community with servicing commercial customers' needs for a predictable product roadmap, indemnification, single point of contact and full tech support with fast response times. For those requiring the super scale database capabilities, you can find the same easy to learn and implement solution in Versant's commercial products which have been in development for over a decade. This technology is also far more affordable that traditional relational database systems such as Oracle, Sybase, SQL Server, etc and to boot users also enjoy overall reductions in system footprints by as much as 50% due to less indexing data, simpler design, zero mapping.

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. 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 relations, attributes and tuples.

The object-relational mismatch has become enormously significant with the total adoption of OO technology. This resulted in the rapid development of object-relational mappers (ORM), such as Hibernate, EclipseLink or Entity Framework. This solution "cures" the symptoms of the object relational 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.

Significant side effects of the object relational mismatch manifest themselves in unnecessary system overhead with bloated footprints and runtime performance issues. Of course, there is also overall time to market delays due to poor developer productivity. The overhead still exists in ORM because under the covers, the runtime is still query driven. And, despite improvements in productivity for developers, incremental changes to your object models reek havoc during ORM schema evolution pitfalls. The more complicated your models are, the more problematic keeping changes in sync with the internal mapping.

Primary performance issues come from the fact that despite being called a "relational database", an **RDBMS**[1] does not store direct relations. Relations are resolved at runtime by performing set based operations on primary-foreign key pairs. This means the application has to constantly re-discover data relationships at runtime resulting in immense CPU consumption for something that should be an inherent part of your application model. Further, because discovering these relations over and over again requires continual access to index structures and data to perform the set operations, contention is much higher within database internals leading to poor scalability of individual database processes.

Further, lack of direct storage of relations cause the application design to become query driven instead of object modeling driven. Using an object database, the relations are a fundamental part of the storage architecture. So, application design is model driven. You do not have to suffer any performance overhead for discovering an M-M relationship. The relationships are just there and immediately accessible to the requesting thread. This makes the internal structures much simpler and therefore less contention exists with data requests being isolated to data of interest instead of leveraging indexes or sequential scans. The result, individual processes become more scalable under concurrency.

Technology is ever changing and today there is a whole world of object database experts in the software community. Anyone who is an expert in ORM technology is an expert in object database technology. All of the concepts found in object life cycle management within ORM technologies were invented by the object database community in the early 90's. All of the tuning concepts of closure, fetch configurations, value vs reference types, light weight transactions - are concepts created by and applicable to object database technologies. Now with the growing popularity of object based design and the proliferation of ORM tools, thousands of developers are becoming experts in the object database API.

OODBMS

The emergence of distributed data architectures - in networks, on clients and embedded in "smart" products such as cars or medical devices - is causing companies in an array of industries to look beyond traditional **RDBMS**[2] 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, offer powerful replication and query capabilities, reduce development and maintenance costs and require minimum to zero administration overhead.

These requirements can be fulfilled by using an Object Oriented Database Management System (**OODBMS**[3]). OODBMS provides an ideal match with object oriented environments like Java and .NET reducing the cost of development, support and versioning and hence overall system costs.

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

[1]Relational Database Management System
[2]Relational Database Management System
[3]Object Oriented Database Management System

- Continuous refactoring.
- Agile modeling.
- Continuous regression testing.
- Configuration management.
- Developer "sandboxes".

(For more information about OODBMS technology refer to the ODBMS.ORG website.)

db4o Position

The db4o database 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. Versant's vision for db4o is to become the affordable, dominant, open source persistence solution of object oriented developers of Java and .NET. In a very short time, the db4o team has achieved mainstream adoption with a fast growing user community currently boasting over 60,000 members. Community adoption is continually driven by db4o's efficient innovative technology, native queries, deployment in Java and .NET 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**[1] 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 Seagateto 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.

For deployments requiring a highly scalable client/server database solution, Versant's commercial product line can deliver a solution with equal ease of use at a surprisingly low cost compared to relational database solutions.

Open Source

db4o database technology 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 (60,000 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.

[1] Relational Database Management System

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, 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.

db4o Applications

db4o can be used in a wide range of production and educational software. The primary focus is on embedded usage, like mobile systems (phones and handhelds), device electronics (printers, cars, robots), SCADA systems etc. The following table provides many (but not all) possible implementations with an explanations of the benefits of db4o in the selected environment:

Environment	Benefits
Educational systems	One-line persistence, Object-oriented model, intuitive programming interface make db4o an ideal educational tool. It is easy to use and it provides a meaningful example of object-oriented world. It is also native to most widely used OO languages: Java and .NET
Prototypes	Using db4o to build a prototype system is much quicker than using an **RDBMS**[1]. In case of db4o you do not need to create a data model. Further there is no need to map your object model to the database. The general persistence mechanism is almost transparent and requires minimum effort to adapt to. Automatic refactoring allows rapid change of classes without the necessity to update the database.
SCADA	Using db4o in SCADA systems allows to achieve high performance in caching and replay of the events. Another benefit is a small footprint and easy integration with Java and .NET programming languages. db4o can also be run as a memory database, providing better performance though minimizing disk access.
Mobile applications	Mobile applications can benefit from in-process database, which requires zero-administration. Synchronization with the main server can be done with the help of **dRS**[2]. Automatic refactoring can be another valuable factor, which allows to skip the job of updating the databases when a new version of object model is implemented.
Device applications	Device applications enjoy the same benefits as Mobile applications. In addition, smaller footprints can be achieved by using the minimal Micro edition.
Open-source software	GPL, open-source compatibility licence. Native to Java and .NET. Easily integrates with any Java and .NET open-source products.
Web-applications	open-source, reporting support from several Java open-source reporting frameworks and .NET reporting API

[1]Relational Database Management System
[2]db4o Replication System

However, other applications might not be well suited for db4o.

For example, in situations where you have increasing amounts of data (over 10 Gigabytes) and high concurrency (over 20 concurrent users/processes) along with your complex models. In these cases, the Versant database is likely a more appropriate choice. Versant's customer applications span a wide range of use including those exhibiting 1000's of current transactions (100's of thousands of concurrent tx per second) to 100's of gigabytes with some Versant customers in the 25T+ sized database. For more information visit http://www.versant.com/

Another case is when you have simple and flat data model, primarily used for reporting. Simple table-like models of tuple records may be better supported by an RDBMS. In this case, adhoc data access would be more important to your application than well defined use cases using an object model. Typically this is complimented with the need let your users to be able to grab one of the plethora of commercial tools to poke at the database in an adhoc fashion.

Scalability

db4o can potentially be used for rather large databases - up to 254 GB per database file. However, this would be highly unusual and likely due to having the kind of application that was storing relatively few, but very large objects in the database. As a general rule, if you expect your database to grow beyond 10 gigabytes, you should probably be looking at the Versant object database.

Also db4o is currently not designed to scale for highly concurrent access. If you expect a high load of concurrent access to the database, you should also consider a larger database like the Versant object database.

If you want to make sure that your application can grow and scale with db4o database you may take the following steps:

- Write performance tests to accommodate for performance requirements of the growing application. Especially look that you also capture to complexity of the model and don't use a simplified flat data model.
- Use client-server version to enable many clients using data at the same time.

Why Choose db4o

There are many advantages of using "native" object technology over **RDBMS**[1] 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.

Second, with an object database, the object schema 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**[2]) with its Version 5.0 launched in November 2005 and since introduction of LINQ in .NET version 3.5. Db4o Native Queries provide an API

[1]Relational Database Management System
[2]Native Query

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 and LINQ 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'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 back-end 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.

Fifth, 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, object oriented 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**[1] or RDBMS technologies available.

[1]Object Oriented Database Management System

Basics Operations & Concepts

This topic gives you an overview of the most important and basic operations of db4o. First add db4o to your project. db4o doesn't need a complex setup. It's just a library which you can add to your project. See "Getting Started" on page 10

Basic Operations

The basic operations are, not surprising, storing, updating, querying and deleting objects. See "The Basic Operations" on page 11

```
using(IObjectContainer container = Db4oEmbedded.OpenFile("databaseFile.db4o"))
{
    // store a new pilot
    Pilot pilot = new Pilot("Joe");
    container.Store(pilot);

    // query for pilots
    var pilots = from Pilot p in container
                 where p.Name.StartsWith("Jo")
                 select p;

    // update pilot
    Pilot toUpdate = pilots.First();
    toUpdate.Name = "New Name";
    container.Store(toUpdate);

    // delete pilot
    container.Delete(toUpdate);
}
```

Db4oBasics.cs: The basic operations

```
Using container As IObjectContainer = Db4oEmbedded.OpenFile("databaseFile.db4o")
    ' store a new pilot
    Dim pilot As New Pilot("Joe")
    container.Store(pilot)

    ' query for pilots
    Dim pilots = From p As Pilot In container Where p.Name.StartsWith("Jo")

    ' update pilot
    Dim toUpdate As Pilot = pilots.First()
    toUpdate.Name = "New Name"
    container.Store(toUpdate)

    ' delete pilot
    container.Delete(toUpdate)
End Using
```

Db4oBasics.vb: The basic operations

For more information about queries: See "Querying" on page 13

Basic Concepts

There are some basic concepts which are used in db4o. Understanding them helps to understand how db4o behaves. Some behaviors are common behaviors which you expect from a database, like transactions and ACID properties. See "ACID Properties And Transactions" on page 40 db4o manages objects by identity. This means db4o doesn't need additional id-field on your object. See "Identity Concept" on page 44

Other concepts are more unique to db4o. For example the activation concept, which controls which objects are loaded from the storage to memory. See "Activation Concept" on page 47. The same principals are also applied when updating (See "Update Concept" on page 71) or deletion objects (See "Delete Behavior" on page 99

Getting Started

You can start using db4o within a few minutes following these few steps.

1. Download db4o

First download db4o on the official download site. There are different releases available on the web site. The latest production version, beta versions, continues build versions and older stable releases. Use the production version at the top to get started.

For each release there's a download for a particular .NET version. Choose the .NET version which you're using. For example .NET 4.0. You also can choose between a ZIP-file or a MSI-installer.

Download the file to your computer. If you've chosen the ZIP-file distribution unpack it. In case you've chosen the MSI-installer run it.

2. Content Of The db4o Distribution

The db4o distribution has following content.

db4o-folder/bin: Contains the db4o database engine and the supporting libraries. This folder contains sub-folders for different .NET platforms like the regular .NET framework, the compact framework etc.

db4o-folder/doc: Contains all the db4o documentation. There's the complete API-documentation (db4o-folder/doc/api/db4o.chm), the tutorial (db4o-folder/doc/tutorial/Db4oTutorial.exe) and the reference documentation (db4o-folder/doc/api/index.html)

db4o-folder/omn: Contains the installer for the Object Manager. See "Object Manager Enterprise" on page 343

db4o-folder/src: Contains the full source code of db4o.

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

3. Adding db4o To Your Project

After you've downloaded and unpacked the db4o distribution, you can start using it. The core of db4o is the single Db4objects.Db4o.dll-assembly and has no additional dependencies. The other assemblies provide additional functionality See "Dependency Overview" on page 365.

To use db4o in your project you only need to add the required assemblies to your project and then you're ready to go.

Here is how to do this with Visual Studio:

- Open your project or create a new one.
- Right-click on "References" in the Solution Explorer.
- Choose "Add Reference".
- Select "Browse".
- Navigate to db4o distribution folder, there to the binaries folder and choose the assembly-files to add.
- Confirm your choices.

4. Ready To Go

That's it, now you're ready to go and can use db4o in your project.

The Basic Operations

The object container is the door to the database access. It's the starting point for all database operations.

Accessing a Database

The object container is the interface for accessing the database. To open the database you pass the file-name to the object container factory. Normally you should open an object container when the application starts and close it when it is shuts down.

```
using (IObjectContainer container = Db4oEmbedded.OpenFile("databaseFile.db4o"))
{
    // use the object container
}
```

Db4oBasics.cs: Open the object container to use the database

```
' use the object container
Using container As IObjectContainer = Db4oEmbedded.OpenFile("databaseFile.db4o")
End Using
```

Db4oBasics.vb: Open the object container to use the database

Storing Objects

Storing a object with db4o is extremely easy. Open the object container and pass your object to the store method and db4o will do the rest. There's no mapping required. db4o will read the class meta data, the read the object values with reflection and store the data.

```
using (IObjectContainer container = Db4oEmbedded.OpenFile("databaseFile.db4o"))
{
    Pilot pilot = new Pilot("Joe");
    container.Store(pilot);
}
```

Db4oBasics.cs: Store a object

```
Using container As IObjectContainer = Db4oEmbedded.OpenFile("databaseFile.db4o")
    Dim pilot As New Pilot("Joe")
    container.Store(pilot)
End Using
```

Db4oBasics.vb: Store a object

Queries

Querying for objects is also easy. There are different query interfaces available with different benefits.
See "Querying" on page 13

The most natural query method is using LINQ.

```
using (IObjectContainer container = Db4oEmbedded.OpenFile("databaseFile.db4o"))
{
    var pilots = from Pilot p in container
                 where p.Name == "Joe"
                 select p;
    foreach (var pilot in pilots)
    {
        Console.Out.WriteLine(pilot.Name);
    }
}
```

Db4oBasics.cs: Query for objects

```
Using container As IObjectContainer = Db4oEmbedded.OpenFile("databaseFile.db4o")
    Dim pilots = From p As Pilot In container Where p.Name = "Joe"
    For Each pilot As Pilot In pilots
        Console.Out.WriteLine(pilot.Name)
    Next
End Using
```

Db4oBasics.vb: Query for objects

Update Objects

Updating objects is also easy. First you query for the object which you want to update. Then you change the object and store it again in the database.

```
using (IObjectContainer container = Db4oEmbedded.OpenFile("databaseFile.db4o"))
{
    var pilot = (from Pilot p in container
                 where p.Name == "Joe"
                 select p).First();
    pilot.Name = "New Name";
    // update the pilot
    container.Store(pilot);
}
```

Db4oBasics.cs: Update a pilot

```
Using container As IObjectContainer = Db4oEmbedded.OpenFile("databaseFile.db4o")
    Dim pilot = (From p As Pilot In container Where p.Name = "Joe").First()
    pilot.Name = "New Name"
    ' update the pilot
    container.Store(pilot)
End Using
```

Db4oBasics.vb: Update a pilot

Delete Objects

Use the delete-operation to delete objects.

```
using (IObjectContainer container = Db4oEmbedded.OpenFile("databaseFile.db4o"))
{
    var pilot = (from Pilot p in container
                 where p.Name == "Joe"
                 select p).First();
    container.Delete(pilot);
}
```

Db4oBasics.cs: Delete a object

```
Using container As IObjectContainer = Db4oEmbedded.OpenFile("databaseFile.db4o")
    Dim pilot = (From p As Pilot In container Where p.Name = "Joe").First()
    container.Delete(pilot)
End Using
```

Db4oBasics.vb: Delete a object

Querying

db4o supports different query mechanisms.

The most convenient query mechanism is LINQ. You can use .NET's query API to query the objects in your database.

On .NET 2.0, Native Queries provide a sensible alternative for writing queries.

Queries-By-Example 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.

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 Native Queries are too strongly typed. There may be queries that will execute faster in SODA style, so it can be used to tune applications. You can also run snippets of custom query code as part of the SODA query,

Of course, you can mix these strategies as needed.

LINQ

Of course, db4o supports .NET LINQ to query the database.

First, you need to add the *Db4objects.Db4o.Linq.dll*-assembly to your project. This assembly contains db4o LINQ-Implementation. Make also sure that Mono.Reflection.dll is aside it. See "Dependency Overview" on page 365

Note that LINQ requires .NET 3.5 or newer!

LINQ Queries

Writing a query is simple. You write the LINQ query against the object-container. First you need to import the LINQ-namespaces.

```
using System.Linq;
using Db4objects.Db4o.Linq;
```

LinqExamples.cs: Use the LINQ namespace

```
Imports System.Linq
Imports Db4objects.Db4o.Linq
```

LinqExamples.vb: Use the LINQ namespace

After that, you can start to write a LINQ-Query against the object container.

```
var allPersons = from Person p in container
                 select p;
```

LinqExamples.cs: Simple query

```
Dim allPersons = From p In container _
 Select p
```

LinqExamples.vb: Simple query

In order to learn LINQ, take a look a the LINQ-resources and tutorial out on the Internet. For example the LINQ-Overview at MSDNor at this example-collection.

There are also a few simple LINQ query examples for db4o here.

IQueryable

db4o also support the .NET IQueryable interface. This is useful when you want to build an abstraction layer for db4o or integrate db4o with other frameworks.

```
IQueryable<Person> personQuerable = container.AsQueryable<Person>();
var adults = from p in personQuerable
             where p.Age > 18
             orderby p.Name
             select p;
```

LinqExamples.cs: Get a IQueryable-instance

```
Dim personQuerable As IQueryable(Of Person) = container.AsQueryable(Of Person)()
Dim adults = From p In personQuerable _
 Where p.Age > 18 _
 Order By p.Name _
 Select p
```

LinqExamples.vb: Get a IQueryable-instance

More on db4o's LINQ Provider

db4o LINQ provider translates the LINQ queries into db4o's low level query API. Read what this query optimization need and what its limits are. See "LINQ Optimization" on page 16

Also read how LINQ is supported on the .NET Compact Framework. See "LINQ For Compact Framework" on page 17

LINQ Examples

In order to learn like, take a look a the LINQ-resources and tutorials out on the Internet. For example the LINQ-Overview at MSDN or at this example-collection. Here are a few db4o LINQ examples.

Simplest Possible Query

This query just returns all persons.

```
var allPersons = from Person p in container
                 select p;
```

LinqExamples.cs: Simple query

```
Dim allPersons = From p In container _
 Select p
```

LinqExamples.vb: Simple query

Query For a Name

This query searches for a person by name.

```
var allPersons = from Person p in container
                 where p.Name.Equals("Joe")
                 select p;
```

LinqExamples.cs: Query for name

```
Dim allPersons = From p As Person In container _
 Where p.Name.Equals("Joe") _
 Select p
```

LinqExamples.vb: Query for name

Query For a Certain Age

This queries for a certain age.

```
var allPersons = from Person p in container
                 where p.Age > 21
                 select p;
```

LinqExamples.cs: Query with a constraint

```
Dim allPersons = From p As Person In container _
 Where p.Age > 21 _
 Select p
```

LinqExamples.vb: Query with a constraint

Sorting The Result Of A Query

Sort the results of a query.

```
var allPersons = from Person p in container
                 where p.Age > 21
                 orderby p.Name
                 select p;
```

LinqExamples.cs: Use sorting on the query

```
Dim allPersons = From p As Person In container _
 Where p.Age > 21 _
 Order By p.Name _
 Select p
```

LinqExamples.vb: Use sorting on the query

Get The IQueryable Interface

You can get the .NET IQueryable interface for db4o

```
IQueryable<Person> personQuerable = container.AsQueryable<Person>();
var adults = from p in personQuerable
             where p.Age > 18
             orderby p.Name
             select p;
```

LinqExamples.cs: Get a IQueryable-instance

```
Dim personQuerable As IQueryable(Of Person) = container.AsQueryable(Of Person)()
Dim adults = From p In personQuerable _
 Where p.Age > 18 _
 Order By p.Name _
 Select p
```

LinqExamples.vb: Get a IQueryable-instance

LINQ Optimization

LINQ queries are converted to SODA Query under the hood. The downside of this is that some of the queries cannot be converted to SODA. In this case db4o falls back to the plain LINQ to objects implementation. This means that all objects are instantiated and the query is ran against the objects. This of course is slower by an order of magnitude.

Note that db4o needs the Mono.Reflection.dll-assembly for this optimization. On the .NET compact framework you Mono.Cecil.dll- and Cecil.FlowAnalysis.dll-assembly. See "LINQ For Compact Framework" on page 17

For example a simple query like this can be optimized:

```
var adults = from Person p in container
             where p.Age > 18 && p.Age < 70
             orderby p.Name
             select p;
```

LinqExamples.cs: A query which is optimizable

```
Dim adults = From p As Person In container _
 Where p.Age > 18 AndAlso p.Age < 70 _
 Order By p.Name _
 Select p
```

LinqExamples.vb: A query which is optimizable

However, queries which invoke operations on the objects cannot be optimized:

```
var adults = from Person p in container
             where p.Name.ToLowerInvariant().Equals("joe")
             select p;
```

LinqExamples.cs: Unoptimizable query, because of the 'operations' withing the query

```
Dim adults = From p As Person In container _
 Where p.Name.ToLowerInvariant().Equals("joe") _
 Select p
```

LinqExamples.vb: Unoptimizable query, because of the 'operations' withing the query

Detect Unoptimized Queries

In Visual Studio, you can see a message in the debugger-output for each unoptimized query. First open the debug output window (Debug->Windows->Output). Then run your application. A query which cannot be optimized will produce this message:

'A first chance exception of type 'Db4objects.Db4o.Linq.QueryOptimizationException' occurred in Db4objects.Db4o.Linq.dll'

To find it out for a particular query, break before the query. Then step over the query and see if the message has been printed out.

For a broader picture, you can also use db4o's monitoring support. This will report the amount of unoptimized queries per second.

Improve Unoptimizable Queries

In cases where you have queries which cannot optimized it's often possible to split the query in two parts. The first part runs optimized. After that you run a regular LINQ to Object query to do the rest of the job. Let's look a this example:

```
var adults = from Person p in container
             where p.Age > 18 && p.Age < 70
                   && p.Name.Substring(2).Contains("n")
             select p;
```

LinqExamples.cs: Unoptimizable query

```
Dim adults = From p As Person In container _
 Where p.Age > 18 AndAlso p.Age < 70 AndAlso p.Name.Substring(2).Contains("n") _
 Select p
```

LinqExamples.vb: Unoptimizable query

In this example the part which calls the substring-operation cannot be optimized. Therefore the query runs very slow on large data sets. But the rest of the query could be optimized. So lets split this query into two parts:

```
var optimizedPart = from Person p in container
                    where p.Age > 18 && p.Age < 70
                    select p;
var endResult = from p in optimizedPart.AsEnumerable()
              where p.Name.Substring(2).Contains("n")
              select p;
```

LinqExamples.cs: Splitting into two parts

```
Dim optimizedPart = From p As Person In container _
 Where p.Age > 18 AndAlso p.Age < 70 _
 Select p
Dim endResult = From p In optimizedPart.AsEnumerable() _
 Where p.Name.Substring(2).Contains("n") _
 Select p
```

LinqExamples.vb: Splitting into two parts

The first query contains only the parts which can be optimized. After that, use the AsEnumerable()-operator to force to run the rest of the query with LINQ to objects. This splitting will improve the performance significantly, since parts of the operation run as optimized query.

LINQ For Compact Framework

Compact Framework version 3.5 contains LINQ implementation for querying objects, however it does not contain the namespace System.Linq.Expressions, which is used by all optimized LINQ providers. Luckily there is a solution to this problem. Mono implementation of System.Core can be used to support optimized LINQ providers and expression interpreter contributed by Mainsoft to Mono's System.Core can be used to support full LINQ expression trees.

These assemblies were used by db4o team to compile a new assembly, System.Linq.Expressions.dll, which provides LINQ support to .NET Compact Framework 3.5. In order to use the full LINQ functionality including optimization, add a reference to System.Linq.Expressions.dll in your project and enjoy.

Native Queries

Wouldn't it be nice to write 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?

Since .NET 3.5 LINQ is the way to go and provides a extremely powerful query language. For older .NET versions native queries allow you do to this.

Native Queries are available for all platforms supported by db4o.

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 where possible and use internal query processor to run them against indexes and without instantiating actual objects.

Simple Example

Let's look at how a simple native query will look like. See also a collection of example queries.

```
IList<Pilot> result = container.Query(
    delegate(Pilot pilot) { return pilot.Name == "John"; });
```

NativeQueryExamples.cs: Check for equality of the name

```
Dim result As IList(Of Pilot) = container.Query(Of Pilot)(AddressOf QueryJohns)
```

NativeQueryExamples.vb: Check for equality of the name

Native Query Performance

There's one drawback of native queries: 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 flow-graph. 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 current state of the query optimization process is detailed in the chapter on Native Query Optimization

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

Native Query Examples

Here's a collection of native query examples. These queries assume that there's a Pilot class with a name and age and a Car class with a pilot and name.

Note that for .NET 3.5 or newer we recommend to use LINQ instead of native queries.

Equality

This query shows you how compare a property for equality. In this example we compare the name of a person.

```
IList<Pilot> result = container.Query(
    delegate(Pilot pilot) { return pilot.Name == "John"; });
```

NativeQueryExamples.cs: Check for equality of the name

```
Dim result As IList(Of Pilot) = container.Query(Of Pilot)(AddressOf QueryJohns)
```

NativeQueryExamples.vb: Check for equality of the name

```
Private Shared Function QueryJohns(ByVal pilot As Pilot)
    Return pilot.Name = "John"
End Function
```

NativeQueryExamples.vb: Query for John

Comparison

You can compare values with the usual comparison operators.

```
IList<Pilot> result = container.Query(
    delegate(Pilot pilot) { return pilot.Age < 18; });
```

NativeQueryExamples.cs: Compare values to each other

```
Dim result As IList(Of Pilot) = container.Query(Of Pilot)(AddressOf QueryAdults)
```

NativeQueryExamples.vb: Compare values to each other

```
Private Shared Function QueryAdults(ByVal pilot As Pilot)
    Return pilot.Age < 18
End Function
```

NativeQueryExamples.vb: Query for adults

Query For Value Range

Of course you can combine different comparisons. For example you can combine the greater and smaller than operators to check for a range of values.

```
IList<Pilot> result = container.Query(
    delegate(Pilot pilot) { return pilot.Age > 18 && pilot.Age < 30; });
```

NativeQueryExamples.cs: Query for a particular rage of values

```
Dim result As IList(Of Pilot) = container.Query(Of Pilot)(AddressOf QueryRange)
```

NativeQueryExamples.vb: Query for a particular rage of values

```
Private Shared Function QueryRange(ByVal pilot As Pilot)
    Return pilot.Age > 18 AndAlso pilot.Age < 30
End Function
```

NativeQueryExamples.vb: Query for range

Combine Check With Logical Operators

Of course you can combine a arbitrary set of conditions with logical operators.

```
IList<Pilot> result = container.Query(
    delegate(Pilot pilot)
        {
            return (pilot.Age > 18 && pilot.Age < 30)
                   || pilot.Name == "John";
        });
```

NativeQueryExamples.cs: Combine different comparisons with the logical operators

```
Dim result As IList(Of Pilot) = container.Query(Of Pilot)(AddressOf CombineCriterias)
```

NativeQueryExamples.vb: Combine different comparisons with the logical operators

```
Private Shared Function CombineCriterias(ByVal pilot As Pilot)
    Return (pilot.Age > 18 AndAlso pilot.Age < 30) OrElse pilot.Name = "John"
End Function
```

NativeQueryExamples.vb: Combine criterias

Query In Separate Method

You can implement your query in a separate method and then just us it where you need it. This is especially useful when you reuse the same query multiple times. Or you want to give your query a clear name for documentation purposes.

First write your method:

```
private static bool AllJohns(Pilot pilot)
{
    return pilot.Name == "John";
}
```

NativeQueryExamples.cs: Query as method

```
Private Shared Function AllJohns(ByVal pilot As Pilot) As Boolean
    Return pilot.Name = "John"
End Function
```

NativeQueryExamples.vb: Query as method

And then use it:

```
IList<Pilot> result = container.Query(new Predicate<Pilot>(AllJohns));
```

NativeQueryExamples.cs: Use the predefined query

```
Dim result As IList(Of Pilot) = container.Query(Of Pilot)(AddressOf AllJohns)
```

NativeQueryExamples.vb: Use the predefined query

Arbitrary Code

In principal your query can contain any code and can do the most complex comparisons. However in practice the are limitations. The simple queries are optimized and translated to SODA-queries. This is not possible for complex queries. If the query cannot be optimized, db4o will instantiate all objects and pass it to your query-object. This is a order of magnitude slower than a optimized native query and only feasible for smaller data sets.

```
IList<int> allowedAges = Array.AsReadOnly(new int[] {18, 20, 35});
IList<Pilot> result = container.Query(
    delegate(Pilot pilot)
        {
            return allowedAges.Contains(pilot.Age) ||
                   pilot.Name.ToLowerInvariant() == "John";
        });
```

NativeQueryExamples.cs: Arbitrary code

```
Dim result As IList(Of Pilot) = container.Query(Of Pilot)(AddressOf QueryWithAnyCode)
```

NativeQueryExamples.vb: Arbitrary code

```
Private Shared Function QueryWithAnyCode(ByVal pilot As Pilot)
    Dim allowedAges As IList(Of Integer) = Array.AsReadOnly(New Integer() {18, 20, 35})
    Return allowedAges.Contains(Pilot.Age) _
            OrElse Pilot.Name.ToLowerInvariant() = "John"
End Function
```

NativeQueryExamples.vb: Query with arbitrary code

Native Query Sorting

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

```
IList<Pilot> pilots = container.Query(
    delegate(Pilot p) { return p.Age > 18; },
    delegate(Pilot p1, Pilot p2) { return p1.Name.CompareTo(p2.Name); });
```

NativeQueriesSorting.cs: Native query with sorting

```
Dim pilots As IList(Of Pilot) = container.Query(Of Pilot)(AddressOf QueryForAdults, AddressOf SortByName)
```

NativeQueriesSorting.vb: Native query with sorting

```
Private Shared Function QueryForAdults(ByVal pilot As Pilot) As Boolean
    Return pilot.Age > 18
End Function

Private Shared Function SortByName(ByVal pilot1 As Pilot, ByVal pilot2 As Pilot) As Integer
    Return pilot1.Name.CompareTo(pilot2.Name)
End Function
```

NativeQueriesSorting.vb: Query and sort function

Native Query Optimization

Native queries will run out of the box in any environment. This optimization is turned on by default. Native queries will be converted to SODA where this is possible. This allows 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 the bytecode is 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:

```
IList<Pilot> result = container.Query(
    delegate(Pilot pilot) { return pilot.Name == "John"; });
```

NativeQueryExamples.cs: Check for equality of the name

```
Dim result As IList(Of Pilot) = container.Query(Of Pilot)(AddressOf QueryJohns)
```

NativeQueryExamples.vb: Check for equality of the name

```
Private Shared Function QueryJohns(ByVal pilot As Pilot)
    Return pilot.Name = "John"
End Function
```

NativeQueryExamples.vb: Query for John

First the signature of the given delegate is analyzed to find out the types. This is used to constrain the type in the SODA-query. Then the bytecode of query is analyzed to find out was it does. When the operations a simple and easy to convert, it will be transformed to complete SODA query:

```
IQuery query = container.Query();
query.Constrain(typeof (Pilot));
query.Descend("name").Constrain("John");

IObjectSet result = query.Execute();
```

SodaQueryExamples.cs: A simple constrain on a field

```
Dim query As IQuery = container.Query()
query.Constrain(GetType(Pilot))
query.Descend("name").Constrain("John")

Dim result As IObjectSet = query.Execute()
```

SodaQueryExamples.vb: A simple constrain on a field

Native Query Optimization

Native Query optimizer for .NET requires the following libraries to work: See "Dependency Overview" on page 365

- Db4objects.Db4o.Instrumentation.dll
- Db4objects.Db4o.NativeQueries.dll
- Mono.Cecil.dll
- Cecil.FlowAnalysis.dll
-

These assemblies must be available in your project for successful native query optimization.

Current optimization supports the following constructs well:

- Compile-time constants.
- Simple member access.
- Primitive comparisons.
- Equality operator.
- Contains, StartsWith- and EndsWith-methods for strings.
- Boolean expressions.

- Arbitrary method calls,including property accessors, on predicate fields (without any arguments).
- Candidate methods composed of the above.
- Chained combinations of the above.

Note that the current implementation doesn't support polymorphism yet.

The specifics of Compact Framework platform are explained in NQ Optimization On CF2.0.

An alternative optimization practice can be found in Build-time Optimization For .NET article.

For more information on native queries optimization see Monitoring Optimization.

Monitoring Optimization

In order to optimize native queries the bytecode is analyzed and converted into SODA queries. This task isn't easy. If there's any doubt in the correctness of the conversion db4o won't do it. In such cases db4o falls back and instantiates all objects and runs it against the query. This is a order of magnitude slower than optimized queries. Therefore you probably want to monitor the query optimization and be warned when a query isn't optimized. This is possible with the diagnostic listeners.

```
configuration.Common.Diagnostic.AddListener(new NativeQueryListener());
```

NativeQueryDiagnostics.cs: Use diagnostics to find unoptimized queries

```
configuration.Common.Diagnostic.AddListener(New NativeQueryListener())
```

NativeQueryDiagnostics.vb: Use diagnostics to find unoptimized queries

```
class NativeQueryListener :IDiagnosticListener
{
    public void OnDiagnostic(IDiagnostic diagnostic)
    {
        if(diagnostic is NativeQueryNotOptimized){
            Console.WriteLine("Query not optimized"+diagnostic);
        } else if(diagnostic is NativeQueryOptimizerNotLoaded){
            Console.WriteLine("Missing native query optimisation assemblies" + diagnostic);
        }
    }
}
```

NativeQueryDiagnostics.cs: The native query listener

```
Class NativeQueryListener
    Implements IDiagnosticListener
    Public Sub OnDiagnostic(ByVal diagnostic As IDiagnostic) Implements IDiagnosticListener.OnDiagnostic

        If TypeOf diagnostic Is NativeQueryNotOptimized Then
            Console.WriteLine("Query not optimized" & Convert.ToString(diagnostic))
        ElseIf TypeOf diagnostic Is NativeQueryOptimizerNotLoaded Then
            Console.WriteLine("Missing native query optimisation assemblies" & Convert.ToString(diagnostic))
        End If
    End Sub
End Class
```

NativeQueryDiagnostics.vb: The native query listener

You can register a diagnostic listener and check for certain messages. There are two messages related to the native query optimization. The first is the **NativeQueryNotOptimized**-message. This tells you that a query couldn't be optimized. Consider simplifying the query. The second is the **Nati-**

veQueryOptimizerNotLoaded-message. This message tells you that db4o couldn't find the libraries needed for the native query optimization. Check that you've included the assemblies-files you need.

Native Query Optimization At Build Time

If the platform you're running doesn't support optimization at runtime you can use the compile-time optimization. See "Enhancement Tools" on page 177

Create the Enhancement Task

First we define the enhancement-task. This task will process the assembly and enhance it.

You can add this to the existing project-files. The .csproj or .vbproj are actually MSBuild-files. Open them with a XML-Editor and add the needed parts.

```
<Target Name="AfterBuild">
  <Db4oEnhancerMSBuildTask Assemblies="@(Db4oEnhance)" />
</Target>
```

simple-enhance-example.csproj: Define a target which runs the task

And the execute the task after the compilation.

```
<Target Name="AfterBuild">
  <Db4oEnhancerMSBuildTask Assemblies="@(Db4oEnhance)" CommandLine="-nq" />
</Target>
```

simple-enhance-example.csproj: Only enhance native queries

Now Visual Studio will automatically run the tasks for each build. You don't need to change anything else.

Often it's practical to have all persistent classes in a separate project or compile unit. Then the enhancement script runs only for this project. This makes it easy to enhance only the classes for the persistent objects.

There are lot of possibilities to tweak and configure the build-time enhancement so that it fits your requirements. See "Build Time Enhancement"

NQ Optimization On CF2.0

Due to some platform limitations, CompactFramework 2.0 users using the more convenient delegate based Native Query syntax that want their queries to be optimized are required to run the Db4oTool.exe command line utility on their assemblies prior to deploying them. See "Enhancement Tools" on page 177

The utility, which can be found in the /bin folder of .NET distribution, is required because the CompactFramework API does not expose any of the delegate metadata needed by the Native Query optimizer. The tool works by augmenting the bytecode with the necessary delegate metadata and replacing ObjectContainer#Query invocations with invocations to a lower level method that makes use of the additional information.

Query By Example

Query By Example is a very special query method. Basically you pass in a example object to db4o. Then db4o searches the database for all objects which look alike.

The basic principal is like this. It goes through all fields of the example object. If a field doesn't have the default value it is used as constraint. It takes that value and checks for all objects which have an equal value for this field. If more than one field has a value, each value is used as constraint and combined. Here's a simple example.

```
Pilot theExample = new Pilot();
theExample.Name = "John";
IList result = container.QueryByExample(theExample);
```

QueryByExamples.cs: Query for John by example

```
Dim theExample As New Pilot()
theExample.Name = "John"
Dim result As IList = container.QueryByExample(theExample)
```

QueryByExamples.vb: Query for John by example

Take a look a the various examples of query by example examples. See "Query By Example Examples" on page 25

Query by Example has fundamental limitations, which limits its use cases. See "Query By Example Limitations" on page 27

Query By Example Examples

Query By Example Basics

For a query you pass in an example object to db4o. db4o will exanimate the object with reflection. Each field which doesn't have the default value will be use as a constrain. This means a number-field which isn't zero, a reference which isn't null or a string which isn't null. In this example we set the name to John. Then db4o will return all pilots with the name John.

```
Pilot theExample = new Pilot();
theExample.Name = "John";
IList result = container.QueryByExample(theExample);
```

QueryByExamples.cs: Query for John by example

```
Dim theExample As New Pilot()
theExample.Name = "John"
Dim result As IList = container.QueryByExample(theExample)
```

QueryByExamples.vb: Query for John by example

Or we set the age to 33 and db4o will return all 33 years old pilots.

```
Pilot theExample = new Pilot();
theExample.Age = 33;
IList result = container.QueryByExample(theExample);
```

QueryByExamples.cs: Query for 33 year old pilots

```
Dim theExample As New Pilot()
theExample.Age = 33
Dim result As IList = container.QueryByExample(theExample)
```

QueryByExamples.vb: Query for 33 year old pilots

Combine Values

When you set multiple values all will be used as constrain. For example when we set the name to Jo and the age to 29 db4o will return all pilots which are 29 years with the name Jo.

```
Pilot theExample = new Pilot();
theExample.Name = "Jo";
theExample.Age = 29;
IList result = container.QueryByExample(theExample);
```

QueryByExamples.cs: Query a 29 years old Jo

```
Dim theExample As New Pilot()
theExample.Name = "Jo"
theExample.Age = 29
Dim result As IList = container.QueryByExample(theExample)
```

QueryByExamples.vb: Query a 29 years old Jo

All Objects Of A Type

If you pass a empty example db4o will return all objects of that type.

```
Pilot example = new Pilot();
IList result = container.QueryByExample(example);
```

QueryByExamples.cs: All objects of a type by passing a empty example

```
Dim example As New Pilot()
Dim result As IList = container.QueryByExample(example)
```

QueryByExamples.vb: All objects of a type by passing a empty example

Alternatively you also can directly pass in the type.

```
IList result = container.QueryByExample(typeof (Pilot));
```

QueryByExamples.cs: All objects of a type by passing the type

```
Dim result As IList = container.QueryByExample(GetType(Pilot))
```

QueryByExamples.vb: All objects of a type by passing the type

All Objects

When you pass null all objects stored in the database will be returned.

```
IList result = container.QueryByExample(null);
```

QueryByExamples.cs: All objects

```
Dim result As IList = container.QueryByExample(Nothing)
```

QueryByExamples.vb: All objects

Nested Objects

You can also use nested objects as an example. For example with a car and a pilot. We can query for a car which has a pilot with certain constrains. In this example we get the cars which pilot is called Jenny.

```
Pilot pilotExample = new Pilot();
pilotExample.Name = "Jenny";

Car carExample = new Car();
carExample.Pilot = pilotExample;
IList result = container.QueryByExample(carExample);
```

QueryByExamples.cs: Nested objects example

```
Dim pilotExample As New Pilot()
pilotExample.Name = "Jenny"

Dim carExample As New Car()
carExample.Pilot = pilotExample
Dim result As IList = container.QueryByExample(carExample)
```

QueryByExamples.vb: Nested objects example

Contains Example

Collections and arrays act a little different. Query by example returns all object which have at least the items in the array or collection from the example. For example it returns the blog post which has the "db4o"-tag in its tag-collection.

```
BlogPost pilotExample = new BlogPost();
pilotExample.AddTags("db4o");
IList result = container.QueryByExample(pilotExample);
```

QueryByExamples.cs: Contains in collections

```
Dim pilotExample As New BlogPost()
pilotExample.AddTags("db4o")
Dim result As IList = container.QueryByExample(pilotExample)
```

QueryByExamples.vb: Contains in collections

Structured Contains

You can even check that a item in a collection fulfills certain criteria's. For example we can check that the blog post has an author with the name John in its author-collection.

```
BlogPost pilotExample = new BlogPost();
pilotExample.AddAuthors(new Author("John"));
IList result = container.QueryByExample(pilotExample);
```

QueryByExamples.cs: Structured contains

```
Dim pilotExample As New BlogPost()
pilotExample.AddAuthors(New Author("John"))
Dim result As IList = container.QueryByExample(pilotExample)
```

QueryByExamples.vb: Structured contains

Query By Example Limitations

Query By Example has by design a lot of limitations:

- You cannot perform advanced query expressions. (OR, NOT, etc.)
- You cannot constrain for default-values like 0 on numbers, empty strings or nulls on references because they would be interpreted as unconstrained.
- You need a constructor to create objects without initialized fields.
- Complex queries by example are not easy to read and interpret what they query for.

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 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.
- Native Queries use bytecode analysis to convert to SODA
- LINQ-queries are also converted to SODA-Queries

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

Take a look at the SODA-examples to get a feel for SODA-API. See "SODA Query Examples" on page 28

Also SODA has special capabilities for certain types like collections etc. See "SODA Special Cases Examples" on page 32

Additionally SODA evaluations can help you implementing queries which go beyond the capabilities of pure SODA queries. See "SODA Evaluations" on page 35

At last, you need a way to sort the results of a query. See "SODA Sorting" on page 38

SODA Query Examples

Here's a collection of SODA-query examples. These queries assume that there's a Pilot class with a name and age, a Car class with a pilot and name and a BlogPost class with list of tags, authors and a Dictionary of meta-data.

There are also a few examples for special cases.

Type Constraint

This is the most basic and most used constraint for SODA-queries. SODA acts like a filter on all stored objects. But usually you're only interested for instances of a certain type. Therefore you need to constrain the type of the result.

```
IQuery query = container.Query();
query.Constrain(typeof(Pilot));

IObjectSet result = query.Execute();
```

SodaQueryExamples.cs: Type constrain for the objects

```
Dim query As IQuery = container.Query()
query.Constrain(GetType(Pilot))

Dim result As IObjectSet = query.Execute()
```

SodaQueryExamples.vb: Type constrain for the objects

Field Constraint

You can add constrains on fields. This is done by descending into a field and constrain the value of that field. By default the constrain is an equality comparison.

```
IQuery query = container.Query();
query.Constrain(typeof (Pilot));
query.Descend("name").Constrain("John");

IObjectSet result = query.Execute();
```

SodaQueryExamples.cs: A simple constrain on a field

```
Dim query As IQuery = container.Query()
query.Constrain(GetType(Pilot))
query.Descend("name").Constrain("John")

Dim result As IObjectSet = query.Execute()
```

SodaQueryExamples.vb: A simple constrain on a field

Comparisons

You can do comparison on the field-values. For example to check if something is greater or smaller than something else.

```
IQuery query = container.Query();
query.Constrain(typeof (Pilot));
query.Descend("age").Constrain(42).Greater();

IObjectSet result = query.Execute();
```

SodaQueryExamples.cs: A greater than constrain

```
Dim query As IQuery = container.Query()
query.Constrain(GetType(Pilot))
query.Descend("age").Constrain(42).Greater()

Dim result As IObjectSet = query.Execute()
```

SodaQueryExamples.vb: A greater than constrain

```
IQuery query = container.Query();
query.Constrain(typeof (Pilot));
query.Descend("age").Constrain(42).Greater().Equal();

IObjectSet result = query.Execute();
```

SodaQueryExamples.cs: A greater than or equals constrain

```
Dim query As IQuery = container.Query()
query.Constrain(GetType(Pilot))
query.Descend("age").Constrain(42).Greater().Equal()

Dim result As IObjectSet = query.Execute()
```

SodaQueryExamples.vb: A greater than or equals constrain

Combination of Constraints (AND, OR)

You can combine different constraints with an 'AND' or 'OR' condition. By default all constrains are combined with the 'AND' condition.

```
IQuery query = container.Query();
query.Constrain(typeof (Pilot));
query.Descend("age").Constrain(42).Greater()
    .Or(query.Descend("age").Constrain(30).Smaller());

IObjectSet result = query.Execute();
```

SodaQueryExamples.cs: Logical combination of constrains

```
Dim query As IQuery = container.Query()
query.Constrain(GetType(Pilot))
query.Descend("age").Constrain(42).Greater().[Or](query.Descend("age").Constrain(30).Smaller())

Dim result As IObjectSet = query.Execute()
```

SodaQueryExamples.vb: Logical combination of constrains

Not-Constrain

You can invert a constrain by calling the not-method.

```
IQuery query = container.Query();
query.Constrain(typeof (Pilot));
query.Descend("age").Constrain(42).Not();

IObjectSet result = query.Execute();
```

SodaQueryExamples.cs: Not constrain

```
Dim query As IQuery = container.Query()
query.Constrain(GetType(Pilot))
query.Descend("age").Constrain(42).Not()

Dim result As IObjectSet = query.Execute()
```

SodaQueryExamples.vb: Not constrain

String Comparison

There are special compare operations for strings. By default strings are compared by equality and the comparison is case sensitive.

There's the contains-comparison which checks if a field contains a substring. The like-comparison is the case-insensitive version of the contains-comparison.

Also a start-with- and a ends-with-caparison is available for strings. For this you can specify if the comparison is case sensitive or not.

```
IQuery query = container.Query();
query.Constrain(typeof (Pilot));
// First strings, you can use the contains operator
query.Descend("name").Constrain("oh").Contains()
    // Or like, which is like .contains(), but case insensitive
    .Or(query.Descend("name").Constrain("AnN").Like())
    // The .endsWith and .startWith constrains are also there,
    // the true for case-sensitive, false for case-insensitive
    .Or(query.Descend("name").Constrain("NY").EndsWith(false));

IObjectSet result = query.Execute();
```

SodaQueryExamples.cs: String comparison

```
Dim query As IQuery = container.Query()
query.Constrain(GetType(Pilot))
' First strings, you can use the contains operator
' Or like, which is like .contains(), but case insensitive
' The .endsWith and .startWith constrains are also there,
' the true for case-sensitive, false for case-insensitive
query.Descend("name").Constrain("oh").Contains().[Or](query.Descend("name").Constrain("AnN").[Like]()).[Or](query

Dim result As IObjectSet = query.Execute()
```

SodaQueryExamples.vb: String comparison

Compare Field With Existing Object

When you have a reference type field, you can compare this field with a certain object. It will compare the field and the object by object identity.

Note that this comparison only works with stored objects. When you use a not yet stored object as constrain, it will use query by example. To force a comparison by object identity, you can add a .Identiy() call.

```
Pilot pilot = container.Query<Pilot>()[0];

IQuery query = container.Query();
query.Constrain(typeof(Car));
// if the given object is stored, its compared by identity
query.Descend("pilot").Constrain(pilot);

IObjectSet carsOfPilot = query.Execute();
```

SodaQueryExamples.cs: Compare with existing object

```
Dim pilot As Pilot = container.Query(Of Pilot)()(0)

Dim query As IQuery = container.Query()
query.Constrain(GetType(Car))
' if the given object is stored, its compared by identity
query.Descend("pilot").Constrain(pilot)

Dim carsOfPilot As IObjectSet = query.Execute()
```

SodaQueryExamples.vb: Compare with existing object

Descend Deeper Into Objects

You can descend deeper into the objects by following fields. This allows you to setup complex constraints on nested objects. Note that the deeper you descend into the objects, the more expensive the query is to execute.

```
IQuery query = container.Query();
query.Constrain(typeof (Car));
query.Descend("pilot").Descend("name").Constrain("John");

IObjectSet result = query.Execute();
```

SodaQueryExamples.cs: Descend over multiple fields

```
Dim query As IQuery = container.Query()
query.Constrain(GetType(Car))
query.Descend("pilot").Descend("name").Constrain("John")

Dim result As IObjectSet = query.Execute()
```

SodaQueryExamples.vb: Descend over multiple fields

SODA Special Cases Examples

This topic contains a examples which demonstrate special behavior for some types in SODA. Take also a look at the other SODA examples.

Contains on Collections and Arrays

Collections and arrays have a special behavior in SODA to make them easier to query. For example you can simple use a constrain directly on a collection-field to check if it contains that value.

Note that currently collections cannot be indexed and therefore such a constrain can be slow on a large data set.

```
IQuery query = container.Query();
query.Constrain(typeof (BlogPost));
query.Descend("tags").Constrain("db4o");

IObjectSet result = query.Execute();
```

SodaQueryExamples.cs: Collection contains constrain

```
Dim query As IQuery = container.Query()
query.Constrain(GetType(BlogPost))
query.Descend("tags").Constrain("db4o")

Dim result As IObjectSet = query.Execute()
```

SodaQueryExamples.vb: Collection contains constrain

Constrains on Collection Members

When you have a collection or array field, you can simply descend further to the collection-member fields. This allows you query for a object, which has a collection and certain objects in that collection.

Note that currently collections cannot be indexed and therefore such a constrain can be slow on a large data set.

```
IQuery query = container.Query();
query.Constrain(typeof(BlogPost));
query.Descend("authors").Descend("name").Constrain("Jenny");

IObjectSet result = query.Execute();
```

SodaQueryExamples.cs: Descend into collection members

```
Dim query As IQuery = container.Query()
query.Constrain(GetType(BlogPost))
query.Descend("authors").Descend("name").Constrain("Jenny")

Dim result As IObjectSet = query.Execute()
```

SodaQueryExamples.vb: Descend into collection members

Contains Key on Dictionarys

You can check a dictionary if it contains a certain key. Similar to collections, you just can directly use a constrain on the collection field. This will compare the value with the keys of the Dictionary.

Note that currently collections cannot be indexed and therefore such a constrain can be slow on a large data set.

```
IQuery query = container.Query();
query.Constrain(typeof (BlogPost));
query.Descend("metaData").Constrain("source");

IObjectSet result = query.Execute();
```

SodaQueryExamples.cs: Dictionary contains a key constrain

```
Dim query As IQuery = container.Query()
query.Constrain(GetType(BlogPost))
query.Descend("metaData").Constrain("source")

Dim result As IObjectSet = query.Execute()
```

SodaQueryExamples.vb: Dictionary contains a key constrain

Return the Objects of a Field

With SODA you can navigate to a field and return the objects of that field. Note that this only works for reference objects and not for value objects like strings and numbers.

```
IQuery query = container.Query();
query.Constrain(typeof(Car));
query.Descend("name").Constrain("Mercedes");

// returns the pilot of these cars
IObjectSet result = query.Descend("pilot").Execute();
```

SodaQueryExamples.cs: Return the object of a field

```
Dim query As IQuery = container.Query()
query.Constrain(GetType(Car))
query.Descend("name").Constrain("Mercedes")

' returns the pilot of these cars
Dim result As IObjectSet = query.Descend("pilot").Execute()
```

SodaQueryExamples.vb: Return the object of a field

Mixing With Query By Example

When you have a reference type field, you can also use a query by example constrain for that field. Pass a new object as an example for this.

Note that when you pass a persisted object, it will compare it by object identity and not use it as example. You can force this behavior by adding an explicit by example constrain.

```
IQuery query = container.Query();
query.Constrain(typeof(Car));
// if the given object is not stored,
// it will behave like query by example for the given object
Pilot examplePilot = new Pilot(null, 42);
query.Descend("pilot").Constrain(examplePilot);

IObjectSet carsOfPilot = query.Execute();
```

SodaQueryExamples.cs: Mix with query by example

```
Dim query As IQuery = container.Query()
query.Constrain(GetType(Car))
' if the given object is not stored,
' it will behave like query by example for the given object
Dim examplePilot As New Pilot(Nothing, 42)
query.Descend("pilot").Constrain(examplePilot)

Dim carsOfPilot As IObjectSet = query.Execute()
```

SodaQueryExamples.vb: Mix with query by example

Dynamically Typed

SODA is a dynamically query language. By default SODA acts like a filter on all stored objects. You just add constrains which filters the objects to the desired output.

An example for this behavior: You just add an field-constraint without any type-constrain on the object. This will return all objects which have such a field and match the constrain.

```
IQuery query = container.Query();
// You can simple filter objects which have a certain field
query.Descend("name").Constrain(null).Not();

IObjectSet result = query.Execute();
```

SodaQueryExamples.cs: Pure field constrains

```
Dim query As IQuery = container.Query()
' You can simple filter objects which have a certain field
query.Descend("name").Constrain(Nothing).[Not]()

Dim result As IObjectSet = query.Execute()
```

SodaQueryExamples.vb: Pure field constrains

This also means that you can query for not existing fields. SODA will not complain if a field doesn't exist. Instead it won't return any object, because no object could satisfy the constrain.

```
IQuery query = container.Query();
query.Constrain(typeof (Pilot));
// using not existing fields doesn't throw an exception
// but rather exclude all object which don't use this field
query.Descend("notExisting").Constrain(null).Not();

IObjectSet result = query.Execute();
```

SodaQueryExamples.cs: Using not existing fields excludes objects

```
Dim query As IQuery = container.Query()
query.Constrain(GetType(Pilot))
' using not existing fields doesn't throw an exception
' but rather exclude all object which don't use this field
query.Descend("notExisting").Constrain(Nothing).[Not]()

Dim result As IObjectSet = query.Execute()
```

SodaQueryExamples.vb: Using not existing fields excludes objects

SODA Evaluations

Sometimes the capabilities of regular SODA-queries is not enough. In such cases you can add evaluations to the SODA-query. A evaluation is a piece of code which runs against objects.

To use a evaluation, you need to pass an instance of the IEvaluation-interface as a constrain. db4o will call the match-method of that interface. Implement the match-method of the IEvaluation-interface. In the match-method you can get the candidate-object and the object-container. Compare the object and when it matches, pass true to the include-method. Otherwise pass false.

While SODA evaluations are extremely powerful they are also slow. In order to run the evaluation the objects need to be instantiated from the database and then processed by the evaluator. This means that you should use evaluations only when there's no other possibility.

Simple Evaluation

Here's an example for a simple evaluation. This evaluation filters pilots by the age and picks only pilots with an odd-number as age.

First we need to create the evaluation-class:

```
class OnlyOddAge : IEvaluation
{
    public void Evaluate(ICandidate candidate)
    {
        Pilot pilot = (Pilot)candidate.GetObject();
        candidate.Include(pilot.Age % 2 != 0);
    }
}
```

SodaEvaluationExamples.cs: Simple evaluation which includes only odd aged pilots

```
Class OnlyOddAge
    Implements IEvaluation
    Public Sub Evaluate(ByVal candidate As ICandidate) _
        Implements IEvaluation.Evaluate

        Dim pilot As Pilot = DirectCast(candidate.GetObject(), Pilot)
        candidate.Include(pilot.Age Mod 2 <> 0)
    End Sub
End Class
```

SodaEvaluationExamples.vb: Simple evaluation which includes only odd aged pilots

After that, you can use the evaluation in the SODA-query. An evaluation is added as a regular constrain.

```
IQuery query = container.Query();
query.Constrain(typeof (Pilot));
query.Constrain(new OnlyOddAge());

IObjectSet result = query.Execute();
```

SodaEvaluationExamples.cs: Simple evaluation

```
Dim query As IQuery = container.Query()
query.Constrain(GetType(Pilot))
query.Constrain(New OnlyOddAge())

Dim result As IObjectSet = query.Execute()
```

SodaEvaluationExamples.vb: Simple evaluation

Evaluation on Field

It's also possible to use the evaluation on a certain field. For this you descend into the field on which the evaluation should be applied. After that, specify the evaluation as a constrain on that field.

```
IQuery query = container.Query();
query.Constrain(typeof (Car));
query.Descend("pilot").Constrain(new OnlyOddAge());

IObjectSet result = query.Execute();
```

SodaEvaluationExamples.cs: Evaluation on field

```
Dim query As IQuery = container.Query()
query.Constrain(GetType(Car))
query.Descend("pilot").Constrain(New OnlyOddAge())

Dim result As IObjectSet = query.Execute()
```

SodaEvaluationExamples.vb: Evaluation on field

Regex on Fields

Evaluation also allow you to add very specific additional query capabilities. On of the most useful ones is regular expressions. First create a regular expression evaluation:

```
class RegexConstrain : IEvaluation
{
    private readonly Regex pattern;

    public RegexConstrain(String pattern)
    {
        this.pattern = new Regex(pattern);
    }

    public void Evaluate(ICandidate candidate)
    {
        string stringValue = (string)candidate.GetObject();
        candidate.Include(pattern.IsMatch(stringValue));
    }
}
```

SodaEvaluationExamples.cs: Regex Evaluator

```
Class RegexConstrain
    Implements IEvaluation
    Private ReadOnly pattern As Regex

    Public Sub New(ByVal pattern As [String])
        Me.pattern = New Regex(pattern)
    End Sub

    Public Sub Evaluate(ByVal candidate As ICandidate) _
        Implements IEvaluation.Evaluate
        Dim stringValue As String = DirectCast(candidate.GetObject(), String)
        candidate.Include(pattern.IsMatch(stringValue))
    End Sub
End Class
```

SodaEvaluationExamples.vb: Regex Evaluator

After that you can use it on any string field:

```
IQuery query = container.Query();
query.Constrain(typeof (Pilot));
query.Descend("name").Constrain(new RegexConstrain("J.*nn.*"));

IObjectSet result = query.Execute();
```

SodaEvaluationExamples.cs: Regex-evaluation on a field

```
Dim query As IQuery = container.Query()
query.Constrain(GetType(Pilot))
query.Descend("name").Constrain(New RegexConstrain("J.*nn.*"))

Dim result As IObjectSet = query.Execute()
```

SodaEvaluationExamples.vb: Regex-evaluation on a field

SODA Sorting

You can specify to sort by certain fields in SODA. For this you need to descend to the field and use the appropriate order ascending or order descending method.

In cases where this is not enough, you can use a special comparator.

Sorting by Field

To sort by a field navigate to the field and call a order ascending or descending method. Note that this only works for fields which have natural sortable values, such as strings and numbers.

```
IQuery query = container.Query();
query.Constrain(typeof(Pilot));
query.Descend("name").OrderAscending();

IObjectSet result = query.Execute();
```

SodaSorting.cs: Order by a field

```
Dim query As IQuery = container.Query()
query.Constrain(GetType(Pilot))
query.Descend("name").OrderAscending()

Dim result As IObjectSet = query.Execute()
```

SodaSorting.vb: Order by a field

Sort by Multiple Fields

You can sort by multiple fields. Add a order constrain for each field. The first order statement has the highest priority and last added the lowest.

```
IQuery query = container.Query();
query.Constrain(typeof(Pilot));
query.Descend("age").OrderAscending();
query.Descend("name").OrderAscending();

IObjectSet result = query.Execute();
```

SodaSorting.cs: Order by multiple fields

```
Dim query As IQuery = container.Query()
query.Constrain(GetType(Pilot))
query.Descend("age").OrderAscending()
query.Descend("name").OrderAscending()

Dim result As IObjectSet = query.Execute()
```

SodaSorting.vb: Order by multiple fields

Sort With Your Own Comperator

In cases where you have more complex sorting requirements, you can specify your own comparator. It is used like a regular .NET-comparator.

```
IQuery query = container.Query();
query.Constrain(typeof(Pilot));
query.SortBy(new NameLengthComperator());

IObjectSet result = query.Execute();
```

SodaSorting.cs: Order by your comparator

```
Dim query As IQuery = container.Query()
query.Constrain(GetType(Pilot))
query.SortBy(New NameLengthComperator())

Dim result As IObjectSet = query.Execute()
```

SodaSorting.vb: Order by your comparator

```
class NameLengthComperator :IQueryComparator
{
    public int Compare(object first, object second)
    {
        Pilot p1 = (Pilot) first;
        Pilot p2 = (Pilot)second;
        // sort by string-length
        return Math.Sign(p1.Name.Length - p2.Name.Length);
    }
}
```

SodaSorting.cs: The string length comperator

```
Private Class NameLengthComperator
    Implements IQueryComparator
    Public Function Compare(ByVal first As Object, ByVal second As Object) As Integer _
        Implements IQueryComparator.Compare

        Dim p1 As Pilot = DirectCast(first, Pilot)
        Dim p2 As Pilot = DirectCast(second, Pilot)
        ' sort by string-length
        Return Math.Sign(p1.Name.Length - p2.Name.Length)
    End Function
End Class
```

SodaSorting.vb: The string length comperator

SODA Processing

The SODA processing runs in two stages.

First Stage: Index-Lookups

First SODA tries to find the best fitting index for the query. It looks up all available field- and class-indexes. Then it goes through all possible indexes and chooses the index with the smallest candidate result set.

In practice this means that SODA first uses the field-indexes. If no suitable field index is found it falls back to the class-index, which contains all objects of a certain type.

With this index a first set of candidates is created. This means SODA looks up in the index the objects and returns the IDs of the candidate-objects.

Second Stage: Constrain Evaluation

The second stage uses the ids from the first stage and runs all those objects against the given constrains. If the constrains are regular SODA-constrains, it directly uses the values in the database to compare it. If additional evaluations are present, SODA will instantiate the candidate-objects and pass it to the evaluation-function.

Finally SODA will apply the sorting and then return all IDs of objects which match the query criteria.

ACID Properties And Transactions

The ACID properties are one of the oldest and most important concepts of database theory. It sets out the requirements for the database reliability:

- **Atomicity**: This means that must follow an "all or nothing" rule. Each transaction is either successfully completed or in case of failure the state of the database isn't changed at all.
  For example in a bank transfer transaction there are two steps: 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.
- **Consistency**: Consistency ensures that the database stays always in a consistent state. Each transaction takes the database from one consistent state to the next consistent state.
- **Isolation**: Isolation means that different operations cannot access modified data from another transaction that has not yet completed. There are different isolation-models. See "Isolation" on page 42
- **Durability**: This just refers to the real goal of any data store. It just means that the data should be persistently stored.

db4o fulfills the ACID properties. Each object container has its own transaction. Each transaction is a unit of work and ensures the ACID properties. This means, that a db4o transaction is an atomic operation. Either all changes of the db4o transactions are committed and made persistent. Or in case of a failure or rollback no state is changed. The database is kept consistent even on application or database crashes. And the db4o transactions are isolated from each other. See "db4o Transactions" on page 40

db4o Transactions

All db4o operations are transactional and there's always a transaction running. Each object container has its own transaction running. The transaction is started implicitly.

You can commit the transaction at any time. When the commit-call returns, all changes are made persistent.

Commit A Transactions

In order to commit a transaction, you need to call the commit-method. This will make all changes of the current transaction persistent. When the commit call is finished, everything is safely stored. If something goes wrong during the commit-operation or the commit-operation is interrupted (power-off, crash etc) the database has the state of either before or after the commit-call.

```
container.Store(new Pilot("John"));
container.Store(new Pilot("Joanna"));

container.Commit();
```

Transactions.cs: Commit changes

```
container.Store(New Pilot("John"))
container.Store(New Pilot("Joanna"))

container.Commit()
```

Transactions.vb: Commit changes

Rollback A Transaction

Of course you also can rollback a transaction. Just call rollback on the object container.

```
container.Store(new Pilot("John"));
container.Store(new Pilot("Joanna"));

container.Rollback();
```

Transactions.cs: Rollback changes

```
container.Store(New Pilot("John"))
container.Store(New Pilot("Joanna"))

container.Rollback()
```

Transactions.vb: Rollback changes

Note that when you rollback the changes, db4o won't rollback the objects in memory. All objects in memory will keep the state. If you want to make sure that objects in memory have the same state as in the database, you need to refresh the objects.

```
Pilot pilot = container.Query<Pilot>()[0];
pilot.Name = "New Name";
container.Store(pilot);
container.Rollback();

// use refresh to return the in memory objects back
// to the state in the database.
container.Ext().Refresh(pilot, int.MaxValue);
```

Transactions.cs: Refresh objects after rollback

```
Dim pilot As Pilot = container.Query(Of Pilot)()(0)
pilot.Name = "New Name"
container.Store(pilot)
container.Rollback()

' use refresh to return the in memory objects back
' to the state in the database.
container.Ext().Refresh(pilot, Integer.MaxValue)
```

Transactions.vb: Refresh objects after rollback

Implicit Commits

db4o commits implicitly when you close the object-container. The assumption is that normally you want to make the changes persistent when you close the object container. That's why it commits automatically. When you want to prevent this you should rollback the transaction before closing the container,

Multiple Concurrent Transactions

db4o transactions are always bound to their object container. When you want multiple concurrent transactions, you need to open multiple object containers. You can easily do this with the open session method. See "Session Containers" on page 164

Note that in this mode, db4o uses the read committed isolation. See "Isolation" on page 42

```
using (IObjectContainer rootContainer = Db4oEmbedded.OpenFile(DatabaseFileName))
{
    // open the db4o-session. For example at the beginning for a web-request
    using (IObjectContainer session = rootContainer.Ext().OpenSession())
    {
        // do the operations on the session-container
        session.Store(new Person("Joe"));
    }
}
```

Db4oSessions.cs: Session object container

```
Using rootContainer As IObjectContainer = Db4oEmbedded.OpenFile(DatabaseFileName)
    ' open the db4o-session. For example at the beginning for a web-request
    Using session As IObjectContainer = rootContainer.Ext().OpenSession()
        ' do the operations on the session-container
        session.Store(New Person("Joe"))
    End Using
End Using
```

Db4oSessions.vb: Session object container

Isolation

Isolation imposes rules which ensure that transactions do not interfere with each other even if they are executed at the same time. If two or more transactions are executed at the same time, they must be executed in a way so that transaction A won't be impacted by the intermediate data of transaction B. 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. 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. This is the model db4o uses.
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.

Read more about isolation levels on Wikipedia.

Dangerous Practices

db4o lets you configure a lot of low level details. It even lets you configure things which endanger the safety of the database integrity and transaction integrity. You should avoid these settings and only use them in very special cases.

One dangerous setting is disabling file-flushes. When you add the non-flushing decorator you get better performance. However due to the missing file-flushes, the ACID-properties cannot be guaranteed.

```
IStorage fileStorage = new FileStorage();
configuration.File.Storage = new NonFlushingStorage(fileStorage);
```

DangerousPractises.cs: Using the non-flushing storage weakens the ACID-properties

```
Dim fileStorage As IStorage = New FileStorage()
configuration.File.Storage = New NonFlushingStorage(fileStorage)
```

DangerousPractises.vb: Using the non-flushing storage weakens the ACID-properties

Another setting which endangers the ACID properties is disabling the commit-recovery. 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.

```
configuration.File.DisableCommitRecovery();
```

DangerousPractises.cs: Disabling commit-recovery weakens the ACID-properties

```
configuration.File.DisableCommitRecovery()
```

DangerousPractises.vb: Disabling commit-recovery weakens the ACID-properties

db4o's Commit Process

How does db4o make transactions safe, so that it can recover failures? Here's the short overview of the transaction-phases db4o uses.

Phase	In Case Of A Crash
1. During the transactions. New and updated objects are written to a new Slot in the database-file. The id-mapping and freespace changes are kept in the transaction.	The changes are lost, because the id-mapping and freespace changes weren't persisted. Therefore the changes are invisible to the database. The transaction is rolled back.
2. Committing starts: The id-changes and	The changes are lost, because the id-mapping and freespace changes haven't been

free-space changes are written to a new slot, without damaging the old information.	completely stored. The transaction is rolled back.
3. Write the location of latest id-records, and free-space changes to the first location with and additional checksum.	If the record write was completed, the transaction is resumed and completed. If not, the old information is used.
4. Write the of latest id-recods, and free-space changes to the backup location with and additional checksum.	If the record wasn't completely written, the transaction is resumed.

Of course you don't need to worry about this. db4o ensures that a transaction either completes or is rolled back. Whenever you call commit and the call succeeds, all changes are persisted. If your application or db4o crashes before a successfully commit-call, all changes are undone.

Identity Concept

You've maybe noticed that you don't need to add an identifier to your objects in order to store them with db4o. So how does db4o manage objects? db4o uses the object-identity to identify objects. db4o ensures that each stored object in the database has only one in memory representation per object container. If you load an object in different ways db4o will always return the same object. Or as rule of thumb: The objects in the database behave like objects in memory. When you run multiple queries or retrieve objects in another way, the same object in the database will always represented by the same object in memory.

```
Car theCar = container.Query<Car>()[0];
Pilot thePilot = container.Query<Pilot>()[0];
Pilot pilotViaCar = theCar.Pilot;
AssertTrue(thePilot == pilotViaCar);
```

IdentityConcepts.cs: db4o ensures reference equality

```
Dim theCar As Car = container.Query(Of Car)()(0)
Dim thePilot As Pilot = container.Query(Of Pilot)()(0)
Dim pilotViaCar As Pilot = theCar.Pilot
```

IdentityConcepts.vb: db4o ensures reference equality

In order to implement this behavior each object container keeps a mapping between the objects in memory and the stored object representation. When you load the same object with multiple object-containers (for example with session-containers or in client-server-mode), it will have different in memory-identity. db4o ensures the same identity only for a single object-container.

```
Car loadedWithContainer1 = container1.Query<Car>()[0];
Car loadedWithContainer2 = container2.Query<Car>()[0];
AssertFalse(loadedWithContainer1 == loadedWithContainer2);
```

IdentityConcepts.cs: Loading with different object container results in different objects

```
Dim loadedWithContainer1 As Car = container1.Query(Of Car)()(0)
Dim loadedWithContainer2 As Car = container2.Query(Of Car)()(0)
```

IdentityConcepts.vb: Loading with different object container results in different objects

This also means that an object should always be processed with the same object-container. When you load a object in one container and store it with another container db4o cannot recognize the object and will store it as a completely new object. Therefore you need to use the same container to store and load objects.

The identity concept works really well for desktop and embedded applications where you can have a single object container and keep that container open while the application is running. In such a case the behavior is just like you would work with regular objects. However this behavior doesn't work where you need to serialize objects, for example in web-applications. In such scenarios you need to do some extra work. See "Disconnected Objects" on page 227

Further Information

Maybe your wondering why db4o manages object by identity. Why not by equality? There are good reasons why this is the case. See "Identity Vs Equality" on page 45

In order to manage objects by identity db4o has a reference cache which contains all loaded objects. See "The Reference Cache" on page 45

Identity Vs Equality

One of the most common questions is why db4o doesn't allow to use equals and hash code to identify objects in the database. From the first glance it seems like a very attractive idea to 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 a Pilot and a Car class and their equals-method is based on comparing field values:

1. Store a pilot with the name 'Joe' and a car with that pilot in the database
2. Retrieve the pilot.
3. Change the pilot-name from 'Joe' to 'John'. Note that though it is the same object from the runtime point of view, these are two different objects for the database based on equals comparison.
4. Now what happens when we load the pilot. Should it return a pilot with the original name 'Joe'. Or the update pilot with the 'John'? What happens if there are hundreds of pilots witch had a pilot with the name 'Joe'. Do all those cars return the new Pilot name? Or the old one? How do you update only the name of a Pilot for only one car?

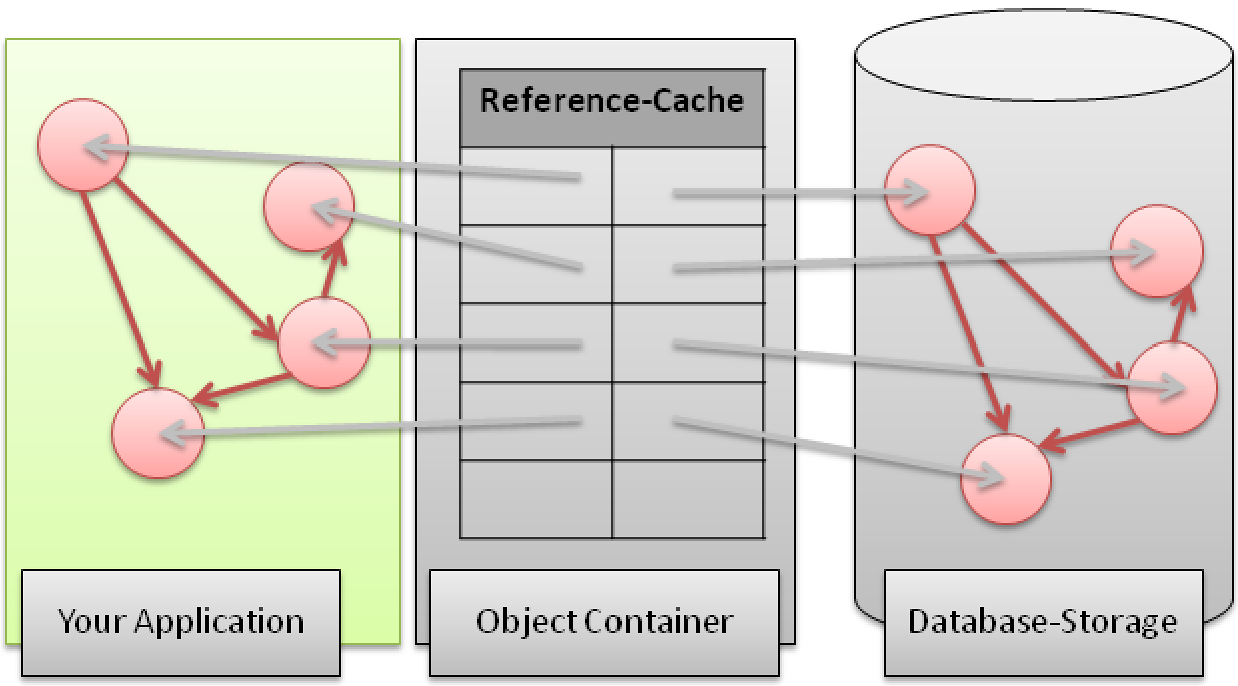
This questions shows that the update-issue is not solvable when the database manages objects by equality. Objects without identity also make Transparent Persistence and Activation impossible, as there will be no way to decide which instance is the right one for update or activation.

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.

The Reference Cache

We know that db4o manages objects by identity. But how does db4o recognize objects? How does it know if it needs to update a object? To archive this, db4o has a reference-cache. This is a table which maps objects in memory to their internal id. The internal id is used to find the object on disk.

Since this table has a reference to the object in memory it also acts as cache. Therefore it's called reference cache. When you load objects, db4o will fist lookup in the reference cache to get objects from there. This avoids loading the data from the disk and also returns the local state of the object. If object isn't in the reference cache, db4o will load it from disk.



Weak Reference

By default db4o uses weak references in the reference cache. While your application has at least one references to an object, the reference cache has reference to it. But as soon as your application has no reference to the object anymore, it can be collected by the garbage collector. db4o will never prevent any object from being garbage collected. In the end persisted objects are garbage collected like any other objects.

To keep the cache clean, db4o does periodically remove all empty weak references. You can configure that clean-up interval. See "Weak Reference Collection Interval" on page 120

You even can disable the weak reference. See "Disable Weak References" on page 120. Then db4o holds regular references to your objects.This prevents the objects from being garbage collected. This means that you need to remove object from the reference cache manually. Or only use short living object containers. See "Session Containers" on page 164

Manually Remove A Object From The Reference Cache

You can manually remove a object from the reference cache. This only required when you have disabled the weak-references.

```
Car theCar = container.Query<Car>()[0];
container.Ext().Purge(theCar);
```

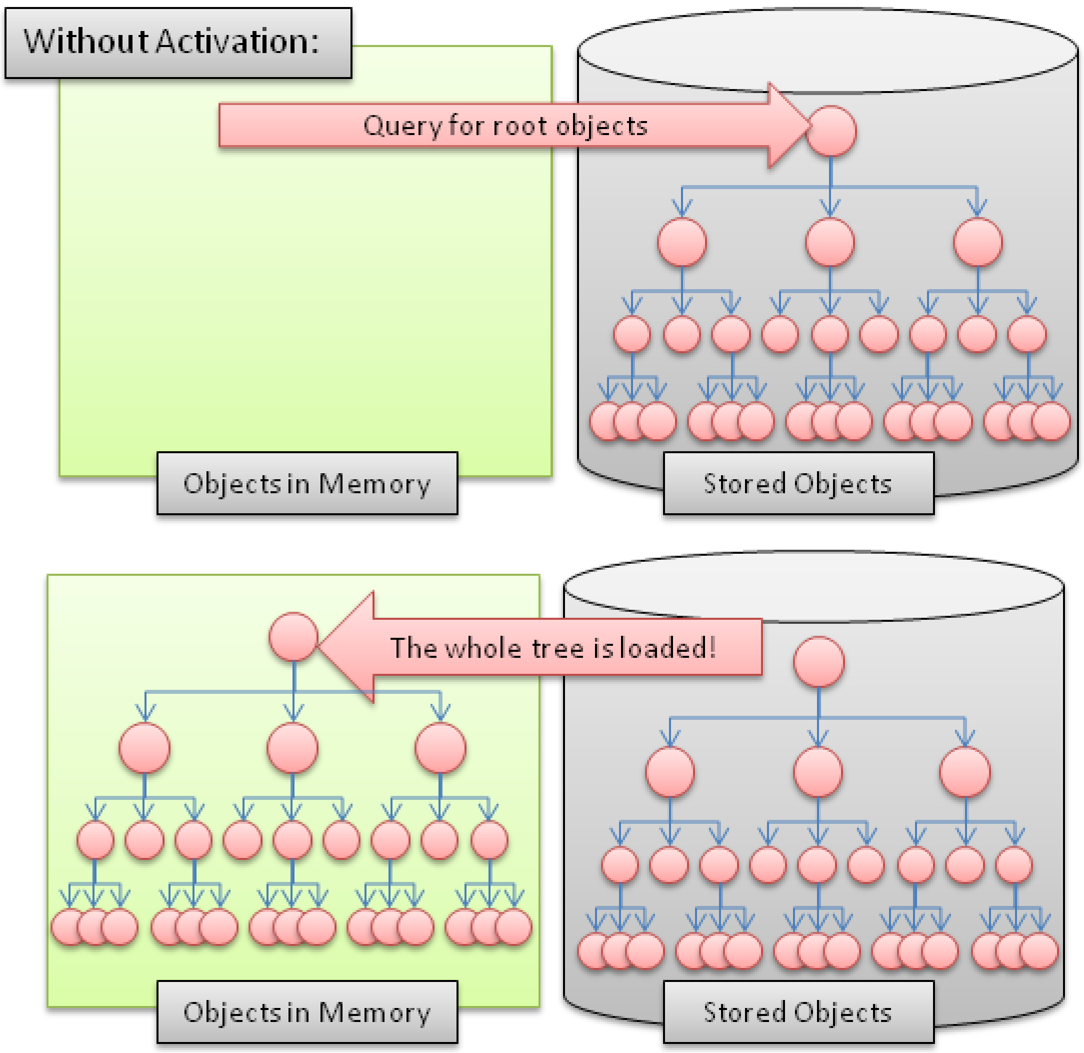
IdentityConcepts.cs: With purge you can remove objects from the reference cache

```
Dim theCar As Car = container.Query(Of Car)()(0)
```

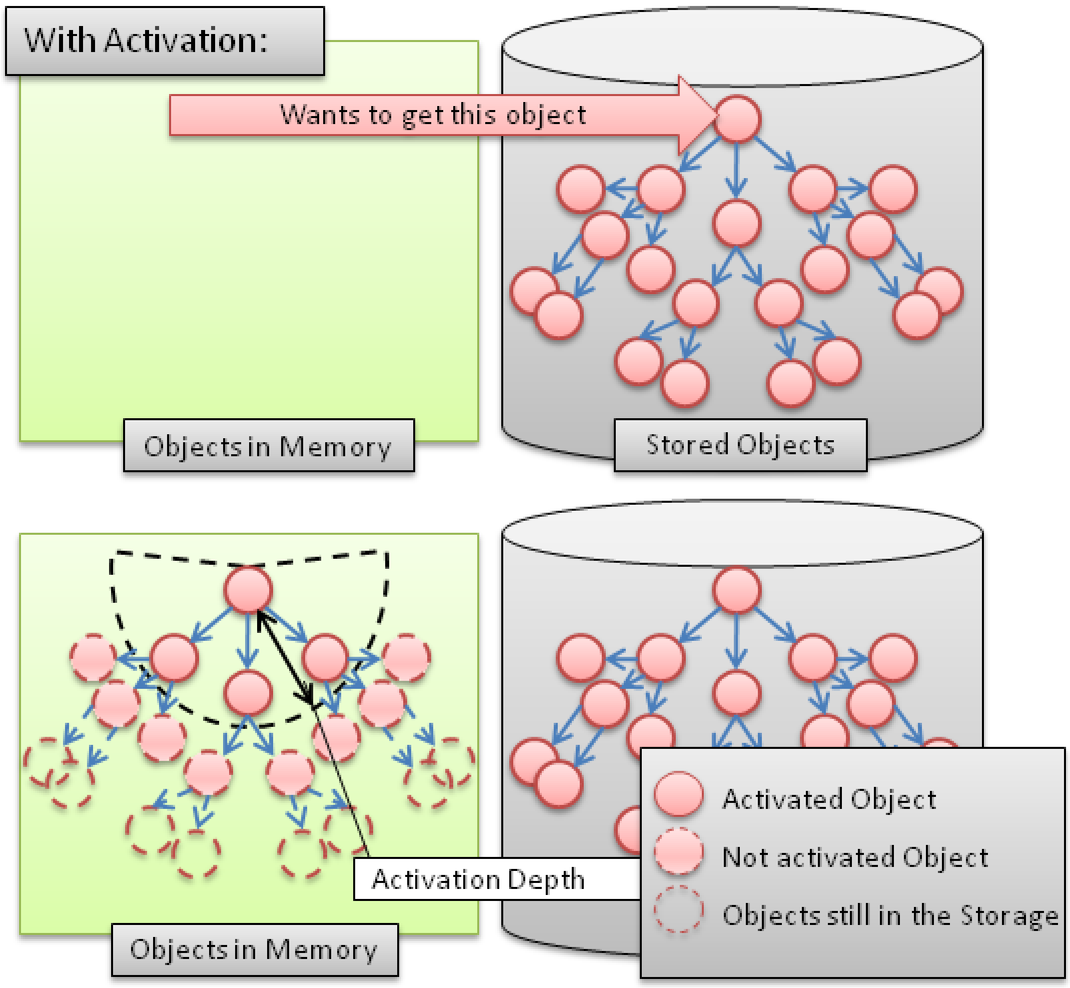
IdentityConcepts.vb: With purge you can remove objects from the reference cache

Activation Concept

Activation is a db4o mechanism which controls object instantiation. Why is it necessary? Let's look at an example of a stored tree structure. There's one root object, which has a bunch nodes. Each node has again a few subnodes and so on and so forth. What happens when you run a query and retrieve the root object? All the sub-objects will have to be created in the memory! If the tree is very large, this will fill up your memory.



Luckily db4o does not behave like this.When a query retrieves objects, their fields are loaded into memory (activated in db4o terms) only to a certain 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 or to default values. So db4o doesn't load the whole object graph from the database. Instead, db4o loads only parts of the object graph you're interested in.



Activation occurs in the following cases:

1. When you iterate over query results.
2. Object is activated explicitly with the object containers activate method.
3. Collections members are activated automatically, when the collection is activated, using at least depth 1 for lists and depth 2 for maps.

For a concrete example of the activation process: See "Activation In Action" on page 48

If you want to automate the activation process: See "Transparent Activation" on page 51

Activation In Action

Let's see db4o's activation in action. To see activation you need a deep object-graph. To keep this example simple we create a person-class with a mother-field. This allows us to simply create a very deep object graph.

First the Person class:

```
internal class Person
{
    private Person mother;
    private string name;

    public Person(string name)
    {
        this.name = name;
    }

    public Person(Person mother, string name)
    {
        this.mother = mother;
        this.name = name;
    }

    public Person Mother
    {
        get { return mother; }
    }

    public string Name
    {
        get { return name; }
    }
}
```

Person.cs: Person with a reference to the mother

```
Friend Class Person
    Private m_mother As Person
    Private m_name As String

    Public Sub New(ByVal name As String)
        m_mother = m_mother
        Me.m_name = name
    End Sub

    Public Sub New(ByVal mother As Person, ByVal name As String)
        Me.m_mother = mother
        Me.m_name = name
    End Sub

    Public ReadOnly Property Mother() As Person
        Get
            Return m_mother
        End Get
    End Property

    Public ReadOnly Property Name() As String
        Get
            Return m_name
        End Get
    End Property
End Class
```

Person.vb: Person with a reference to the mother

After that we store a deep hierarchy of persons. Let's say we store a hierarchy of seven people. Then we query for it and traverse this object graph. When we hit the sixth person, that object won't be activated, because it's outside the activation depth. That object will have all fields set to null.

```
Person jodie = QueryForJodie(container);

Person julia = jodie.Mother.Mother.Mother.Mother.Mother;

// This will print null
// Because julia is not activated
// and therefore all fields are not set
Console.WriteLine(julia.Name);
// This will throw a NullPointerException.
// Because julia is not activated
// and therefore all fields are not set
string joannaName = julia.Mother.Name;
```

ActivationDepthPitfall.cs: Run into not activated objects

```
Dim jodie As Person = QueryForJodie(container)

Dim julia As Person = jodie.Mother.Mother.Mother.Mother.Mother

' This will print null
' Because julia is not activated
' and therefore all fields are not set
Console.WriteLine(julia.Name)
' This will throw a NullPointerException.
' Because julia is not activated
' and therefore all fields are not set
Dim joannaName As String = julia.Mother.Name
```

ActivationDepthPitfall.vb: Run into not activated objects

Use Explicit Activation

When we traverse deep object graphs, we know that we might run into not activated objects. Therefore you can activate objects explicitly.

```
Person julia = jodie.Mother.Mother.Mother.Mother.Mother;
container.Activate(julia,5);

Console.WriteLine(julia.Name);
string joannaName = julia.Mother.Name;
Console.WriteLine(joannaName);
```

ActivationDepthPitfall.cs: Fix with explicit activation

```
Dim julia As Person = jodie.Mother.Mother.Mother.Mother.Mother
container.Activate(julia, 5)

Console.WriteLine(julia.Name)
Dim joannaName As String = julia.Mother.Name
```

ActivationDepthPitfall.vb: Fix with explicit activation

Configure Activation

You can configure db4o to increase the activation depth. You can increase it globally or for certain classes. Or you can cascade activate certain objects.

However remember that activation is there to improve the performance and save memory. If you set the activation depth to high it will hurt the performance.

Transparent Activation

If you have a very complex model or don't want to deal with all the activation hassle then transparent activation is the best option. Transparent activation will manage the activation for you. See "Transparent Activation" on page 51

Transparent Activation

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** [1]) 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.

First take a look at a simple example: See "Transparent Activation Example" on page 51

For more information how transparent activation works: See "TA Implementation" on page 55

You probably want to automate the process of implementing the required interfaces manually: See "TA Enhanced Example" on page 56

Its also possible to mix transparent activation aware object and other objects. See "Object Types In TA" on page 65

Collections can be also integrated into the transparent activation framework. See "TA Aware Collections" on page 61

Public fields can also integrated with transparent activation to a certain limit: See "TA For Public Fields" on page 66

For diagnosis, take a look here: See "TA Diagnostics" on page 66

Also take a look at the pitfalls: See "Transparent Activation Pitfalls" on page 67

Transparent Activation Example

In order to support Transparent Activation, the objects which are stored in the database need to implement the IActivatable-interface.

An object which implements the IActivatable-interface is responsible for activating itself. For this purpose the class needs a field to keep its activator. This field is only used by the transparent activation framework. Therefore it's marked as transient, to avoid that it's stored in the database.

```
public class Person : IActivatable
{
    [NonSerialized] private IActivator activator;
```

Person.cs: Implement the required activatable interface and add activator

[1] Transparent Activation

```
Public Class Person
    Implements IActivatable
    <Transient()> _
    Private m_activator As IActivator
```

Person.vb: Implement the required activatable interface and add activator

Then implement the two methods of the IActivatable-interface. The bind-method binds an activator to the object. It's called by the transparent activation framework. The activate-method needs to be called before any read or write operation on the object. Since these two methods are always the same, you can move the implementation to a common super class or to a static utility class.

```
public void Bind(IActivator activator)
{
    if (this.activator == activator)
    {
        return;
    }
    if (activator != null && null != this.activator)
    {
        throw new InvalidOperationException("Object can only be bound to one activator");
    }
    this.activator = activator;
}

public void Activate(ActivationPurpose activationPurpose)
{
    if (null != activator)
    {
        activator.Activate(activationPurpose);
    }
}
```

Person.cs: Implement the activatable interface methods

```
Public Sub Bind(ByVal activator As IActivator) _
        Implements IActivatable.Bind
    If m_activator Is activator Then
        Exit Sub
    End If
    If activator IsNot Nothing AndAlso m_activator IsNot Nothing Then
        Throw New InvalidOperationException("Object can only be bound to one activator")
    End If
    m_activator = activator
End Sub

Public Sub Activate(ByVal activationPurpose As ActivationPurpose) _
        Implements IActivatable.Activate
    If m_activator IsNot Nothing Then
        m_activator.Activate(activationPurpose)
    End If
End Sub
```

Person.vb: Implement the activatable interface methods

Now the important part. Every time a field of the class is accessed you need to call the activate-method with the purpose. For example in every property or other method. Probably the best way is to use only property even within the class to access fields. And the property ensures that the activate-method is called.

```
public string Name
{
    get
    {
        Activate(ActivationPurpose.Read);
        return name;
    }
    set
    {
        Activate(ActivationPurpose.Write);
        name = value;
    }
}


public override string ToString()
{
    // use the getter/setter withing the class,
    // to ensure the activate-method is called
    return Name;
}
```

Person.cs: Call the activate method on every field access

```
Public Property Name() As String
    Get
        Activate(ActivationPurpose.Read)
        Return m_name
    End Get
    Set(ByVal value As String)
        Activate(ActivationPurpose.Write)
        m_name = value
    End Set
End Property


Public Overloads Overrides Function ToString() As String
    ' use the getter/setter withing the class,
    ' to ensure the activate-method is called
    Return Name
End Function
```

Person.vb: Call the activate method on every field access

Implementing the IActivatable-interface manually for every class is repetitive and error prone. That's why this process can be automated. See "TA Enhanced Example" on page 56

The last step is to enable transparent persistence via configuration.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.Add(new TransparentActivationSupport());
IObjectContainer container = Db4oEmbedded.OpenFile(configuration, DatabaseFileName);
```

TransparentActivationExamples.cs: Activate transparent activation

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.Add(New TransparentActivationSupport())
Dim container As IObjectContainer = Db4oEmbedded.OpenFile(configuration, DatabaseFileName)
```

TransparentActivationExamples.vb: Activate transparent activation

Now transparent persistence is enabled. Now you can navigate the object-graph as deep as you want. The transparent activation will load the objects from the database as you need them.

```
using (IObjectContainer container = OpenDatabaseTA())
{
    Person person = Person.PersonWithHistory();
    container.Store(person);
}
using (IObjectContainer container = OpenDatabaseTA())
{
    Person person = QueryByName(container, "Joanna the 10");
    Person beginOfDynasty = person.Mother;

    // With transparent activation enabled, you can navigate deeply
    // nested object graphs. db4o will ensure that the objects
    // are loaded from the database.
    while (null != beginOfDynasty.Mother)
    {
        beginOfDynasty = beginOfDynasty.Mother;
    }
    Console.WriteLine(beginOfDynasty.Name);

    // Updating a object doesn't requires no store call.
    // Just change the objects and the call commit.
    beginOfDynasty.Name = "New Name";
    container.Commit();
}
```

TransparentActivationExamples.cs: Transparent activation in action

```
Using container As IObjectContainer = OpenDatabaseTA()
    Dim joanna As Person = Person.PersonWithHistory()
    container.Store(joanna)
End Using
Using container As IObjectContainer = OpenDatabaseTA()
    Dim joanna As Person = QueryByName(container, "Joanna the 10")
    Dim beginOfDynasty As Person = joanna.Mother

    ' With transparent activation enabled, you can navigate deeply
    ' nested object graphs. db4o will ensure that the objects
    ' are loaded from the database.
    While beginOfDynasty.Mother IsNot Nothing
        beginOfDynasty = beginOfDynasty.Mother
    End While
    Console.WriteLine(beginOfDynasty.Name)

    ' Updating a object doesn't requires no store call.
    ' Just change the objects and the call commit.
    beginOfDynasty.Name = "New Name"
    container.Commit()
End Using
```

TransparentActivationExamples.vb: Transparent activation in action

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. See "Transparent Activation Example" on page 51
- To instruct db4o to operate in Transparent Activation mode, call: configuration.common().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**[1] 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.

[1]Transparent Activation

- TA Enhanced Example
- Detailed Example

TA Enhanced Example

You can inject **TA**[1] 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 .NET

TA enhancement for .NET can be done at build or post-build time. In both cases it requires Db4oTool project distribution.

When build-time option is used Db4oEnhanceMSBuildTask is added to the build file (*.csproj or *.vbproj in Visual Studio), this requires Db4oTool.MSBuild.dll library.

In post-build enhancement case, Db4oTool.exe command line utility is used to enhanced built assemblies.

- TA Enhancement With Db4oTool
- Build Time Enhancement

TA Enhancement With Db4oTool

TA[2] 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:

```
SensorPanelTA.cs
/**//* Copyright (C) 2004 - 2007 Versant Inc. http://www.db4o.com */
using Db4objects.Db4o;
using Db4objects.Db4o.Activation;
using Db4objects.Db4o.TA;

namespace Db4ojects.Db4odoc.TAExamples
 {
    public class SensorPanelTA /**//*must implement Activatable for TA*/
     {
        private object _sensor;

        private SensorPanelTA _next;


        public SensorPanelTA()
         {
```

[1]Transparent Activation
[2]Transparent Activation

```
            // default constructor for instantiation
        }

        public SensorPanelTA(int value)
         {
            _sensor = value;
        }


        public SensorPanelTA Next
         {
            get
             {
                return _next;
            }
        }

        public object Sensor
         {
            get
             {
                return _sensor;
            }
        }

        public SensorPanelTA CreateList(int length)
         {
            return CreateList(length, 1);
        }

        public SensorPanelTA CreateList(int length, int first)
         {
            int val = first;
            SensorPanelTA root = NewElement(first);
            SensorPanelTA list = root;
            while (--length > 0)
             {
                list._next = NewElement(++val);
                list = list.Next;
            }
            return root;
        }

        protected SensorPanelTA NewElement(int value)
         {
            return new SensorPanelTA(value);
        }

        public override string ToString()
         {
            return "Sensor #" + Sensor;
        }

    }

}
```

```
SensorPanelTA.vb
' Copyright (C) 2004 - 2007 Versant Inc. http://www.db4o.com
```

```
Imports Db4objects.Db4o
Imports Db4objects.Db4o.Activation
Imports Db4objects.Db4o.TA

Namespace Db4ojects.Db4odoc.TAExamples

    Public Class SensorPanelTA

        Private _sensor As Object
        Private _next As SensorPanelTA


        Public Sub New()
        End Sub

        Public Sub New(ByVal value As Integer)
            _sensor = value
        End Sub


        Public ReadOnly Property NextSensor() As SensorPanelTA
            Get
                Return _next
            End Get
        End Property

        Public ReadOnly Property Sensor() As Object
            Get
                Return _sensor
            End Get
        End Property

        Public Function CreateList(ByVal length As Integer) As SensorPanelTA
            Return CreateList(length, 1)
        End Function

        Public Function _
CreateList(ByVal length As Integer, ByVal first As Integer) _
As SensorPanelTA
            Dim val As Integer = first
            Dim root As SensorPanelTA = NewElement(first)
            Dim list As SensorPanelTA = root
            While System.Threading.Interlocked.Decrement(length) > 0
                list._next = NewElement(System.Threading.Interlocked.Increment(val))
                list = list.NextSensor
            End While
            Return root
        End Function

        Protected Function NewElement(ByVal value As Integer) As SensorPanelTA
            Return New SensorPanelTA(value)
        End Function

        Public Overloads Overrides Function ToString() As String
            If Sensor Is Nothing Then
                Return ""
            Else
                Return "Sensor #" + Sensor.ToString()
            End If
        End Function
    End Class
End Namespace
```

In your code you will need to add Transparent Activation support to the configuration:

```
TAExample.cs: ConfigureTA
private static IConfiguration ConfigureTA()
         {
            IConfiguration configuration = Db4oFactory.NewConfiguration();
            // set normal activation to 0
            configuration.ActivationDepth(0);
            // add TA support
            configuration.Add(new TransparentActivationSupport());
            return configuration;
        }
```

```
TAExample.cs: StoreSensorPanel
private static void StoreSensorPanel()
         {
            File.Delete(Db4oFileName);
            IObjectContainer container = Database(Db4oFactory.NewConfiguration());
            if (container != null)
             {
                try
                 {
                    // create a linked list with length 10
                    SensorPanelTA list = new SensorPanelTA().CreateList(10);
                    container.Store(list);
                }
                finally
                 {
                    CloseDatabase();
                }
            }
        }
```

```
TAExample.cs: TestActivation
private static void TestActivation()
         {
            StoreSensorPanel();
            IConfiguration configuration = ConfigureTA();

            IObjectContainer container = Database(configuration);
            if (container != null)
             {
                try
                 {
                    System.Console.WriteLine("Zero activation depth");
                    IObjectSet result = container.QueryByExample(new SensorPanelTA(1));
                    ListResult(result);
                    if (result.Size() > 0)
                     {
                        SensorPanelTA sensor = (SensorPanelTA)result[0];
                        // the object is a linked list, so each call to next()
                        // will need to activate a new object
                        SensorPanelTA next = sensor.Next;
                        while (next != null)
                         {
                            System.Console.WriteLine(next);
                            next = next.Next;
                        }
                    }
                }
                finally
                 {
                    CloseDatabase();
                }
            }
        }
```

```
TAExample.vb: ConfigureTA
Private Shared Function  ConfigureTA() As IConfiguration
            Dim configuration As IConfiguration = Db4oFactory.NewConfiguration
            ' set normal activation to 0
            configuration.ActivationDepth(0)
            ' add TA support
            configuration.Add(New TransparentActivationSupport)
            Return configuration
        End Function
```

```
TAExample.vb: StoreSensorPanel
Private Shared Sub StoreSensorPanel()
            File.Delete(Db4oFileName)
            Dim container As IObjectContainer = Database(Db4oFactory.NewConfiguration)
            If Not (container Is Nothing) Then
                Try
                    ' create a linked list with length 10
                    Dim list As SensorPanelTA = (New SensorPanelTA).CreateList(10)
                    container.Store(list)
                Finally
                    CloseDatabase()
                End Try
            End If
        End Sub
```

```
TAExample.vb: TestActivation
Private Shared Sub TestActivation()
            StoreSensorPanel()
            Dim configuration As IConfiguration = ConfigureTA()
            Dim container As IObjectContainer = Database(configuration)
            If Not (container Is Nothing) Then
                Try
                    System.Console.WriteLine("Zero activation depth")
                    Dim result As IObjectSet = container.QueryByExample(New SensorPanelTA(1))
                    ListResult(result)
                    If result.Size > 0 Then
                        Dim sensor As SensorPanelTA = CType(result(0), SensorPanelTA)
                        ' the object is a linked list, so each call to next()
                        ' will need to activate a new object
                        Dim nextSensor As SensorPanelTA = sensor.NextSensor
                        While Not (nextSensor Is Nothing)
                            System.Console.WriteLine(nextSensor.ToString())
                            nextSensor = nextSensor.NextSensor
                        End While
                    End If
                Finally
                    CloseDatabase()
                End Try
            End If
        End Sub
```

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.

TA Aware Collections

In order to support properly, a class need to implement the IActivatable-interface. For your domain-classes this is quite easy to archive. But what about the .NET-collections? Wouldn't it be nice when the collections also work together the transparent activation framework?

db4o brings special, transparent activation aware collections with it. This collections load the content only when the collections is actually used.

These collections are currently implemented:

- ActivatableList<T>: An activatable version of the List<T>-class
- ActivatableDictionary<T>: An activatable version of the Dictionary<T>

It's recommended to use the collection-interfaces wherever possible instead of the concrete classes. This avoids unnecessary direct dependencies on the implementations and makes it easy to exchange the implementations.

You can use the db4o-collection directly in your code. For example in the case you implement the transparent activation support manually. Take a looks these tips: See "Using TA Collections Directly" on page 61

The enhancement tools can automatically replace the .NET-collections with the db4o-equivalent. However there are few rules and limitations. See "Enhance Collections" on page 62

Using TA Collections Directly

You can use the transparent activation aware collections directly in your code. They behavior is the same as the .NET-collections. Here are a few tips:

- Prefer the collection-interfaces over the concrete classes in your field, parameter and return-type declarations. For example declare a field as a IList<T> instead of a List<T>
- Maybe it useful to create a collection factory in your code. Then you have only one play in your code which decides which classes are used. So it easy to replace the implementations.

An example:

```
public class Team : ActivatableBase
{
    private readonly IList<Pilot> pilots = new ActivatableList<Pilot>();

    public void Add(Pilot pilot)
    {
        Activate(ActivationPurpose.Write);
        pilots.Add(pilot);
    }

    public ICollection<Pilot> Pilots
    {
        get
        {
            Activate(ActivationPurpose.Read);
            return pilots;
        }
    }
}
```

Team.cs: Using the activation aware collections

```
Public Class Team
    Inherits ActivatableBase
    Private ReadOnly m_pilots As IList(Of Pilot) = New ActivatableList(Of Pilot)()

    Public Sub Add(ByVal pilot As Pilot)
        Activate(ActivationPurpose.Write)
        m_pilots.Add(pilot)
    End Sub

    Public ReadOnly Property Pilots() As ICollection(Of Pilot)
        Get
            Activate(ActivationPurpose.Read)
            Return m_pilots
        End Get
    End Property
End Class
```

Team.vb: Using the activation aware collections

Currently these collections are available:

- ActivatableList<T>: An activatable version of the List<T>-class
- ActivatableDictionary<T>: An activatable version of the Dictionary<T>

Enhance Collections

You can use the normal .NET-collections in your code and then replace the implementations with the enhancement-tools. See "TA Enhanced Example" on page 56

The enhancement tools will search for instantiations of collections and replace it with an appropriate transparent activation aware collection.

However this has some implications. The original collection-classes are all sealed and not designed for extension. This means that a complete different implementation is used and has consequences.

Can Be Enhanced: When Using Interfaces In Declaration

The best case is when collection-interface is used, like IList instead of the concrete class. For example a field-declaration like this:

```
public class CanBeEnhanced
{
    private IList<string> _names = new List<string>();

    public bool ContainsName(string item)
    {
        return _names.Contains(item);
    }

    public void AddName(string item)
    {
        _names.Add(item);
    }
}
```

EnhancementLimitations.cs: Can be enhanced by the db4o-tools

```
Public Class CanBeEnhanced
    Private _names As IList(Of String) = New List(Of String)()

    Public Function ContainsName(ByVal item As String) As Boolean
        Return _names.Contains(item)
    End Function

    Public Sub AddName(ByVal item As String)
        _names.Add(item)
    End Sub
End Class
```

EnhancementLimitations.vb: Can be enhanced by the db4o-tools

Are correctly translated by the enhancement tools to:

```
public class CanBeEnhanced
{
    private IList<string> _names = new ActivatableList<string>();

    public bool ContainsName(string item)
    {
        return _names.Contains(item);
    }

    public void AddName(string item)
    {
        _names.Add(item);
    }
}
```

EnhancementLimitations.cs: Is enhanced to

```
Public Class CanBeEnhanced
    Private _names As IList(Of String) = New ActivatableList(Of String)()

    Public Function ContainsName(ByVal item As String) As Boolean
        Return _names.Contains(item)
    End Function

    Public Sub AddName(ByVal item As String)
        _names.Add(item)
    End Sub
End Class
```

EnhancementLimitations.vb: Is enhanced to

Cannot Be Enhanced: When Using Concrete Class In Declaration

When you use the concrete types in field declarations, the enhancer-tools will produce a warning and doesn't change the implementation. The example below cannot be enhanced, because it uses the concrete type.

```
public class CannotBeEnhanced
{
    // cannot be enhanced, because it uses the concrete type
    private List<string> _names = new List<string>();

    public bool ContainsName(string item)
    {
        return _names.Contains(item);
    }

    public void AddName(string item)
    {
        _names.Add(item);
    }
}
```

EnhancementLimitations.cs: Cannot be enhanced by the db4o-tools

```
Public Class CannotBeEnhanced
    ' cannot be enhanced, because it uses the concrete type
    Private _names As New List(Of String)()

    Public Function ContainsName(ByVal item As String) As Boolean
        Return _names.Contains(item)
    End Function

    Public Sub AddName(ByVal item As String)
        _names.Add(item)
    End Sub
End Class
```

EnhancementLimitations.vb: Cannot be enhanced by the db4o-tools

Casts are dangerous

The enhancement tools replace the implementation of collections. When you code has an assumptions about the concrete types and tries to cast, it may fail. In general try to avoid casting to concrete types and use interfaces instead.

Object Types In TA

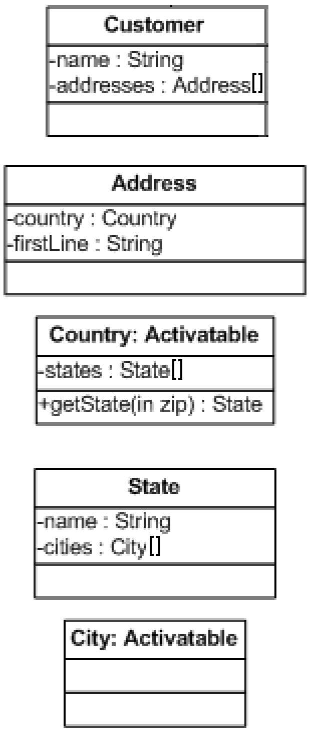
When working in **TA**[1] enabled environment you must remember that db4o treats Activatable (TA Aware) and non Activatable (other) types differently.

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:

c#:

```
Customer c = container.QueryByExample(typeof(Customer)).Next();
```

VB:

```
Dim c as Customer = container.QueryByExample(GetType(Customer)).Next();
```

At this point the following paths should be already activated (Customer is not Activatable):

```
c.namec.addresses.addresses[N].firstLine
```

`c.addresses[N].country` - available but not activated (Activatable type).

[1]Transparent Activation

Country.getState would cause the Country object to be activated

c#:

```
State state = c.Address[0].Country.GetState(someZipCode);
```

VB:

```
Dim                 state                 As                 State                 =
c.Address(0).Country.GetState(someZipCode);
```

At this point the following paths become activated

```
c.addresses[0].country.states                             .addresses[0]
.country.states[N].name
```

```
c.addresses[0].country.states[N].city                         .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.

TA Diagnostics

You can use Diagnostics to get runtime information about classes with and without **TA**[1] support. Add a call to the following method in the `configureTA()` method and run the example from the previous topic:

```
TAExample.cs: ActivateDiagnostics
private static void ActivateDiagnostics(IConfiguration configuration)
         {
            // Add diagnostic listener that will show all the classes that are not
            // TA aware.
            configuration.Diagnostic().AddListener(new TADiagnostics());
        }
```

```
TAExample.vb: ActivateDiagnostics
Private Shared Sub ActivateDiagnostics(ByVal configuration As IConfiguration)
            ' Add diagnostic listener that will show all the classes that are not
            ' TA aware.
            configuration.Diagnostic.AddListener(New TADiagnostics)
        End Sub
```

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).

TA For Public Fields

Accessing public fields through **TA**[2] seems to be natural - if properties and getters are activated transparently, so should be public fields. However, if we look at TA Implementation, we can see that we need to embed Activate call in the method accessing the field. It is easy in the case of getters and Properties

[1]Transparent Activation
[2]Transparent Activation

as the method is the part of the persistent class. However, in the case of public fields, the access method can exist anywhere in the code. Effectively, TA enhancer has to browse through all the classes and find all references to public fields and instrument them as necessary. Well, this is still feasible.

However, there is a certain catch: if persistent classes are in a separate library/assembly from the main code, you have to make sure that BOTH persistent classes library/assembly and the accessing library/assembly are both instrumented. This is necessary to make sure that TA code is injected both in the persistent class and in the class accessing persistent class public field.

Transparent Activation Pitfalls

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 Activate Call Before Field Access

Before accessing any field you need to call the activate-method. This is true for all property and also for other methods like the to string method or the hash code method. The best strategy is to call the activate-method in the property and then access the field through those even in the class itself.

Or use the enhancement-tools to avoid this issue complete.

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:

.NET:
```
myObject.Bind(null);
```

For more information see an example.

Debugging Instrumented Classes

Debugging instrumented classes may not work 100% correct. Make sure you use the debug-flag for the db4otool

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.

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.cs: Bind
/**//*Bind the class to the specified object container, create the activator*/
        public void Bind(IActivator activator)
         {
            if (_activator == activator)
             {
                return;
            }
            if (activator != null && null != _activator)
             {
                throw new System.InvalidOperationException();
            }
            _activator = activator;
        }
```

```
SensorPanelTA.vb: Bind
' Bind the class to the specified object container, create the activator
        Public Sub Bind(ByVal activator As IActivator) Implements IActivatable.Bind
            If _activator Is activator Then
                Return
            End If
            If Not (activator Is Nothing Or _activator Is Nothing) Then
                Throw New System.InvalidOperationException()
            End If
            _activator = activator
        End Sub
```

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):

```
TAExample.cs: TestSwitchDatabases
private static void TestSwitchDatabases()
         {
            StoreSensorPanel();

            IObjectContainer firstDb = Db4oFactory.OpenFile(ConfigureTA(), FirstDbName);
            IObjectContainer secondDb = Db4oFactory.OpenFile(ConfigureTA(), SecondDbName);
            try
             {
                IObjectSet result = firstDb.QueryByExample(new SensorPanelTA(1));
                if (result.Count > 0)
                 {
                    SensorPanelTA sensor = (SensorPanelTA)result[0];
                    firstDb.Close();
                    // Migrating an object from the first database
                    // into a second database
                    secondDb.Store(sensor);
                }
            }
            finally
```

```
              {
                 firstDb.Close();
                 secondDb.Close();
             }
         }
```

```
TAExample.vb: TestSwitchDatabases
Private Shared Sub TestSwitchDatabases()
            StoreSensorPanel()

            Dim firstDb As IObjectContainer = _
Db4oFactory.OpenFile(ConfigureTA(), FirstDbName)
            Dim secondDb As IObjectContainer = _
Db4oFactory.OpenFile(ConfigureTA(), SecondDbName)
            Try
                Dim result As IObjectSet = _
firstDb.QueryByExample(New SensorPanelTA(1))
                If result.Count > 0 Then
                    Dim sensor As SensorPanelTA = _
DirectCast(result(0), SensorPanelTA)
                    firstDb.Close()
                    ' Migrating an object from the first database
                    ' into a second database
                    secondDb.Store(sensor)
                End If
            Finally
                firstDb.Close()
                secondDb.Close()
            End Try
        End Sub
```

The solution to this problem is simple: activator should be unbound from the object:

c#:

```
sensor.Bind(null);
```

VB:

```
sensor.Bind(Nothing)
```

Note, that the object will quit being activatable for the first object container. The following example shows the described behaviour:

```
TAExample.cs: TestSwitchDatabasesFixed
private static void TestSwitchDatabasesFixed()
         {
            StoreSensorPanel();

            IObjectContainer firstDb = Db4oFactory.OpenFile(ConfigureTA(), FirstDbName);
            IObjectContainer secondDb = Db4oFactory.OpenFile(ConfigureTA(), SecondDbName);
            try
             {
                IObjectSet result = firstDb.QueryByExample(new SensorPanelTA(1));
                if (result.Count > 0)
                 {
                    SensorPanelTA sensor = (SensorPanelTA)result[0];
                    // Unbind the object from the first database
                    sensor.Bind(null);
                    // Migrating the object into the second database
                    secondDb.Store(sensor);
                    System.Console.WriteLine("Retrieving previous query results from "
                            + FirstDbName + ":");
                    SensorPanelTA next = sensor.Next;
                    while (next != null)
                     {
                        System.Console.WriteLine(next);
                        next = next.Next;
                    }

                    System.Console.WriteLine("Retrieving previous query results from "
                            + FirstDbName + " with manual activation:");
                    firstDb.Activate(sensor, Int32.MaxValue);
                    next = sensor.Next;
                    while (next != null)
                     {
                        System.Console.WriteLine(next);
                        next = next.Next;
                    }

                    System.Console.WriteLine("Retrieving sensorPanel from " + SecondDbName + ":");
                    result = secondDb.QueryByExample(new SensorPanelTA(1));
                    next = sensor.Next;
                    while (next != null)
                     {
                        System.Console.WriteLine(next);
                        next = next.Next;
                    }
                }
            }
            finally
             {
                firstDb.Close();
                secondDb.Close();
            }
        }
```

```
TAExample.vb: TestSwitchDatabasesFixed
Private Shared Sub TestSwitchDatabasesFixed()
            StoreSensorPanel()

            Dim firstDb As IObjectContainer = _
Db4oFactory.OpenFile(ConfigureTA(), FirstDbName)
            Dim secondDb As IObjectContainer = _
Db4oFactory.OpenFile(ConfigureTA(), SecondDbName)
            Try
                Dim result As IObjectSet = _
firstDb.QueryByExample(New SensorPanelTA(1))
                If result.Count > 0 Then
                    Dim sensor As SensorPanelTA = _
DirectCast(result(0), SensorPanelTA)
                    ' Unbind the object from the first database
                    sensor.Bind(Nothing)
                    ' Migrating the object into the second database
                    secondDb.Store(sensor)


                    System.Console.WriteLine( _
"Retrieving previous query results from " + FirstDbName + ":")
                    Dim [next] As SensorPanelTA = sensor.NextSensor
                    While [next] IsNot Nothing
                        System.Console.WriteLine([next])
                        [next] = [next].NextSensor
                    End While

                    System.Console.WriteLine( _
"Retrieving previous query results from " + FirstDbName + _
" with manual activation:")
                    firstDb.Activate(sensor, Int32.MaxValue)
                    [next] = sensor.NextSensor
                    While [next] IsNot Nothing
                        System.Console.WriteLine([next])
                        [next] = [next].NextSensor
                    End While

                    System.Console.WriteLine( _
"Retrieving sensorPanel from " + SecondDbName + ":")
                    result = secondDb.QueryByExample(New SensorPanelTA(1))
                    [next] = sensor.NextSensor
                    While [next] IsNot Nothing
                        System.Console.WriteLine([next])
                        [next] = [next].NextSensor
                    End While
                End If
            Finally
                firstDb.Close()
                secondDb.Close()
            End Try
        End Sub
```

Update Concept

Updating objects in db4o is as easy as storing them. You just call then store-method again to update a object. How does a update work? There are two main questions. First, how does db4o recognize a object so that it knows whenever it should update a object or store it as a new object? And what's the scope of updates? All all objects updated? Or just the objects you explicitly store?

Object Recognition

How does db4o know which object needs to be updated and which object has to be stored as new object? Well db4o uses the object-identity and looks up it if has loaded this object. If the object was loaded by db4o, it is an existing object and will be updated. Otherwise it has to be a new object and is stored as a new object.

Update Depth

When you update of a object, db4o only stores the changes to a certain depth. This update depth avoids that db4o needs to go through the whole object graph and find out which objects have changed.

By default this update depth is one. This means when you update a object, only the changes on that object are stored. Changes on other objects are not included. When you want to store changes of multiple objects you need either to increase the update-depth, store each object individually or use transparent persistence.

Take a look at this concrete example to see how the update-depth affects you're operations. See "Update Depth In Action" on page 72

Since collection are regular object in db4o the update depth also applies to collections. See "Updating Collections" on page 73

If you want to automate the update process of object: See "Transparent Persistence" on page 75

Update Depth In Action

Let's see db4o's update depth in action. We store a few cars with their pilots in the database. Then we update a car and its driver and store the car. The we reopen the database and check if everything was updated. To our surprise the car-name was updated, but the driver isn't. This is the direct result of db4o's update depth policy. It only updates object to a certain update-depth.

```
using (IObjectContainer container = Db4oEmbedded.OpenFile(DatabaseFile))
{
    Car car = QueryForCar(container);
    car.CarName = "New Mercedes";
    car.Driver.Name = "New Driver Name";

    // With the default-update depth of one, only the changes
    // on the car-object are stored, but not the changes on
    // the person
    container.Store(car);
}
using (IObjectContainer container = Db4oEmbedded.OpenFile(DatabaseFile))
{
    Car car = QueryForCar(container);
    Console.WriteLine("Car-Name:" + car.CarName);
    Console.WriteLine("Driver-Name:" + car.Driver.Name);
}
```

UpdateDepthPitfall.cs: Update depth limits what is store when updating objects

```
Using container As IObjectContainer = Db4oEmbedded.OpenFile(DatabaseFile)
    Dim car As Car = QueryForCar(container)
    car.CarName = "New Mercedes"
    car.Driver.Name = "New Driver Name"

    ' With the default-update depth of one, only the changes
    ' on the car-object are stored, but not the changes on
    ' the person
    container.Store(car)
End Using
Using container As IObjectContainer = Db4oEmbedded.OpenFile(DatabaseFile)
    Dim car As Car = QueryForCar(container)
    Console.WriteLine("Car-Name:" & car.CarName)
    Console.WriteLine("Driver-Name:" & car.Driver.Name)
End Using
```

UpdateDepthPitfall.vb: Update depth limits what is store when updating objects

Explicitly Store The Driver

One solution to this issue is to store updated object explicitly, except value objects. So in our case we would store the car and the pilot. This works fine for simple models. However as the model gets more complex this is probably not a feasible solution.

```
Car car = QueryForCar(container);
car.CarName = "New Mercedes";
car.Driver.Name = "New Driver Name";

// Explicitly store the driver to ensure that those changes are also in the database
container.Store(car);
container.Store(car.Driver);
```

UpdateDepthPitfall.cs: Explicitly store changes on the driver

```
Dim car As Car = QueryForCar(container)
car.CarName = "New Mercedes"
car.Driver.Name = "New Driver Name"

' Explicitly store the driver to ensure that those changes are also in the database
container.Store(car)
```

UpdateDepthPitfall.vb: Explicitly store changes on the driver

There also a variation of this. You can use the store method of the extended container and explicitly state the update depth for the store operation.

```
Car car = QueryForCar(container);
car.CarName = "New Mercedes";
car.Driver.Name = "New Driver Name";

// Explicitly state the update depth
container.Ext().Store(car, 2);
```

UpdateDepthPitfall.cs: Explicitly use the update depth

```
Dim car As Car = QueryForCar(container)
car.CarName = "New Mercedes"
car.Driver.Name = "New Driver Name"

' Explicitly state the update depth
```

UpdateDepthPitfall.vb: Explicitly use the update depth

Configure Update Depth

As alternative you can configure the update depth. You can increase it globally or for certain classes. It's also possible to enable cascading updates for certain classes or fields.

Transparent Persistence

You can get rid of all the update depth troubles by using transparent persistence. In this mode db4o tracks all changes and stores them. See "Transparent Persistence" on page 75

Updating Collections

From the db4o perspective collections behave like ordinary objects. This means that the update-depth also applies to collections. When you change a collection and store the object which contains it, the changes are not stored by default.

```
using (IObjectContainer container = Db4oEmbedded.OpenFile(DatabaseFile))
{
    Person jodie = QueryForJodie(container);
    jodie.Add(new Person("Jamie"));
    // Remember that a collection is also a regular object
    // so with the default-update depth of one, only the changes
    // on the person-object are stored, but not the changes on
    // the friend-list.
    container.Store(jodie);
}
using (IObjectContainer container = Db4oEmbedded.OpenFile(DatabaseFile))
{
    Person jodie = QueryForJodie(container);
    foreach (Person person in jodie.Friends)
    {
        // the added friend is gone, because the update-depth is to low
        Console.WriteLine("Friend=" + person.Name);
    }
}
```

UpdateDepthPitfall.cs: Update doesn't work on collection

```
Using container As IObjectContainer = Db4oEmbedded.OpenFile(DatabaseFile)
    Dim jodie As Person = QueryForJodie(container)
    jodie.Add(New Person("Jamie"))
    ' Remember that a collection is also a regular object
    ' so with the default-update depth of one, only the changes
    ' on the person-object are stored, but not the changes on
    ' the friend-list.
    container.Store(jodie)
End Using
Using container As IObjectContainer = Db4oEmbedded.OpenFile(DatabaseFile)
    Dim jodie As Person = QueryForJodie(container)
    For Each person As Person In jodie.Friends
        ' the added friend is gone, because the update-depth is to low
        Console.WriteLine("Friend=" & person.Name)
    Next
End Using
```

UpdateDepthPitfall.vb: Update doesn't work on collection

For collections the same rules and settings work as for regular objects. For example when you increase the update depth to two, you can store the parent object and the changes of the collection are persisted as well.

```
IEmbeddedConfiguration config = Db4oEmbedded.NewConfiguration();
config.Common.UpdateDepth = 2;
using (IObjectContainer container = Db4oEmbedded.OpenFile(DatabaseFile))
{
```

UpdateDepthPitfall.cs: A higher update depth fixes the issue

```
Dim config As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
config.Common.UpdateDepth = 2
Using container As IObjectContainer = Db4oEmbedded.OpenFile(DatabaseFile)
```

UpdateDepthPitfall.vb: A higher update depth fixes the issue

Transparent Persistence

One of db4o's goals is to make the database transparent to the application logic. Wouldn't it be nice to initially register an object with a single store()-call and then let the database manage all future object modifications? Transparent Persistence does exactly that.It keeps track of changes and stores all modified objects automatically when committing.

This has several benefits:

- Clean an refactorable code. The code doesn't depend on the right activation-depth.
- Performance-benefits. Only the objects which are needed are loaded into memory. And only modified objects are stored.
- No changes can be lost due to miss configured update-depth.

Transparent persistence takes care of activating the objects and storing changes. Take a look a simple example. See "Transparent Persistence Example" on page 75

In order to support transparent persistence the persisted object need to implement certain interfaces. To manually implementing this interface is time consuming. Therefore it can be automated with enhancement tools. See "TP Enhancement" on page 80

When using Transparent Persistence collections might need special attention. See "Transparent Persistence Collections" on page 99

For a deeper understanding of the mechanism: See "Transparent Persistence Implementation" on page 79

Transparent Persistence Example

In order to support **TP**[1], the persistent objects need to implement the IActivatable-interface.

An object which implements the IActivatable-interface is responsible for activating itself. For this purpose the class needs a field to keep its activator. This field is only used by the transparent activation framework. Therefore it's marked as transient, to avoid that it's stored in the database.

```
public class Person : IActivatable
{
    [NonSerialized] private IActivator activator;
```

Person.cs: Implement the required activatable interface and add activator

```
Public Class Person
    Implements IActivatable
    <Transient()> _
    Private m_activator As IActivator
```

Person.vb: Implement the required activatable interface and add activator

Then implement the two methods of the IActivatable-interface. The bind-method binds an activator to the object. It's called by the transparent activation framework. The activate-method needs to be called before any read or write operation on the object. Since these two methods are always the same, you can move the implementation to a common super class or to a static utility class.

[1]Transparent Persistence

```
public void Bind(IActivator activator)
{
    if (this.activator == activator)
    {
        return;
    }
    if (activator != null && null != this.activator)
    {
        throw new InvalidOperationException("Object can only be bound to one activator");
    }
    this.activator = activator;
}

public void Activate(ActivationPurpose activationPurpose)
{
    if (null != activator)
    {
        activator.Activate(activationPurpose);
    }
}
```

Person.cs: Implement the activatable interface methods

```
Public Sub Bind(ByVal activator As IActivator) _
        Implements IActivatable.Bind
    If m_activator Is activator Then
        Exit Sub
    End If
    If activator IsNot Nothing AndAlso m_activator IsNot Nothing Then
        Throw New InvalidOperationException("Object can only be bound to one activator")
    End If
    m_activator = activator
End Sub

Public Sub Activate(ByVal activationPurpose As ActivationPurpose) _
        Implements IActivatable.Activate
    If m_activator IsNot Nothing Then
        m_activator.Activate(activationPurpose)
    End If
End Sub
```

Person.vb: Implement the activatable interface methods

Now the important part. Every time a field of the class is accessed you need to call the activate-method with the purpose. For example in every property or other method. Probably the best way is to use only property even within the class to access fields. And the property ensures that the activate-method is called.

```
public string Name
{
    get
    {
        Activate(ActivationPurpose.Read);
        return name;
    }
    set
    {
        Activate(ActivationPurpose.Write);
        name = value;
    }
}


public override string ToString()
{
    // use the getter/setter withing the class,
    // to ensure the activate-method is called
    return Name;
}
```

Person.cs: Call the activate method on every field access

```
Public Property Name() As String
    Get
        Activate(ActivationPurpose.Read)
        Return m_name
    End Get
    Set(ByVal value As String)
        Activate(ActivationPurpose.Write)
        m_name = value
    End Set
End Property


Public Overloads Overrides Function ToString() As String
    ' use the getter/setter withing the class,
    ' to ensure the activate-method is called
    Return Name
End Function
```

Person.vb: Call the activate method on every field access

Implementing the IActivatable-interface manually for every class is repetitive and error prone. That's why this process can be automated. See "TP Enhancement" on page 80

The last step is to enable transparent persistence via configuration.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.Add(new TransparentPersistenceSupport());
IObjectContainer container = Db4oEmbedded.OpenFile(configuration, DatabaseFileName);
```

TransparentActivationExamples.cs: Activate transparent persistence

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.Add(New TransparentPersistenceSupport())
Dim container As IObjectContainer = Db4oEmbedded.OpenFile(configuration, DatabaseFileName)
```

TransparentActivationExamples.vb: Activate transparent persistence

Now transparent persistence is enabled. You have to store the object only once initially. After that, changes are automatically stored on the commit call.

```
using (IObjectContainer container = OpenDatabaseTP())
{
    Person person = Person.PersonWithHistory();
    container.Store(person);
}
using (IObjectContainer container = OpenDatabaseTP())
{
    Person person = QueryByName(container, "Joanna the 10");
    Person beginOfDynasty = person.Mother;

    // With transparent persistence enabled, you can navigate deeply
    // nested object graphs. db4o will ensure that the objects
    // are loaded from the database.
    while (null != beginOfDynasty.Mother)
    {
        beginOfDynasty = beginOfDynasty.Mother;
    }
    Console.WriteLine(beginOfDynasty.Name);

    // Updating a object doesn't requires no store call.
    // Just change the objects and the call commit.
    beginOfDynasty.Name = "New Name";
    container.Commit();
}
using (IObjectContainer container = OpenDatabaseTP())
{
    Person person = QueryByName(container, "New Name");
    // The changes are stored, due to transparent persistence
    Console.WriteLine(person.Name);
}
```

TransparentActivationExamples.cs: Transparent persistence in action

```
Using container As IObjectContainer = OpenDatabaseTP()
    Dim joanna As Person = Person.PersonWithHistory()
    container.Store(joanna)
End Using
Using container As IObjectContainer = OpenDatabaseTP()
    Dim joanna As Person = QueryByName(container, "Joanna the 10")
    Dim beginOfDynasty As Person = joanna.Mother

    ' With transparent persistence enabled, you can navigate deeply
    ' nested object graphs. db4o will ensure that the objects
    ' are loaded from the database.
    While beginOfDynasty.Mother IsNot Nothing
        beginOfDynasty = beginOfDynasty.Mother
    End While
    Console.WriteLine(beginOfDynasty.Name)

    ' Updating a object doesn't requires no store call.
    ' Just change the objects and the call commit.
    beginOfDynasty.Name = "New Name"
    container.Commit()
End Using
Using container As IObjectContainer = OpenDatabaseTP()
    Dim joanna As Person = QueryByName(container, "New Name")
    ' The changes are stored, due to transparent persistence
    Console.WriteLine(joanna.Name)
End Using
```

TransparentActivationExamples.vb: Transparent persistence in action

Transparent Persistence Implementation

The basic logic of Transparent Persistence (**TP**[1]) is the following:

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

  .NET: `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 objects 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, Transparent Persistent 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 the Transparent Activation documentation first.

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

[1]Transparent Persistence

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

   .NET: `configuration.Common.Add(new TransparentPersistenceSupport());`

2. Implement 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:

   .NET `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**[1]. So TA should be utilized 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.

TP Enhancement

Implementing the IActivatable-interface manually is time consuming, repetitive and error-prone. Therefore it's desirable to automate this process. There are different ways to automatically enhance classes to support transparent persistence.

- One possibility is to use a build-script which enhances the classes. For this a special MSBuild-task enhances the classes to support **TP**[2]. See "Build Time Enhancement Example" on page 178
- Another possibility is to use the a command-line tool to enhance the classes. See "Db4oTool" on page 180

SensorPanel

```
SensorPanel.Java
/**//* Copyright (C) 2004 - 2007 Versant Inc. http://www.db4o.com */

package com.db4odoc.tpbuildtime;

public class SensorPanel  {

  private Object _sensor;

  private SensorPanel _next;

  public SensorPanel()  {
    // default constructor for instantiation
  }
  // end SensorPanel

  public SensorPanel(int value)  {
    _sensor = new Integer(value);
  }
  // end SensorPanel

  public SensorPanel getNext()  {
    return _next;
```

[1]Transparent Activation
[2]Transparent Persistence

```
  }
  // end getNext

  public Object getSensor()  {
    return _sensor;
  }
  // end getSensor

  public void setSensor(Object sensor)  {
    _sensor = sensor;
  }
  // end setSensor

  public SensorPanel createList(int length)  {
    return createList(length, 1);
  }
  // end createList

  public SensorPanel createList(int length, int first)  {
    int val = first;
    SensorPanel root = newElement(first);
    SensorPanel list = root;
    while (--length > 0)  {
      list._next = newElement(++val);
      list = list._next;
    }
    return root;
  }
  // end createList

  protected SensorPanel newElement(int value)  {
    return new SensorPanel(value);
  }
  // end newElement

  public String toString()  {
    return "Sensor #" + getSensor();
  }
  // end toString
}
```

Transparent Persistence Pitfalls

Transparent Persistence is closely coupled with Transparent Activation, therefore the same pitfalls apply here. However there are some additional catches:

- Object Clone
- Rollback Strategies

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

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.cs: ModifyAndRollback
private static void ModifyAndRollback()
         {
            IObjectContainer container = Database(ConfigureTP());
            if (container != null)
             {
                try
                 {
                    // create a car
                    Car car = (Car)container.QueryByExample(new Car(null, null))
                            [0];
                    System.Console.WriteLine("Initial car: " + car + "("
                            + container.Ext().GetID(car) + ")");
                    car.Model = "Ferrari";
                    car.Pilot = new Pilot("Michael Schumacher", 123);
                    container.Rollback();
                    System.Console.WriteLine("Car after rollback: " + car + "("
                            + container.Ext().GetID(car) + ")");
                }
                finally
                 {
                    CloseDatabase();
                }
            }
        }
```

```
TPRollback.vb: ModifyAndRollback
Private Shared Sub ModifyAndRollback()
            Dim container As IObjectContainer = Database(ConfigureTP())
            If container IsNot Nothing Then
                Try
                    ' create a car
                    Dim car As Car = DirectCast(container. _
QueryByExample(New Car(Nothing, Nothing))(0), Car)
                    System.Console.WriteLine("Initial car: " + _
car.ToString() + "(" + container.Ext().GetID(car).ToString() + ")")
                    car.Model = "Ferrari"
                    car.Pilot = New Pilot("Michael Schumacher", 123)
                    container.Rollback()
                    System.Console.WriteLine("Car after rollback: " _
+ car.ToString() + "(" + _
container.Ext().GetID(car).ToString() + ")")
                Finally
                    CloseDatabase()
                End Try
            End If
        End Sub
```

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.cs: ModifyRollbackAndCheck
private static void ModifyRollbackAndCheck()
         {
            IObjectContainer container = Database(ConfigureTP());
            if (container != null)
             {
                try
                 {
                    // create a car
                    Car car = (Car)container.QueryByExample(new Car(null, null))
                            [0];
                    Pilot pilot = car.Pilot;
                    System.Console.WriteLine("Initial car: " + car + "("
                            + container.Ext().GetID(car) + ")");
                    System.Console.WriteLine("Initial pilot: " + pilot + "("
                            + container.Ext().GetID(pilot) + ")");
                    car.Model = "Ferrari";
                    car.ChangePilot("Michael Schumacher", 123);
                    container.Rollback();
                    container.Deactivate(car, Int32.MaxValue);
                    System.Console.WriteLine("Car after rollback: " + car + "("
                            + container.Ext().GetID(car) + ")");
                    System.Console.WriteLine("Pilot after rollback: " + pilot + "("
                            + container.Ext().GetID(pilot) + ")");
                }
                finally
                 {
                    CloseDatabase();
                }
            }
        }
```

```
TPRollback.vb: ModifyRollbackAndCheck
Private Shared Sub ModifyRollbackAndCheck()
            Dim container As IObjectContainer = Database(ConfigureTP())
            If container IsNot Nothing Then
                Try
                    ' create a car
                    Dim car As Car = DirectCast(container. _
QueryByExample(New Car(Nothing, Nothing))(0), Car)
                    Dim pilot As Pilot = car.Pilot
                    System.Console.WriteLine("Initial car: " _
+ car.ToString() + "(" + container.Ext().GetID(car).ToString() + ")")
                    System.Console.WriteLine("Initial pilot: " _
+ pilot.ToString() + "(" + _
container.Ext().GetID(pilot).ToString() + ")")
                    car.Model = "Ferrari"
                    car.ChangePilot("Michael Schumacher", 123)
                    container.Rollback()
                    container.Deactivate(car, Int32.MaxValue)
                    System.Console.WriteLine("Car after rollback: " _
+ car.ToString() + "(" + _
container.Ext().GetID(car).ToString() + ")")
                    System.Console.WriteLine("Pilot after rollback: " _
+ pilot.ToString() + "(" + _
container.Ext().GetID(pilot).ToString() + ")")
                Finally
                    CloseDatabase()
                End Try
            End If
        End Sub
```

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 max int 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.

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.cs: RollbackDeactivateStrategy
private class RollbackDeactivateStrategy : IRollbackStrategy
         {
            public void Rollback(IObjectContainer container, Object obj)
             {
                container.Ext().Deactivate(obj);
            }
        }
```

```
TPRollback.cs: ConfigureTPForRollback
private static IConfiguration ConfigureTPForRollback()
         {
            IConfiguration configuration = Db4oFactory.NewConfiguration();
            // add TP[1] support and rollback strategy
            configuration.Add(new TransparentPersistenceSupport(
                    new RollbackDeactivateStrategy()));
            return configuration;
        }
```

```
TPRollback.vb: RollbackDeactivateStrategy
Private Class RollbackDeactivateStrategy
            Implements IRollbackStrategy
            Public Sub  Rollback(ByVal container As IObjectContainer, _
ByVal obj As Object) _
            Implements IRollbackStrategy.Rollback
                container.Ext().Deactivate(obj)
            End Sub
        End Class
```

```
TPRollback.vb: ConfigureTPForRollback
Private Shared Function ConfigureTPForRollback() As IConfiguration
```

[1]Transparent Persistence

```
            Dim configuration As IConfiguration = Db4oFactory.NewConfiguration()
            ' add TP support and rollback strategy
            configuration.Add(New TransparentPersistenceSupport _
(New RollbackDeactivateStrategy()))
            Return configuration
        End Function
```

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.cs: ModifyWithRollbackStrategy
private static void ModifyWithRollbackStrategy()
         {
            IObjectContainer container = Database(ConfigureTPForRollback());
            if (container != null)
             {
                try
                 {
                    // create a car
                    Car car = (Car)container.QueryByExample(new Car(null, null))
                            [0];
                    Pilot pilot = car.Pilot;
                    System.Console.WriteLine("Initial car: " + car + "("
                            + container.Ext().GetID(car) + ")");
                    System.Console.WriteLine("Initial pilot: " + pilot + "("
                            + container.Ext().GetID(pilot) + ")");
                    car.Model = "Ferrari";
                    car.ChangePilot("Michael Schumacher", 123);
                    container.Rollback();
                    System.Console.WriteLine("Car after rollback: " + car + "("
                            + container.Ext().GetID(car) + ")");
                    System.Console.WriteLine("Pilot after rollback: " + pilot + "("
                            + container.Ext().GetID(pilot) + ")");
                }
                finally
                 {
                    CloseDatabase();
                }
            }
        }
```

```
TPRollback.vb: ModifyWithRollbackStrategy
Private Shared Sub ModifyWithRollbackStrategy()
            Dim container As IObjectContainer = Database(ConfigureTPForRollback())
            If container IsNot Nothing Then
                Try
                    ' create a car
                    Dim car As Car = DirectCast(container. _
QueryByExample(New Car(Nothing, Nothing))(0), Car)
                    Dim pilot As Pilot = car.Pilot
                    System.Console.WriteLine("Initial car: " + _
car.ToString() + "(" + container.Ext().GetID(car).ToString() + ")")
                    System.Console.WriteLine("Initial pilot: " + _
pilot.ToString() + "(" + _
container.Ext().GetID(pilot).ToString() + ")")
                    car.Model = "Ferrari"
                    car.ChangePilot("Michael Schumacher", 123)
                    container.Rollback()
                    System.Console.WriteLine("Car after rollback: " + _
car.ToString() + "(" + _
container.Ext().GetID(car).ToString() + ")")
                    System.Console.WriteLine("Pilot after rollback: " + _
pilot.ToString() + _
"(" + container.Ext().GetID(pilot).ToString() + ")")
                Finally
                    CloseDatabase()
                End Try
            End If
        End Sub
```

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).

Car

```
Car.cs
/** Copyright (C) 2004 - 2008 Versant Inc. http://www.db4o.com */
using System;
using Db4objects.Db4o.TA[1];
using Db4objects.Db4o.Activation;

namespace Db4objects.Db4odoc.TP[2].Rollback
 {
    public class Car : IActivatable
     {
        private string _model;
        private Pilot _pilot;
        /**//*activator registered for this class*/
        [System.NonSerialized]
        public IActivator _activator;


        public Car(string model, Pilot pilot)
         {
            _model = model;
            _pilot = pilot;
        }
        // end Car

        /**//*Bind the class to the specified object container, create the activator*/
        public void Bind(IActivator activator)
         {
            if (_activator == activator)
             {
                return;
            }
            if (activator != null && null != _activator)
             {
                throw new System.InvalidOperationException();
            }
            _activator = activator;
        }
        // end Bind
```

[1]Transparent Activation
[2]Transparent Persistence

```
        public void Activate(ActivationPurpose purpose)
         {
            if (_activator == null)
                return;
            _activator.Activate(purpose);
        }
        // end Activate

        public string Model
         {
            get
             {
                Activate(ActivationPurpose.Read);
                return _model;
            }
            set
             {
                Activate(ActivationPurpose.Write);
                _model = value;
            }
        }

        public Pilot Pilot
         {
            get
             {
                Activate(ActivationPurpose.Read);
                return _pilot;
            }
            set
             {
                Activate(ActivationPurpose.Write);
                _pilot = value;
            }
        }

        public void ChangePilot(String name, int id)
         {
            _pilot.Name = name;
            _pilot.Id.Change(id);
        }

        override public string ToString()
         {
            Activate(ActivationPurpose.Read);
            return string.Format("{0}[{1}]", _model, _pilot);
        }
        // end ToString
    }
}
```

```
Car.vb
' Copyright (C) 2004 - 2008 Versant Inc. http://www.db4o.com

Imports System
Imports Db4objects.Db4o
Imports Db4objects.Db4o.TA
Imports Db4objects.Db4o.Activation
```

```
Namespace Db4objects.Db4odoc.TP.Rollback
    Public Class Car
        Implements IActivatable
        Private _model As String
        Private _pilot As Pilot
        'activator registered for this class

        <Transient()> _
        Public _activator As IActivator


        Public Sub New(ByVal model As String, ByVal pilot As Pilot)
            _model = model
            _pilot = pilot
        End Sub
        ' end Car

        'Bind the class to the specified object container, create the activator

        Public Sub Bind(ByVal activator As IActivator) Implements IActivatable.Bind
            If _activator Is activator Then
                Return
            End If
            If activator IsNot Nothing AndAlso _activator IsNot Nothing Then
                Throw New System.InvalidOperationException()
            End If
            _activator = activator
        End Sub
        ' end Bind

        Public Sub Activate(ByVal purpose As ActivationPurpose) _
Implements IActivatable.Activate
            If _activator Is Nothing Then
                Return
            End If
            _activator.Activate(purpose)
        End Sub
        ' end Activate

        Public Property Model() As String
            Get
                Activate(ActivationPurpose.Read)
                Return _model
            End Get
            Set(ByVal value As String)
                Activate(ActivationPurpose.Write)
                _model = value
            End Set
        End Property

        Public Property Pilot() As Pilot
            Get
                Activate(ActivationPurpose.Read)
                Return _pilot
            End Get
            Set(ByVal value As Pilot)
                Activate(ActivationPurpose.Write)
                _pilot = value
            End Set
        End Property

        Public Sub ChangePilot(ByVal name As String, ByVal id As Integer)
            _pilot.Name = name
            _pilot.Id.Change(id)
        End Sub

        Public Overloads Overrides Function ToString() As String
            Activate(ActivationPurpose.Read)
            Return String.Format("{0}[{1}]", _model, _pilot)
        End Function
        ' end ToString
    End Class
End Namespace
```

Id

```
Id.cs
/**//* Copyright (C) 2004 - 2008 Versant Inc. http://www.db4o.com */
using Db4objects.Db4o;
using Db4objects.Db4o.Activation;
using Db4objects.Db4o.TA[1];

namespace Db4objects.Db4odoc.TP[2].Rollback
 {
    public class Id : IActivatable
     {
        int _number = 0;

        [System.NonSerialized]
        IActivator _activator;

        public Id(int number)
         {
            _number = number;
        }

        // Bind the class to an object container
        public void Bind(IActivator activator)
         {
            if (_activator == activator)
             {
                return;
            }
            if (activator != null && null != _activator)
             {
                throw new System.InvalidOperationException();
            }
            _activator = activator;
        }

        // activate the object fields
        public void Activate(ActivationPurpose purpose)
         {
```

[1]Transparent Activation
[2]Transparent Persistence

```
            if (_activator == null)
                return;
            _activator.Activate(purpose);
        }

        public void Change(int number)
         {
            Activate(ActivationPurpose.Write);
            _number = number;
        }

        public override string ToString()
         {
            Activate(ActivationPurpose.Read);
            return _number.ToString();
        }
    }

}
```

```
Id.vb
' Copyright (C) 2004 - 2008 Versant Inc. http://www.db4o.com

Imports Db4objects.Db4o
Imports Db4objects.Db4o.Activation
Imports Db4objects.Db4o.TA

Namespace Db4objects.Db4odoc.TP.Rollback
    Public Class Id
        Implements IActivatable
        Private _number As Integer = 0

        <Transient()> _
        Private _activator As IActivator

        Public Sub New(ByVal number As Integer)
            _number = number
        End Sub

        ' Bind the class to an object container
        Public Sub Bind(ByVal activator As IActivator) _
Implements IActivatable.Bind
            If _activator Is activator Then
                Return
            End If
            If activator IsNot Nothing AndAlso _activator _
IsNot Nothing Then
                Throw New System.InvalidOperationException()
            End If
            _activator = activator
        End Sub

        ' activate the object fields
        Public Sub Activate(ByVal purpose As ActivationPurpose) _
Implements IActivatable.Activate
            If _activator Is Nothing Then
                Return
            End If
            _activator.Activate(purpose)
        End Sub

        Public Sub Change(ByVal number As Integer)
            Activate(ActivationPurpose.Write)
            _number = number
        End Sub

        Public Overloads Overrides Function  ToString() As String
            Activate(ActivationPurpose.Read)
            Return _number.ToString()
        End Function
    End Class

End Namespace
```

Pilot

```
Pilot.cs
/**//* Copyright (C) 2004 - 2008 Versant Inc. http://www.db4o.com */
using Db4objects.Db4o;
using Db4objects.Db4o.Activation;
using Db4objects.Db4o.TA[1];

namespace Db4objects.Db4odoc.TP[2].Rollback
 {
    public class Pilot : IActivatable
     {
        private string _name;
        private Id _id;

        [System.NonSerialized]
        IActivator _activator;

        public Pilot(string name, int id)
         {
            _name = name;
            _id = new Id(id);
        }

        // Bind the class to an object container
        public void Bind(IActivator activator)
         {
            if (_activator == activator)
             {
                return;
            }
            if (activator != null && null != _activator)
             {
                throw new System.InvalidOperationException();
            }
            _activator = activator;
        }

        // activate the object fields
        public void Activate(ActivationPurpose purpose)
```

[1] Transparent Activation
[2] Transparent Persistence

```
         {
            if (_activator == null)
                return;
            _activator.Activate(purpose);
        }

        public Id Id
         {
            get
             {
                Activate(ActivationPurpose.Read);
                return _id;
            }
            set
             {
                Activate(ActivationPurpose.Write);
                _id = value;
            }
        }

        public string Name
         {
            get
             {
                // even simple string needs to be activated
                Activate(ActivationPurpose.Read);
                return _name;
            }
            set
             {
                Activate(ActivationPurpose.Write);
                _name = value;
            }
        }

        public override string ToString()
         {
            return string.Format("{0}[{1}]",Name, Id) ;
        }
    }

}
```

Pilot.vb

```
' Copyright (C) 2004 - 2008 Versant Inc. http://www.db4o.com

Imports Db4objects.Db4o
Imports Db4objects.Db4o.Activation
Imports Db4objects.Db4o.TA

Namespace Db4objects.Db4odoc.TP.Rollback
    Public Class Pilot
        Implements IActivatable
        Private _name As String
        Private _id As Id

        <Transient()> _
        Private _activator As IActivator
        Public Sub New(ByVal name As String, ByVal id As Integer)
            _name = name
            _id = New Id(id)
        End Sub

        ' Bind the class to an object container
        Public Sub Bind(ByVal activator As IActivator) _
Implements IActivatable.Bind
            If _activator Is activator Then
                Return
            End If
            If activator IsNot Nothing AndAlso _activator IsNot _
Nothing Then
                Throw New System.InvalidOperationException()
            End If
            _activator = activator
        End Sub

        ' activate the object fields
        Public Sub Activate(ByVal purpose As ActivationPurpose) _
Implements IActivatable.Activate
            If _activator Is Nothing Then
                Return
            End If
            _activator.Activate(purpose)
        End Sub

        Public Property Id() As Id
            Get
                Activate(ActivationPurpose.Read)
                Return _id
            End Get
            Set(ByVal value As Id)
                Activate(ActivationPurpose.Write)
                _id = value
            End Set
        End Property

        Public Property Name()Property Name() As String
            Get
                ' even simple string needs to be activated
                Activate(ActivationPurpose.Read)
                Return _name
            End Get
            Set(ByVal value As String)
                Activate(ActivationPurpose.Write)
                _name = value
            End Set
        End Property

        Public Overloads Overrides Function ToString() As String
            Return String.Format("{0}[{1}]", Name, Id)
        End Function
    End Class

End Namespace
```

Car

```
Car.cs
/** Copyright (C) 2004 - 2008 Versant Inc. http://www.db4o.com */
using System;
using Db4objects.Db4o.TA[1];
using Db4objects.Db4o.Activation;

namespace Db4objects.Db4odoc.TP[2].Rollback
 {
    public class Car : IActivatable
     {
        private string _model;
        private Pilot _pilot;
        /**//*activator registered for this class*/
        [System.NonSerialized]
        public IActivator _activator;


        public Car(string model, Pilot pilot)
         {
            _model = model;
            _pilot = pilot;
        }
        // end Car

        /**//*Bind the class to the specified object container, create the activator*/
        public void Bind(IActivator activator)
         {
            if (_activator == activator)
             {
                return;
            }
            if (activator != null && null != _activator)
             {
                throw new System.InvalidOperationException();
            }
            _activator = activator;
        }
        // end Bind

        public void Activate(ActivationPurpose purpose)
         {
            if (_activator == null)
                return;
            _activator.Activate(purpose);
        }
        // end Activate

        public string Model
         {
            get
             {
                Activate(ActivationPurpose.Read);
                return _model;
            }
            set
```

[1]Transparent Activation
[2]Transparent Persistence

```
            {
               Activate(ActivationPurpose.Write);
               _model = value;
            }
        }

        public Pilot Pilot
         {
            get
             {
               Activate(ActivationPurpose.Read);
               return _pilot;
            }
            set
             {
               Activate(ActivationPurpose.Write);
               _pilot = value;
            }
        }

        public void ChangePilot(String name, int id)
         {
            _pilot.Name = name;
            _pilot.Id.Change(id);
        }

        override public string ToString()
         {
            Activate(ActivationPurpose.Read);
            return string.Format("{0}[{1}]", _model, _pilot);
        }
        // end ToString
    }
}
```

```
Car.vb
' Copyright (C) 2004 - 2008 Versant Inc. http://www.db4o.com

Imports System
Imports Db4objects.Db4o
Imports Db4objects.Db4o.TA
Imports Db4objects.Db4o.Activation

Namespace Db4objects.Db4odoc.TP.Rollback
    Public Class Car
        Implements IActivatable
        Private _model As String
        Private _pilot As Pilot
        'activator registered for this class

        <Transient()> _
        Public _activator As IActivator


        Public Sub New(ByVal model As String, ByVal pilot As Pilot)
            _model = model
            _pilot = pilot
        End Sub
        ' end Car
        'Bind the class to the specified object container, create the activator

        Public Sub Bind(ByVal activator As IActivator) Implements IActivatable.Bind
            If _activator Is activator Then
                Return
            End If
            If activator IsNot Nothing AndAlso _activator IsNot Nothing Then
                Throw New System.InvalidOperationException()
            End If
            _activator = activator
        End Sub
        ' end Bind

        Public Sub Activate(ByVal purpose As ActivationPurpose) _
Implements IActivatable.Activate
            If _activator Is Nothing Then
                Return
            End If
            _activator.Activate(purpose)
        End Sub
        ' end Activate

        Public Property Model() As String
            Get
                Activate(ActivationPurpose.Read)
                Return _model
            End Get
            Set(ByVal value As String)
                Activate(ActivationPurpose.Write)
                _model = value
            End Set
        End Property

        Public Property Pilot() As Pilot
            Get
                Activate(ActivationPurpose.Read)
                Return _pilot
            End Get
            Set(ByVal value As Pilot)
                Activate(ActivationPurpose.Write)
                _pilot = value
            End Set
        End Property

        Public Sub ChangePilot(ByVal name As String, ByVal id As Integer)
            _pilot.Name = name
            _pilot.Id.Change(id)
        End Sub

        Public Overloads Overrides Function ToString() As String
            Activate(ActivationPurpose.Read)
            Return String.Format("{0}[{1}]", _model, _pilot)
        End Function
        ' end ToString
    End Class
End Namespace
```

Object Clone

Platform implementations of #clone is not compatible with **TP**[1].

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 Implementationyou 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.cs: StoreCar
private static void StoreCar()
         {
            File.Delete(Db4oFileName);
            IObjectContainer container = Database(Db4oFactory.NewConfiguration());
            if (container != null)
             {
                try
                 {
                    // create a car
                    Car car = new Car("BMW", new Pilot("Rubens Barrichello"));
                    container.Store(car);
                    Car car1 = (Car)car.Clone();
                    container.Store(car1);
                }
                finally
                 {
                    CloseDatabase();
                }
            }
        }
```

```
TPCloneExample.vb: StoreCar
Private Shared Sub StoreCar()
            File.Delete(Db4oFileName)
            Dim container As IObjectContainer = _
Database(Db4oFactory.NewConfiguration())
            If container IsNot Nothing Then
                Try
                    ' create a car
                    Dim car As New Car("BMW", New _
Pilot("Rubens Barrichello"))
                    container.Store(car)
                    Dim car1 As Car = DirectCast(car.Clone(), Car)
                    container.Store(car1)
                Finally
                    CloseDatabase()
                End Try
```

[1]Transparent Persistence

```
            End If
        End Sub
```

So it works for the first store, but what if we will clone an object retrieved from the database?

```
TPCloneExample.cs: TestClone
private static void TestClone()
         {
            IConfiguration configuration = ConfigureTP();

            IObjectContainer container = Database(configuration);
            if (container != null)
             {
                try
                 {
                    IObjectSet result = container.QueryByExample(new Car(null, null));
                    ListResult(result);
                    Car car = null;
                    Car car1 = null;
                    if (result.Size() > 0)
                     {
                        car = (Car)result[0];
                        System.Console.WriteLine("Retrieved car: " + car);
                        car1 = (Car)car.Clone();
                        System.Console.WriteLine("Storing cloned car: " + car1);
                        container.Store(car1);
                        container.Commit();
                    }
                }
                finally
                 {
                    CloseDatabase();
                }
            }
        }
```

```
TPCloneExample.vb: TestClone
Private Shared Sub TestClone()
            Dim configuration As IConfiguration = ConfigureTP()

            Dim container As IObjectContainer = Database(configuration)
            If container IsNot Nothing Then
                Try
                    Dim result As IObjectSet = container.QueryByExample( _
New Car(Nothing, Nothing))
                    ListResult(result)
                    Dim car As Car = Nothing
                    Dim car1 As Car = Nothing
                    If result.Size() > 0 Then
                        car = DirectCast(result(0), Car)
                        System.Console.WriteLine("Retrieved car: " _
+ car.ToString())
                        car1 = DirectCast(car.Clone(), Car)
                        System.Console.WriteLine("Storing cloned car: " _
+ car1.ToString())
                        container.Store(car1)
                        container.Commit()
                    End If
                Finally
                    CloseDatabase()
                End Try
            End If
        End Sub
```

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.cs: Clone
public object Clone()
         {
            Car test = (Car)base.MemberwiseClone();
            test._activator = null;
            return test;
        }
```

```
Car.vb: Clone
Public Function Clone() As Object Implements ICloneable.Clone
            Dim test As Car = DirectCast(MyBase.MemberwiseClone(), Car)
            test._activator = Nothing
            Return test
        End Function
```

Transparent Persistence Collections

In the previous example we reviewed how Transparent Persistence should be used with simple types. For your domain-classes this is quite easy to archive. But what about the .NET-collections? Wouldn't it be nice when the collections also work together the transparent persistence framework? So that when you add a object to a persistent collection, the object is also stored when committing?

For this purpose db4o brings special, transparent persistence aware collections with it. These collections store the added objects when on commit.

These are the same collections as for transparent activation and you can use them exactly the same way. See "TA Aware Collections" on page 61

Delete Behavior

Deleting an object is as simple as storing an object. You simply call the delete-method on the container to delete it. By default only the object you pass to the delete method is deleted. All referenced objects are not deleted.

```
Car car = FindCar(container);
container.Delete(car);
// We've deleted the only care there is
AssertEquals(0, AllCars(container).Count);
// The pilots are still there
AssertEquals(1, AllPilots(container).Count);
```

DeletionExamples.cs: Deleting object is as simple as storing

```
Dim car As Car = FindCar(container)
container.Delete(car)
' We've deleted the only care there is
AssertEquals(0, AllCars(container).Count)
' The pilots are still there
AssertEquals(1, AllPilots(container).Count)
```

DeletionExamples.vb: Deleting object is as simple as storing

Reference To Deleted Objects

What happens when you delete a object which is still referenced by other objects? Well in such cases that reference is set to null.

```
Pilot pilot = FindPilot(container);
container.Delete(pilot);
```

DeletionExamples.cs: Delete the pilot

```
Dim pilot As Pilot = FindPilot(container)
container.Delete(pilot)
```

DeletionExamples.vb: Delete the pilot

```
// Now the car's reference to the car is set to null
Car car = FindCar(container);
AssertEquals(null, car.Pilot);
```

DeletionExamples.cs: Reference is null after deleting

```
' Now the car's reference to the car is set to null
Dim car As Car = FindCar(container)
AssertEquals(Nothing, car.Pilot)
```

DeletionExamples.vb: Reference is null after deleting

Often you want to ensure that a object isn't referenced anymore, before you can delete it. However such referential integrity isn't supported at the moment. You need to implement your integrity checks manually, for example with callbacks.

Cascading Deletion And Collections.

Additionally you can configure cascading behavior for deletion. See "Cascading Deletion" on page 100

Also collections are treated like regular objects and need to be deleted explicitly. See "Collections And Arrays" on page 101

Cascading Deletion

By default db4o only deletes objects which are passed to the delete-method and doesn't delete referenced objects. You can easily change that. Configure the cascading deletion behavior in the configuration for certain classes or certain fields.

For example we mark that the object in the 'pilot'-field is also deleted:

```
IEmbeddedConfiguration config = Db4oEmbedded.NewConfiguration();
config.Common.ObjectClass(typeof (Car)).ObjectField("pilot").CascadeOnDelete(true);
using (IObjectContainer container = Db4oEmbedded.OpenFile(config, DatabaseFile))
{
```

DeletionExamples.cs: Mark field for cascading deletion

```
Dim config As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
config.Common.ObjectClass(GetType(Car)).ObjectField("pilot").CascadeOnDelete(True)
Using container As IObjectContainer = Db4oEmbedded.OpenFile(config, DatabaseFile)
```

DeletionExamples.vb: Mark field for cascading deletion

When we now delete the car, the pilot of that car is also deleted.

```
Car car = FindCar(container);
container.Delete(car);
// Now the pilot is also gone
AssertEquals(0, AllPilots(container).Count);
```

DeletionExamples.cs: Cascade deletion

```
Dim car As Car = FindCar(container)
container.Delete(car)
' Now the pilot is also gone
AssertEquals(0, AllPilots(container).Count)
```

DeletionExamples.vb: Cascade deletion

Collections And Arrays

Collections and arrays don't have a special behavior in db4o. When you delete a collection, the collection-members are not deleted. The collection and objects are two independent objects for db4o.

Removing From A Collection

To remove object from a collection you can simple use the regular collection-operations and then store that collection.

```
PilotGroup group = FindGroup(container);
Pilot pilot = group.Pilots[0];
group.Pilots.Remove(pilot);
container.Store(group.Pilots);

AssertEquals(3, AllPilots(container).Count);
AssertEquals(2, group.Pilots.Count);
```

DeletionExamples.cs: Removing from a collection doesn't delete the collection-members

```
Dim group As PilotGroup = FindGroup(container)
Dim pilot As Pilot = group.Pilots(0)
group.Pilots.Remove(pilot)
container.Store(group.Pilots)

AssertEquals(3, AllPilots(container).Count)
AssertEquals(2, group.Pilots.Count)
```

DeletionExamples.vb: Removing from a collection doesn't delete the collection-members

Remove And Delete Collection Members

If you want to delete a collection-member, remove it and then delete it.

```
PilotGroup group = FindGroup(container);
Pilot pilot = group.Pilots[0];
group.Pilots.Remove(pilot);
container.Store(group.Pilots);
container.Delete(pilot);

AssertEquals(2, AllPilots(container).Count);
AssertEquals(2, group.Pilots.Count);
```

DeletionExamples.cs: Remove and delete

```
Dim group As PilotGroup = FindGroup(container)
Dim pilot As Pilot = group.Pilots(0)
group.Pilots.Remove(pilot)
container.Store(group.Pilots)
container.Delete(pilot)

AssertEquals(2, AllPilots(container).Count)
AssertEquals(2, group.Pilots.Count)
```

DeletionExamples.vb: Remove and delete

Indexing

db4o supports indexes like most databases do. Indexes are data structures which allow efficient lookup of data. When you enable an index, db4o will add an entry to index for each object. This makes the insert and update operation a little slower. However it makes queries a lot faster. A query which uses an index is a order of magnitude faster that a query which cannot use a index.

You can create a index by enabling it on a field. See "Adding a Field Index" on page 148

Note that you need to set the index on a field, not a property. This is also true for auto-properties.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Person)).ObjectField("name").Indexed(true);
```

ObjectFieldConfigurations.cs: Index a certain field

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).ObjectField("name").Indexed(True)
```

ObjectFieldConfigurations.vb: Index a certain field

As an alternative you also can use the appropriate Attribute on the field which you want to index.

```
[Indexed]
private string zipCode;
```

ObjectFieldConfigurations.cs: Index a field

```
<Indexed()> _
Private m_zipCode As String
```

ObjectFieldConfigurations.vb: Index a field

When And Where Do I Need An Index

When do you need an index? As a rule of thumb: Add an index all fields which are used in queries. See "When And Where" on page 103

There are different factors which need to be fulfilled to profit from an index.

- Read operations dominate the database operations: When you application only writes objects but rarely query for objects, there's no benefit of faster queries. However in most system reading is the dominate operation and should be fast.
- You're using the field / class in a query: If no query touches the field or class you're have a index on, you have not benefit from the index. The index improves only the query performance, but slows down store and update-operations.
- You actually need a substantial amount of data. The performance gains of an index are negligible for small data sets. When you test indexes use 10'000 and more objects.

When And Where

When do you need an index? And on which fields do I need an index? An index has a costs and benefits. See "Costs And Benefits" on page 104. Therefore you should only add one when you actually benefit from it.

When To Add Queries

Indexes speed up queries but slow down store and delete operations. You only need to add indexes when queries are too slow. There's no real benefit to add a index when your queries are already fast enough.

Also add index only on fields which are used in queries. Indexes on fields which are never used in a query have no benefit.

Where To Add Queries

How do I find the queries which can benefit from indexes? How do I find queries which couldn't utilize indexes? You can use the diagnostic-API find out. Filter for the LoadedFromClassIndex-message. Every time this message arrives the query didn't use a field index. You can add a break-point to the message-output and find out which query is the source and then add the index.

```
internal class IndexDiagnostics : IDiagnosticListener
{
    public void OnDiagnostic(IDiagnostic diagnostic)
    {
        if (diagnostic is LoadedFromClassIndex)
        {
            Console.WriteLine("This query couldn't use field indexes " +
                              ((LoadedFromClassIndex) diagnostic).Reason());
            Console.WriteLine(diagnostic);
        }
    }
}
```

WhereToIndexExample.cs: Index diagnostics

```
Friend Class IndexDiagnostics
    Implements IDiagnosticListener
    Public Sub OnDiagnostic(ByVal diagnostic As IDiagnostic) _
        Implements IDiagnosticListener.OnDiagnostic
        If TypeOf diagnostic Is LoadedFromClassIndex Then
            Console.WriteLine("This query couldn't use field indexes " & DirectCast(diagnostic, LoadedFromClassIn
            Console.WriteLine(diagnostic)
        End If
    End Sub
End Class
```

WhereToIndexExample.vb: Index diagnostics

```
configuration.Common.Diagnostic.AddListener(new IndexDiagnostics());
```

WhereToIndexExample.cs: Find queries which cannot use index

```
configuration.Common.Diagnostic.AddListener(New IndexDiagnostics())
```

WhereToIndexExample.vb: Find queries which cannot use index

Costs And Benefits

Maintaining an index is additional work and therefore it has its costs. Each time you store a objects which has an index on it, db4o needs to look up the a appropriate place in the index and store a entry there. This costs is paid for each store, update or delete operation. Therefore there's a tradeoff when adding indexes.

The costs are mostly the time consumed to maintain the index. Additionally there's some additional space consumed to store the index. The benefits is that queries which run a order of magnitude faster when using an index.

In practice you should measure and benchmark your solution and check if a index is a benefit to your application. Ensure that you have enough test data. The influence of indexes shows better with a lot of objects. (10'000 and more objects). However since most applications do a lot more read operations than store and update operations adding a index brings a huge benefit.

Types And Limitations

There are limitations to the db4o indexing. Not all types can be indexed.

Types Which Can Be Indexed

Basically all primitive types like int, long, double, decimals, enums etc can be indexed. Indexes on primitive types work extremely well.

Additionally you can index strings, which are handled by db4o like primitives. Strings can be arbitrary long, so a index on string is usually slower than a index on a primitive value. But it's still fast for straight lookups. Note also that a string index can only be used for equality comparison. Comparisons like contains, start with etc don't use the index.

The DateTime, DateTimeOffset and Guid can also be indexed without any issues.

You also can index any object reference except arrays and strings, which are handled like primitives. This means you can index a field which holds a reference to a object and then look up for objects which have a certain reference.

Types Which Cannot Be Indexed

Arrays and collections cannot be indexed. The current db4o index implementation cannot deal with those types. This also means that you cannot do fast look-ups on arrays or collections.

Limitations

Currently the index on strings cannot be used for advanced comparisons like contains, starts with etc.

Check For Existing Indexes

Sometime you may want to know if a index exists on a certain field. You can use the db4o-meta information to find out if a field is indexed.

```
IStoredClass metaInfo = container.Ext().StoredClass(typeof(IndexedClass));
// list a fields and check if they have a index
foreach (IStoredField field in metaInfo.GetStoredFields())
{
    if (field.HasIndex())
    {
        Console.WriteLine("The field '" + field.GetName() + "' is indexed");
    }
    else
    {
        Console.WriteLine("The field '" + field.GetName() + "' isn't indexed");
    }
}
```

CheckForAndIndex.cs: Check for a index

```
Dim metaInfo As IStoredClass = container.Ext().StoredClass(GetType(IndexedClass))
' list a fields and check if they have a index
For Each field As IStoredField In metaInfo.GetStoredFields()
    If field.HasIndex() Then
        Console.WriteLine("The field '" & field.GetName() & "' is indexed")
    Else
        Console.WriteLine("The field '" & field.GetName() & "' isn't indexed")
    End If
```

CheckForAndIndex.vb: Check for a index

Configuration

db4o is configured by its configuration API. The configuration allows you to adjust db4o to your scenarios.

Configure db4o

In order to configure db4o, you need to create a new configuration-instance and set the desired settings on it. After that, you pass the configuration-instance to the object-container factory.

Note that you cannot share a configuration-instance. For each object-container you create, you need to pass in a new configuration-instance. It's recommended to create a method which will return a new configuration instance on request:

Embedded-Configuration

For an embedded container you can configure the common-, file- and id-system-configuration.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
// change the configuration...
IObjectContainer container = Db4oEmbedded.OpenFile(configuration, "database.db4o");
```

ConfigurationBasics.cs: Configure embedded object container

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
' change the configuration...
Dim container As IObjectContainer = Db4oEmbedded.OpenFile(configuration, "database.db4o")
```

ConfigurationBasics.vb: Configure embedded object container

Server-Configuration

For an server you can configure the common-, file-, networking-, server- and id-system-configuration.

```
IServerConfiguration configuration = Db4oClientServer.NewServerConfiguration();
// change the configuration...
IObjectServer server = Db4oClientServer.OpenServer(configuration, "database.db4o", 1337);
```

ConfigurationBasics.cs: Configure the db4o-server

```
Dim configuration As IServerConfiguration = Db4oClientServer.NewServerConfiguration()
' change the configuration...
Dim server As IObjectServer = Db4oClientServer.OpenServer(configuration, "database.db4o", 1337)
```

ConfigurationBasics.vb: Configure the db4o-server

Client-Configuration

For an client you can configure the common-, networking-, client- and id-system-configuration.

```
IClientConfiguration configuration = Db4oClientServer.NewClientConfiguration();
// change the configuration...
IObjectContainer container = Db4oClientServer.OpenClient(configuration, "localhost", 1337, "user", "pwd");
```

ConfigurationBasics.cs: Configure a client object container

```
Dim configuration As IClientConfiguration = Db4oClientServer.NewClientConfiguration()
' change the configuration...
Dim container As IObjectContainer = Db4oClientServer.OpenClient(configuration, "localhost", 1337, "user", "pwd")
```

ConfigurationBasics.vb: Configure a client object container

Configuration Is Not Persistent

The db4o configuration is not persistent with a few exception. This means that you need to configure db4o each time you create a object-container instance.

Configuration in Client/Server-Mode

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.

Configuration-Settings Overview

The configuration-settings which are common across -client, embedded and db4o-server are summed up here: See "Common Configuration" on page 107

Common Configuration

The common-configuration applies to the embedded-, client- and the server-mode of db4o. All the common configuration is accessible via the common-property on the configuration-object.

Overview

Here's a overview over all common configuration-settings which you can change:

	Same in **C/S**[1][2]	Can not change [3]
ActivationDepth: Change globally the activation-depth.		
Aliases: Configure aliases for class and package-names.		
AllowVersionUpdates: Allow/Disallow to update the database-format.	Yes	
AutomaticShutDown: Close the database when the application exits.		
BTreeNodeSize: Tune the size of the B-tree-node which are used for the indexes.		
Callbacks: Turn object-callbacks on an off.		
CallConstructors: Use or bypass the constructor for creating objects.		
DetectSchemaChanges: Disable/Enable schema changes detection.		
Diagnostic: Add diagnostic-listeners.		
ExceptionsOnNotStorable: Enable/Disable exceptions on not stor-		

[1]Client-Server
[2]This setting needs to be the same on the server and all clients.
[3]This setting has to be set the first time when the database is created. You cannot change is for an existing database.

able objects.		
InternStrings: Will call the intern-method on the retrieved strings.		
MarkTransient: Configure a Attribute for marking fields as transient.		
MessageLevel: Configure the logging-message level.		
NameProvider: Configure the toString() value of the object-container.		
ObjectClass: Configure class-specific settings.		
OptimizeNativeQueries: Enable runtime query optimization.		
OutStream: Configure the log message output stream		
Queries: Configure query behaviors		
ReflectWith: Configure a reflector.		
RegisterTypeHandler: Register a new TypeHandler.	Required	
StringEncoding: Configure the string-encoding.	Required	Yes
TestConstructors: Configure if db4o checks for valid constructors.		
UpdateDepth: Change the update-depth.		
WeakReferenceCollectionInterval: Change the weak-reference cleanup interval. Default setting is 1000 milliseconds.		
WeakReferences: Enable/disable weak references.		

Additional Configuration Items

There are additional configuration items which add for additional features. You can add then on the common- configuration. For example to enable transparent persistence you add the TransparentPersistenceSupport configuration item. Take a look a the available configuration items. See "Common Configuration Items" on page 108

Common Configuration Items

Configuration items add special capability to the system. Here's a list of all configuration items available.

TransparentActivationSupport

Support for transparent activation. See "Transparent Activation" on page 51

TransparentPersistenceSupport

Support for transparent persistence. See "Transparent Persistence" on page 75

UniqueFieldValueConstraint

Set up unique Field constraints. See "Unique Constraints" on page 166

FreespaceMonitoringSupport

Enables you to monitor the free-space-manager. See "Runtime Monitoring" on page 320

IOMonitoringSupport

Enables you to monitor the IO-activity of db4o. See "Runtime Monitoring" on page 320

NativeQueryMonitoringSupport

Enables you to monitor the native queries. See "Runtime Monitoring" on page 320

NetworkingMonitoringSupport

Enables you to monitor the network activity of db4o. See "Runtime Monitoring" on page 320

ObjectLifecycleMonitoringSupport

Enables you to monitor the object lifecycles. See "Runtime Monitoring" on page 320

QueryMonitoringSupport

Enables you to monitor db4o queries. See "Runtime Monitoring" on page 320

ReferenceSystemMonitoringSupport

Enables you to monitor db4o's reference system. See "Runtime Monitoring" on page 320

Activation Depth

db4o uses the concept of activation to avoid loading to much data into memory. You can change the global activation depth with this setting.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ActivationDepth = 2;
```

CommonConfigurationExamples.cs: Change activation depth

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ActivationDepth = 2
```

CommonConfigurationExamples.vb: Change activation depth

A higher activation depth is usually more convenient to work with, because you don't face inactivated objects. However, a higher activation depth costs performance, because more data has to read from the database. Therefore a good balance need to be found. Take also a look a transparent activation, since it solves the activation issue completely.

Class Specific Configuration

You can also configure a class specific activation depth. See "Class Specific Configuration" on page 144

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Person)).MinimumActivationDepth(2);
```

ObjectConfigurationExamples.cs: Set minimum activation depth

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).MinimumActivationDepth(2)
```

ObjectConfigurationExamples.vb: Set minimum activation depth

Update Depth

By default db4o only stores changes on the updated object but not the changes on referenced objects. With a higher update-depth db4o will traverse along the object graph to a certain depth and update all objects. See "Update Concept" on page 71

With the update-depth you configure how deep db4o updates the object-graph.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.UpdateDepth = 2;
```

CommonConfigurationExamples.cs: Increasing the update-depth

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.UpdateDepth = 2
```

CommonConfigurationExamples.vb: Increasing the update-depth

A higher update depth is usually more convenient, because you don't need to explicitly store each changed object. However the higher the update depth is the more time it takes to update the objects. Therefore it is a tradeoff. Note that you can also use transparent persistence, which takes of updating the right objects.

Class Specific Configuration

You can also configure a class specific update depth. See "Class Specific Configuration" on page 144

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Person)).UpdateDepth(2);
```

ObjectConfigurationExamples.cs: Set the update depth

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).UpdateDepth(2)
```

ObjectConfigurationExamples.vb: Set the update depth

Aliases

Aliases allow you to use different names for your persistent classes in the database and in your application. Before a class is saved, db4o checks if an alias exists. If this is the case, the alias-name is used instead of the original name.

Adding Aliases

You can add aliases by adding instance of the Alias-interface to the configuration. You can add as many aliases to the configuration as you want. Following options are available.

- TypeAlias: Allows you to give an alias for a certain type.
- WildCardAlias: Allows you to give an alias for a namespace / multiple types. It allows you to use a wildcard for the name. (The *-character)
- Your own implementation, by implementing the Alias-interface.

Each alias has two arguments. The first argument it the name which the type has in the database. The second argument is the type which is currently used. Ensure that you alias only types which can be resolved.

Note that in .NET the types-names contain the full namespace, classname and the assembly-name. For example the when you have an assembly called 'MyProject', a namespace 'Company.Project' and a class named Person, then the name of the type is: 'Company.Project.Person, MyProject'

Furthermore the order in which you add the aliases matters. Add first the most specific alias and the go to the more general alias. For example add first all TypeAlias and then the WildCardAlias.

```
// add an alias for a specific type
configuration.Common.AddAlias(
    new TypeAlias("Db4oDoc.Code.Configuration.Alias.OldTypeInDatabase, Db4oDoc",
                  "Db4oDoc.Code.Configuration.Alias.NewType, Db4oDoc"));
// or add an alias for a whole namespace
configuration.Common.AddAlias(
    new WildcardAlias("Db4oDoc.Code.Configuration.Alias.Old.Namespace.*, Db4oDoc",
                      "Db4oDoc.Code.Configuration.Alias.Current.Namespace.*, Db4oDoc"));
```

AliasExamples.cs: Adding aliases

```
' add an alias for a specific type
configuration.Common.AddAlias(New TypeAlias("Db4oDoc.Code.Configuration.Alias.OldTypeInDatabase, Db4oDoc", _
                                            "Db4oDoc.Code.Configuration.Alias.NewType, Db4oDoc"))
' or add an alias for a whole namespace
configuration.Common.AddAlias(New WildcardAlias("Db4oDoc.Code.Configuration.Alias.Old.Namespace.*, Db4oDoc", _
                                                "Db4oDoc.Code.Configuration.Alias.Current.Namespace.*, Db4oDoc"))
```

AliasExamples.vb: Adding aliases

Allow Version Updates

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, db4o does not upgrade databases by default. Database upgrading can be turned on with the following configuration switch:

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.AllowVersionUpdates = true;

// reopen and close the database to do the update
IObjectContainer container = Db4oEmbedded.OpenFile(configuration, DatabaseFile);
container.Close();
```

CommonConfigurationExamples.cs: Update the database-format

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.AllowVersionUpdates = True

' reopen and close the database to do the update
Dim container As IObjectContainer = Db4oEmbedded.OpenFile(configuration, DatabaseFile)
container.Close()
```

CommonConfigurationExamples.vb: Update the database-format

Please note that, once the database file version is updated, there is no way to get back to the older version. When 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.
Recommendations for upgrading:

- Backup your database file to be able to switch back.
- Defragmenting a database file with the new db4o version after upgrading can make the database more efficient.

Automatic Shutdown

With this setting you can disable the shutdown monitoring for db4o. By default, db4o will close the database when the CLR exits. However on some embedded devices this can lead to issues. So disable it when you experience problems when terminating the application.

Note that its recommended to dispose the object-container / -server when you close the application anyway, even when this setting is enabled.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.AutomaticShutDown = false;
```

CommonConfigurationExamples.cs: Disable automatic shutdown

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.AutomaticShutDown = False
```

CommonConfigurationExamples.vb: Disable automatic shutdown

B-Tree Node Size

db4o uses B-tree indexes for increased query performance and reduced memory consumption. B-trees are used for field-index, class-index and for the id-system. B-trees are optimized for scenarios where a part of the data is on secondary storage such as a hard disk, since disk accesses is extremely expensive. B-trees minimize the number of disk accesses required. You find more information about B-trees on the internet.

You can tune the B-trees by configuring the size of a node. Larger node sizes require less disk access, since a node is read on one read-operation. However it consumes more memory to keep the larger nodes available. Also larger nodes take longer to write back to disk. You need to tune the B-tree node size according to your application requirements. When benchmarking the settings, use large data sets. The influence of B-trees is more significant for larger dabases with 100'000 and more objects.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.BTreeNodeSize = 256;
```

CommonConfigurationExamples.cs: Change B-tree node size

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.BTreeNodeSize = 256
```

CommonConfigurationExamples.vb: Change B-tree node size

Disable Callbacks

db4o supports callback-methods on each class. In order to support this, db4o scans all persistent classes and looks for the callback-method signature. When a lot of different classes are stored, this may take some time, especially on embedded devices. Therefore you can disable the object callbacks, when you don't need them.

Note that this doesn't disable the global events. These can still be used, even when callbacks are disabled.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.Callbacks = false;
```

CommonConfigurationExamples.cs: Disable callbacks

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.Callbacks = False
```

CommonConfigurationExamples.vb: Disable callbacks

Calling Constructors

By default db4o by passes the constructor to instantiate the objects. Therefore it uses the reflection capabilities of the current platform. On some embedded platform this is not possible because of security and platform constrains. On other platforms bypassing the constructor is significantly slower than calling it. Therefore you change the behavior so that db4o calls the constructor. Note that when you enable this setting, you classes need a constructor which can be called with default-arguments without throwing an exception.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.CallConstructors = true;
```

CommonConfigurationExamples.cs: Call constructors

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.CallConstructors = True
```

CommonConfigurationExamples.vb: Call constructors

Class Specific Configuration

You can enable this setting only for certain classes. For example when the constructor needs to initialize transient fields.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Person)).CallConstructor(true);
```

ObjectConfigurationExamples.cs: Call constructor

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).CallConstructor(True)
```

ObjectConfigurationExamples.vb: Call constructor

Disable Schema Change Detection

db4o scans the class structure to find out the schema of the objects. This takes a little time. When a lot of classes are persistent this may take some time, especially on embedded devices.

Therefore you can disable this check. You can disable it only, when db4o already knows all stored classes. This means a object of each class has already been stored once. Furthermore there shouldn't be any further changes.

This setting is only useful for very special scenarios with no schema evolution at all. Otherwise this setting may cause strange and subtle errors!

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.DetectSchemaChanges = false;
```

CommonConfigurationExamples.cs: Disable schema evolution

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.DetectSchemaChanges = False
```

CommonConfigurationExamples.vb: Disable schema evolution

Diagnostics

Enables you to add diagnostic listeners to db4o. Read more in the tuning chapters. See "Diagnostics" on page 328

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.Diagnostic.AddListener(new DiagnosticToConsole());
```

CommonConfigurationExamples.cs: Add a diagnostic listener

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.Diagnostic.AddListener(New DiagnosticToConsole())
```

CommonConfigurationExamples.vb: Add a diagnostic listener

Exceptions On Not Storable Objects

When db4o cannot store a object, it will throw an exception. This is the default behavior. When you disable this object, db4o will silently ignore objects which cannot be stored instead of throwing an exception.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ExceptionsOnNotStorable = false;
```

CommonConfigurationExamples.cs: Disable exceptions on not storable objects

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ExceptionsOnNotStorable = False
```

CommonConfigurationExamples.vb: Disable exceptions on not storable objects

Intern Strings

You can configure db4o to call the intern method on all strings. See more on the intern method for your platform.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.InternStrings = true;
```

CommonConfigurationExamples.cs: intern strings

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.InternStrings = True
```

CommonConfigurationExamples.vb: intern strings

Benefits

When a lot of strings contain a the exact same content, calling intern on them can safe some memory.

Disadvantage

Calling intern on a string adds that string to a global pool. Therefore this string cannot be garbage collected. So when you load a lot of strings which you use only once, you can run into memory-problems.

Mark Transient

This allows you to configure additional Attributes to mark fields as transient. You can add multiple such Attributes by calling the method for each Attribute once.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.MarkTransient(typeof (TransientMarkerAttribute).FullName);
```

CommonConfigurationExamples.cs: add an transient marker annotatin

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.MarkTransient(GetType(TransientMarkerAttribute).FullName)
```

CommonConfigurationExamples.vb: add an transient marker annotatin

Message Level

This allows you to enable debug-messages of db4o. Currently four message levels are supported:

- Level = 0: No messages, default configuration.
- Level> 0: Normal messages.
- Level> 1: State messages (new object, object update, delete);
- Level> 2: Activation messages (object activated, deactivated).

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.MessageLevel = 4;
```

CommonConfigurationExamples.cs: Change the message-level

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.MessageLevel = 4
```

CommonConfigurationExamples.vb: Change the message-level

By default the output is send to the console. But you can redirect the output to any output stream. See "Changing the Output Stream" on page 116

Name Provider

You can overwrite how the object-containers is named. This can be helpful for debugging multiple-container scenarios.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.NameProvider(new SimpleNameProvider("Database"));
```

CommonConfigurationExamples.cs: set a name-provider

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.NameProvider(New SimpleNameProvider("Database"))
```

CommonConfigurationExamples.vb: set a name-provider

Disable Optimize Native Queries

Normally db4o tries to optimize native queries at runtime. See "Native Query Optimization" on page 21

However on some limited embedded platforms like Android this doesn't work. In such cases you can disable the native query optimizer and use instead the compile time optimizer.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.OptimizeNativeQueries = false;
```

CommonConfigurationExamples.cs: Disable the runtime native query optimizer

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.OptimizeNativeQueries = False
```

CommonConfigurationExamples.vb: Disable the runtime native query optimizer

Register Type Handler

You can add special type-handlers for your datatypes. This allows you to plug in your own serialization-handling for that type. See "TypeHandlers" on page 204

Note that you need the type-handler on the client and server installed.

Changing the Output Stream

Normally the debug messages of db4o are printed to the default console. However you can configure db4o to send the information to any output stream:

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.OutStream = Console.Out;
```

CommonConfigurationExamples.cs: Change the output stream

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.OutStream = Console.Out
```

CommonConfigurationExamples.vb: Change the output stream

Query Modes

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

Situations, when query result is bigger than the memory available or when query time is longer than the whole functional operation.

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

- Immediate: The default mode. The result is determined immediately and only the objects are lazy loaded.
- Lazy: The query is evaluated while iterating to the result.
- Snapshot: Runs the index-processing and creates a snapshot of the result. Further processing is done while iterating over the result.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.Queries.EvaluationMode(QueryEvaluationMode.Immediate);
```

CommonConfigurationExamples.cs: Change the query mode

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.Queries.EvaluationMode(QueryEvaluationMode.Immediate)
```

CommonConfigurationExamples.vb: Change the query mode

Immediate Queries

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

Obviously evaluation takes some time and in a case of big result sets 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.

This mode makes the whole objects result set available at once. An result 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 the result set 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:

Advantages

- 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).

Disadvantages

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

Lazy Queries

With the Lazy Querying no criteria evaluated at all. Instead an iterator is created against the best index found. Further query processing, including all index processing, will happen when the application iterates through the result. This allows you to get the first query results almost immediately.

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

Advantages

- 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.

Disadvantages

- Lazy queries check candidates when iterating through the 'result'. 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 same ways as 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.

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` result set of the query.

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

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.

Advantages

- 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.

Disadvantages

- The entire candidate index will be loaded into memory. It will stay there until the query ObjectSet is garbage collected. In a **C/S**[1] setup, the memory will be used on the server side

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

Changing The Reflector

This setting allows you to change the reflector for db4o. The reflector is responsible to inspect the metadata of objects and report them to db4o. See "db4o Reflection API" on page 218

There are currently two reflectors available:

- **NetReflector**: The default reflector which uses standard .NET reflection for inspecting and accessing objects.
- **FastNetReflector**: A reflector which generates special access-code for each class. Therefore it is a lot faster than the default NetReflector. However it has a higher memory-consumption due to the generated code.

[1]Client-Server

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ReflectWith(new FastNetReflector());
```

CommonConfigurationExamples.cs: Change the reflector

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ReflectWith(New FastNetReflector())
```

CommonConfigurationExamples.vb: Change the reflector

String Encoding

When db4o stores and loads strings it has to convert them to and from byte arrays. By default db4o provides three types of string encoding, which do this job: Unicode, UTF-8 and ISO 8859-1, Unicode being the default one. In general Unicode can represent any character set. However, it also imposes a certain overhead, as character values are stored in 4 bytes (less generic encoding usually use 2 or even 1 byte per character). In order to save on the database file size, it is recommended to use UTF-8, which only required one byte per character.

Note that the string encoding need to be the same on the server and clients. Also you cannot change the string encoding for an existing database. If you want to change the encoding, you need to defragment the database.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.StringEncoding = StringEncodings.Utf8();
```

CommonConfigurationExamples.cs: Use the utf8 encoding

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.StringEncoding = StringEncodings.Utf8()
```

CommonConfigurationExamples.vb: Use the utf8 encoding

Of course you can also implement your own string encoding. You just need to implement the string encoding interface and pass it to this configuration.

Test For Constructors

This setting is only relevant for some embedded platforms which need a constructor (for example Android).

On startup, db4o checks that the classes have a callable constructor. This may take some time, especially on embedded platforms. When you run in production, you can disable this check, when you are sure that all classes can be instantiated.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.TestConstructors = false;
```

CommonConfigurationExamples.cs: Disable testing for callable constructors

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.TestConstructors = False
```

CommonConfigurationExamples.vb: Disable testing for callable constructors

Weak Reference Collection Interval

By default db4o uses weak references to keep track of loaded objects. These weak references need to be clean up from time to time. You can change this collection-interval.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.WeakReferenceCollectionInterval = (10*1000);
```

CommonConfigurationExamples.cs: change weak reference collection interval

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.WeakReferenceCollectionInterval = (10 * 1000)
```

CommonConfigurationExamples.vb: change weak reference collection interval

Disable Weak References

By default db4o uses weak references cache to all loaded objects. This ensures that the objects can be garbage collected. However it does impose a small overhead. You can disable weak reference if you like. Then db4o uses regular references. When disabled you need to remove objects explicit from the cache.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.WeakReferences = false;
```

CommonConfigurationExamples.cs: Disable weak references

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.WeakReferences = False
```

CommonConfigurationExamples.vb: Disable weak references

File Configuration

The file-configuration contains all the configuration-settings which are related to file access. It can be set in the db4o-embedded-container or on the db4o-server. All the file configuration is accessible via the file-property on the configuration-object.

Overview

Here's a overview over all file configuration-settings which you can change:

	Can not change [1]	Only useful for recovery [2]
AsynchronousSync: Enables asynchronous commits.		
BlobPath: Specify where blobs are stored.		
BlockSize: Set the block-size of the database. Larger Blocks allow larger databases.	Yes	
DatabaseGrowthSize: Set the grow step size when the database-file is to small.		
DisableCommitRecovery: Disable the commit-recovery.		Yes
Freespace: Configure the free-space system.		
GenerateUUIDs: Configure to generate UUIDs for objects.		

[1]This setting has to be set the first time when the database is created. You cannot change is for an existing database.
[2]This setting is only useful when you try to recover a corrupted database file.

~~GenerateVersionNumbers~~: Deprecated. Generate commit timestamps instead.		
LockDatabaseFile: Enable/disable the database file lock.		
ReadOnly: Set the database to read only mode.		
RecoveryMode: Set the database to a recovery mode.		Yes
ReserveStorageSpace: Reserve storage-space up front.		
Storage: Configure the storage system.		

Asynchronous File-Sync

One of the most costly operations during commit is flushing the buffers of the database file. In regular mode the commit call has to wait until this operation has completed.

When asynchronous sync is turned on, the sync operation will run in a dedicated thread, blocking all other file access until it has completed. This way the commit can returns immediately. This allows db4o and other processes to continue running side-by-side while the flush call is executed.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.File.AsynchronousSync = true;
```

FileConfiguration.cs: Allow asynchronous synchronisation of the file-flushes

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.File.AsynchronousSync = True
```

FileConfiguration.vb: Allow asynchronous synchronisation of the file-flushes

Advantage

The commit call will return much faster. Because it doesn't have to wait until everything is written to disk.

Disadvantage

After the commit-call, you have no guaranties that everything is persistent. Maybe the commit is still in progress. On a failure, this means that you can loose a commit.

A setup with this option still guarantees ACID transaction processing: A database file is always either in the state before commit or in the state after commit. Corruption can not occur. You can just not rely on the transaction already having been applied when the commit() call returns.

Blob Path

With this setting you can specify in which directory the Blob-files are stored. The db4o-Blobs are stored outside the database file as regular files. With this settings you can set where this directory is. The default is the 'blob'-directory.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.File.BlobPath = "myBlobDirectory";
```

FileConfiguration.cs: Configure the blob-path

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.File.BlobPath = "myBlobDirectory"
```

FileConfiguration.vb: Configure the blob-path

Block Size

The block-size determines how how large a database can be. This value can be between 1 and 127. The default is 1. The resulting maximum database file size will be a multiple of 2GB. For example a a block-size of 8 allows a database-size up to 16 G. With the largest possible setting, 127, the database can grow up to 254 GB. A recommended setting for large database files is 8, since internal pointers have this length.

This configuration-setting has to be set the first time the database is created. You cannot change it for an existing database. If you want to change it, you need to deframent the database.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.File.BlockSize = 8;
```

FileConfiguration.cs: Increase block size for larger databases

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.File.BlockSize = 8
```

FileConfiguration.vb: Increase block size for larger databases

Advantages and Disadvantages

A larger value allows a larger database. However, since objects are aligned to the block size, a larger value will result in less efficient storage space usage. Furthermore a larger value may decrease the performance, because it causes more cache misses.

Database Growth Size

Configures how much the database-file grows, when there not enough space. Whenever no free space in the database is large enough to store a object the database file is enlarged. This setting configures by how much it should be extended, in bytes. This configuration setting is intended to reduce fragmentation. Higher values will produce bigger database files and less fragmentation. To extend the database file, a single byte array is created and written to the end of the file in one write operation. Be aware that a high setting will require allocating memory for this byte array.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.File.DatabaseGrowthSize = 4096;
```

FileConfiguration.cs: Configure the growth size

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.File.DatabaseGrowthSize = 4096
```

FileConfiguration.vb: Configure the growth size

Disable Commit Recovery

This option only makes sense when you try to recover a corrupted database.

Turns commit recovery off. db4o uses a two-phase commit algorithm to ensure ACID properties. 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.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.File.DisableCommitRecovery();
```

FileConfiguration.cs: Disable commit recovery

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.File.DisableCommitRecovery()
```

FileConfiguration.vb: Disable commit recovery

Freespace Configuration

When objects are updated or deleted, the space previously occupied in the database file is marked as "free". The freespace management system takes care of this space by maintaining which places in the file are free.

Discarding Threshold

By default the free space system keeps and maintains every bit of free space even the smallest ones. Very small blocks of storage are hard to reuse, because larger objects don't fit in. Therefore overtime more and more small blocks of free space are maintained. Maintaining these small free spaces can cost performance. Therefore you can configure the free-space system to discard small blocks. Then small blocks are not maintained as free space. On the downside these space is lost until the next defragmentation.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
// discard smaller than 256 bytes
configuration.File.Freespace.DiscardSmallerThan(256);
```

FreeSpaceConfiguration.cs: Discard settings

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
' discard smaller than 256 bytes
configuration.File.Freespace.DiscardSmallerThan(256)
```

FreeSpaceConfiguration.vb: Discard settings

Memory Freespace System

When you use the memory free-space system the information of the free space locations is hold in memory. This is the fastest way to manage free space. However when the database is shut down abnormally, for example by a crash or power off, the free space information is lost. The space only can be reclaimed by defragmentation. This is the default-setting used by db4o.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.File.Freespace.UseRamSystem();
```

FreeSpaceConfiguration.cs: Use the in memory system

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.File.Freespace.UseRamSystem()
```

FreeSpaceConfiguration.vb: Use the in memory system

BTree Freespace System

The B-tree free space system hold the information a B-tree which is stored on commits. Since the free space information is stored on disk, it is usually a slower then the memory free space system. However it doesn't loose the information on abnormal termination. Additionally the B-tree free space system uses less memory resources than the memory free space system.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.File.Freespace.UseBTreeSystem();
```

FreeSpaceConfiguration.cs: Use BTree system

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.File.Freespace.UseBTreeSystem()
```

FreeSpaceConfiguration.vb: Use BTree system

Freespace Filler

When you delete a object in db4o the storage it consumed isn't deleted. Instead only the storage space is marked as free and can be reused. Therefore it's possible to read also the content of deleted objects. If you want to avoid that, you can specify a free-space filler. This filler is responsible to overwrite the free-space.

Note that this costs performance, since additional IO operations are performed.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.File.Freespace.FreespaceFiller(new MyFreeSpaceFiller());
```

FreeSpaceConfiguration.cs: Using a freespace filler

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.File.Freespace.FreespaceFiller(New MyFreeSpaceFiller())
```

FreeSpaceConfiguration.vb: Using a freespace filler

Generate UUIDs

db4o can generate UUIDs for each stored object. These UUIDs are mainly used for replication together with commit timestamps. Of course it can be used also for other purposes.

Enable UUIDs for all objects.

You can enable UUIDs for all objects. Set the global scope on the UUID setting.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.File.GenerateUUIDs = ConfigScope.Globally;
```

FileConfiguration.cs: Enable db4o uuids globally

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.File.GenerateUUIDs = ConfigScope.Globally
```

FileConfiguration.vb: Enable db4o uuids globally

Enable UUIDs for certain classes

You can also enable uuids only for certain classes:

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.File.GenerateUUIDs = ConfigScope.Individually;
configuration.Common.ObjectClass(typeof (SpecialClass)).GenerateUUIDs(true);
```

FileConfiguration.cs: Enable db4o uuids for certain classes

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.File.GenerateUUIDs = ConfigScope.Individually
configuration.Common.ObjectClass(GetType(SpecialClass)).GenerateUUIDs(True)
```

FileConfiguration.vb: Enable db4o uuids for certain classes

Generate Commit Timestamps

db4o can store transaction timestamps. Those timestamps to compare and check if an objects has been changed. These commit timestamps are mainly used for replication together with UUIDs.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.File.GenerateCommitTimestamps = true;
```

FileConfiguration.cs: Enable db4o commit timestamps

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.File.GenerateCommitTimestamps = True
```

FileConfiguration.vb: Enable db4o commit timestamps

Lock Database File

You can disable the database lock. This is useful for two some scenarious. For example accessing a pure-readonly database at the same time. Or for increasing the performance on some small embedded devices.

However this risks the database integrity. **Never access the database file at the same time with multiple object containers!**

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.File.LockDatabaseFile = false;
```

FileConfiguration.cs: Disable the database file lock

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.File.LockDatabaseFile = False
```

FileConfiguration.vb: Disable the database file lock

Read Only

You can set db4o to a read only mode. In this mode db4o won't do any changes to the database file. This mode is optimal for reading the database without doing some accidental changes to it.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.File.ReadOnly = true;
```

FileConfiguration.cs: Set read only mode

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.File.ReadOnly = True
```

FileConfiguration.vb: Set read only mode

Recovery Mode

This option only makes sense when you try to recover a corrupted database.

Turns recovery mode on and off. Recovery mode can be used to try to retrieve as much as possible out of an corrupted database. In recovery mode internal checks are more relaxed. Null or invalid objects may be returned instead of throwing exceptions. Use this method with care as a last resort to get data out of a corrupted database.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.File.RecoveryMode = true;
```

FileConfiguration.cs: Enable recovery mode to open a corrupted database

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.File.RecoveryMode = True
```

FileConfiguration.vb: Enable recovery mode to open a corrupted database

Reserve Storage Space

Reserves a number of bytes in database files. Without this setting storage space will be allocated continuously as necessary.

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. This will result in less read/write head movement on disk based storage. 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.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.File.ReserveStorageSpace = 1024 * 1024;
```

FileConfiguration.cs: Reserve storage space

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.File.ReserveStorageSpace = 1024 * 1024
```

FileConfiguration.vb: Reserve storage space

Storage

db4o allows a user to configure the IO storage mechanism to be used. Currently db4o provides the following mechanisms:

- FileStorage: Raw file access.
- CachingStorage: A caching layer, used on top of other storages.
- MemoryStorage and PagingMemoryStorage: For in memory databases.
- NonFlushingStorage: A storage which disables the costly flush operation
- The IsolatedStorageStorage for Silverlight

It is also possible to create your own custom mechanism by implementing Storage interface. Possible use cases for a custom IO mechanism:

- Improved performance with a native library.
- Mirrored write to 2 files.
- Encryption of the database file.
- Read-on-write fail-safety control.

FileStorage

FileStorage is the base storage mechanism, providing the functionality of file access. The benefit of using FileStorage directly is in decreased memory consumption.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
IStorage fileStorage = new FileStorage();
configuration.File.Storage = fileStorage;
IObjectContainer container = Db4oEmbedded.OpenFile(configuration, "database.db4o");
```

IOConfigurationExamples.cs: Using the pure file storage

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
Dim fileStorage As IStorage = New FileStorage()
configuration.File.Storage = fileStorage
Dim container As IObjectContainer = Db4oEmbedded.OpenFile(configuration, "database.db4o")
```

IOConfigurationExamples.vb: Using the pure file storage

Without cache, the file storage is significantly slower than with cache. Therefore this storage is normally used as underlying storage for other purposes. Typically it is used together with a CachingStorage on top of it:

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
IStorage fileStorage = new FileStorage();
IStorage cachingStorage = new CachingStorage(fileStorage, 128, 1024);
configuration.File.Storage = cachingStorage;
IObjectContainer container = Db4oEmbedded.OpenFile(configuration, "database.db4o");
```

IOConfigurationExamples.cs: Using a caching storage

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
Dim fileStorage As IStorage = New FileStorage()
Dim cachingStorage As IStorage = New CachingStorage(fileStorage, 128, 1024)
configuration.File.Storage = cachingStorage
Dim container As IObjectContainer = Db4oEmbedded.OpenFile(configuration, "database.db4o")
```

IOConfigurationExamples.vb: Using a caching storage

Memory Storage

The MemoryStorage allows you to create and use a db4o database fully in RAM. This strategy eliminates long disk access times and makes db4o much faster.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
MemoryStorage memory = new MemoryStorage();
configuration.File.Storage = memory;
IObjectContainer container = Db4oEmbedded.OpenFile(configuration, "database.db4o");
```

IOConfigurationExamples.cs: Using memory-storage

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
Dim memory As New MemoryStorage()
configuration.File.Storage = memory
```

IOConfigurationExamples.vb: Using memory-storage

MemoryStorage can be created without any additional parameters passed to the constructor. In this case default configuration values will be used.

PagingMemoryStorage

The regular MemoryStorage implementation keeps all the content in a single byte-array. However this brings some issues. When the database outgrows the array-size, a new, larger array is created and the content is copied over. This can be quite slow. Also can cause this a out of memory exception, because during the copying these two large arrays are present. Also, on some runtimes large objects are treated different by the garbage-collector and are less often collected.

To avoid all this issues, the PagingMemoryStorage uses multiple, small arrays to keep the database in memory. When the database outgrows the storage, only such a smaller arrays needs to be allocated. The old content stays in the existing arrays. No coping is required.

However managing these arrays cost some small overhead. But for lots of cases, the PagingMemoryStorage is the better choice.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
PagingMemoryStorage memory = new PagingMemoryStorage();
configuration.File.Storage = memory;
IObjectContainer container = Db4oEmbedded.OpenFile(configuration, "database.db4o");
```

IOConfigurationExamples.cs: Using paging memory-storage

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
Dim memory As New PagingMemoryStorage()
configuration.File.Storage = memory
Dim container As IObjectContainer = Db4oEmbedded.OpenFile(configuration, "database.db4o")
```

IOConfigurationExamples.vb: Using paging memory-storage

Growth Strategy for MemoryStorage

Growth strategy defines how the database storage (reserved disk or memory space) will grow when the current space is not enough anymore.

DoublingGrowthStrategy - default setting. When the size of the database is not enough, the reserved size will be doubled.

ConstantGrowthStrategy - a configured amount of bytes will be added to the existing size when necessary.

```
IGrowthStrategy growStrategy = new ConstantGrowthStrategy(100);
MemoryStorage memory = new MemoryStorage(growStrategy);
configuration.File.Storage = memory;
```

IOConfigurationExamples.cs: Using memory-storage with constant grow strategy

```
Dim growStrategy As IGrowthStrategy = New ConstantGrowthStrategy(100)
Dim memory As New MemoryStorage(growStrategy)
configuration.File.Storage = memory
```

IOConfigurationExamples.vb: Using memory-storage with constant grow strategy

MemoryBin

Each memory storage can contain a collection of memory bins, which are actually just names memory storages. You can reuse the MemoryBin created earlier for this MemoryStorage. MemoryBins are identified by their URI, i.e. when an object container is opened with:

.NET:

```
Db4oEmbedded.OpenFile(embeddedConfiguration, "myEmbeddedDb.db4o");
```

A MemoryBin with URI = "myEmbeddedDb.db4o" will be used. If this memory bin does not exist in the storage when the container is opened, a new MemoryBin will be created and associated with this URI.

When you pass the same memory storage to multiple object containers these containers can access to the same in memory file when they are using the same name.

More Reading:

- Storing MemoryBin Data

Storing MemoryBin Data

You can use db4o backup functionality to backup your in memory container. See "Backup" on page 167

```
container.Ext().Backup(new FileStorage(), "advanced-backup.db4o");
```

BackupExample.cs: Store a backup with storage

```
container.Ext().Backup(New FileStorage(), "advanced-backup.db4o")
```

BackupExample.vb: Store a backup with storage

CachingStorage

The CachingStorage is db4o's default storage. The default implementation uses the LRU/Q2 caching mechanism to reduce disk access times and to improve performance. The cache is characterized by the amount of pages that can be utilized and the page size. The multiplication of these two parameters gives the maximum cache size that can be used.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
IStorage fileStorage = new FileStorage();
IStorage cachingStorage = new CachingStorage(fileStorage, 128, 1024);
configuration.File.Storage = cachingStorage;
IObjectContainer container = Db4oEmbedded.OpenFile(configuration, "database.db4o");
```

IOConfigurationExamples.cs: Using a caching storage

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
Dim fileStorage As IStorage = New FileStorage()
Dim cachingStorage As IStorage = New CachingStorage(fileStorage, 128, 1024)
configuration.File.Storage = cachingStorage
Dim container As IObjectContainer = Db4oEmbedded.OpenFile(configuration, "database.db4o")
```

IOConfigurationExamples.vb: Using a caching storage

Page Count and Page Size

The CachingStorage takes two parameters, the page count and the page size. Bigger page size means faster handling of information as there is no need to switch between pages for longer. On the other hand a bigger page size will mean higher memory consumption, as memory will be reserved for the whole page size, when the page is needed. Modify these values and run performance tests to find the best performance/memory consumption combination for your system. The default values are the following:

Page count = 64 Page size = 1024KB

This gives a total of 64 KB of cache memory.

Caching Type

By default db4o uses a LRU/Q2 caching strategy. You can more about the LRU/Q2 cache on the Internet or you can look for the concrete implementation in db4o source code: LRU2QCache, LRU2QXCache and LRUCache. The LRU2QCache is a simplified implementation of the 2 Queue algorithm described here:http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1..34.2641

It's possible to exchange the cache-type. Inherit from the CachingStorage-class and overwrite the new cache method. There you return you're caching implementation. For example use a simple LRU-Cache.

```
public class LRUCachingStorage : CachingStorage {
    private readonly int pageCount;

    public LRUCachingStorage(IStorage storage):base(storage) {
        this.pageCount = 128;
    }

    public LRUCachingStorage(IStorage storage, int pageCount, int pageSize)
        : base(storage, pageCount, pageSize)
    {
        this.pageCount = pageCount;
    }

    protected override ICache4 NewCache()
    {
        return CacheFactory.NewLRUCache(pageCount);
    }
}
```

LRUCachingStorage.cs: Exchange the cache-implementation

```
Public Class LRUCachingStorage
    Inherits CachingStorage
    Private ReadOnly pageCount As Integer

    Public Sub New(ByVal storage As IStorage)
        MyBase.New(storage)
        Me.pageCount = 128
    End Sub

    Public Sub New(ByVal storage As IStorage, ByVal pageCount As Integer, ByVal pageSize As Integer)
        MyBase.New(storage, pageCount, pageSize)
        Me.pageCount = pageCount
    End Sub

    Protected Overrides Function NewCache() As ICache4
        Return CacheFactory.NewLRUCache(pageCount)
    End Function
End Class
```

LRUCachingStorage.vb: Exchange the cache-implementation

Cache Configuration Example

The following example helps to see the effect of cache by modifying the cache size. Note, that this example is not a good illustration of the LRU cache algorithm, and only shows the effect of simple caching of the most recent data.

For the test we will use the following 2 classes: Item and ItemStore, holding a collection of items:

```
CachingStorageExample.cs: Item
private class Item {
    int id;
    String name;

    public Item(int id, String name) {
      this.id = id;
      this.name = name;
    }


    public override string ToString() {
      return String.Format("{0} [{1}]", name, id);
    }
  }
```

```
CachingStorageExample.cs: ItemStore
private class ItemStore {
    List<Item> items;

    public ItemStore() {
      this.items = new List<Item>();
    }

    public void AddItem(Item item) {
      this.items.Add(item);
    }
  }
```

```
CachingStorageExample.vb: Item
Private Class Item
            Private id As Integer
            Private name As [String]

            Public Sub  New(ByVal id As Integer, ByVal name As [String])
                Me.id = id
                Me.name = name
            End Sub


            Public Overloads Overrides Function ToString() As String
                Return [String].Format("{0} [{1}]", name, id)
            End Function
        End Class
```

```
CachingStorageExample.vb: ItemStore
Private Class ItemStore
            Private items As List(Of Item)

            Public Sub New()
                Me.items = New List(Of Item)()
            End Sub

            Public Sub AddItem(ByVal item As Item)
                Me.items.Add(item)
            End Sub
        End Class
```

The following methods will be used to fill in and query the database:

```
CachingStorageExample.cs: CreateDatabase
private static void CreateDatabase(IEmbeddedConfiguration config)  {
    File.Delete(Db4oFileName);

      long startTime = DateTime.Now.Ticks;
    IObjectContainer container = Db4oEmbedded.OpenFile(config, Db4oFileName);
    try  {
      ItemStore itemStore = new ItemStore();
      for (int i = 0; i < ObjectCount; i++)  {
        itemStore.AddItem(new Item(i, "title" + i));
      }
      container.Store(itemStore);
      Item item  = (Item) container.QueryByExample(
          new Item(1, "title1"))[0];
      System.Console.WriteLine(item);
    } finally  {
      container.Close();
    }
    System.Console.WriteLine(String.Format("Time to create a database: {0} ms",
            (DateTime.Now.Ticks - startTime)/TimeSpan.TicksPerMillisecond));
  }
```

```
CachingStorageExample.cs: QueryDatabase
private static void QueryDatabase(IEmbeddedConfiguration config)  {

    IObjectContainer container = Db4oEmbedded
        .OpenFile(config, Db4oFileName);
    try  {
      List<Item> temp = new List<Item>();
        long startTime = DateTime.Now.Ticks;
      IQuery q = container.Query();
      q.Constrain(typeof(Item));
      q.Descend("id").Constrain(1).Greater();
      IObjectSet result = q.Execute();
      foreach (Item i in result) {
        temp.Add(i);
      }
      System.Console.WriteLine(String.Format("Time to get an objects first time: {0} ms",
          (DateTime.Now.Ticks - startTime)/TimeSpan.TicksPerMillisecond));
      //
      temp = new List<Item>();
        startTime = DateTime.Now.Ticks;
      foreach (Item i in result) {
        temp.Add(i);
      }
      System.Console.WriteLine(String.Format("Time to get an objects second time: {0} ms",
          (DateTime.Now.Ticks - startTime)/TimeSpan.TicksPerMillisecond));
    } finally  {
      container.Close();
    }
  }
```

```
CachingStorageExample.vb: CreateDatabase
Private Shared Sub CreateDatabase(ByVal config As IEmbeddedConfiguration)
            File.Delete(Db4oFileName)

            Dim startTime As Long = DateTime.Now.Ticks
            Dim container As IObjectContainer = _
Db4oEmbedded.OpenFile(config, Db4oFileName)
            Try
                Dim itemStore As New ItemStore()
                For i As Integer = 0 To ObjectCount - 1
                    itemStore.AddItem(New Item(i, "title" & i))
                Next
                container.Store(itemStore)
                Dim item As Item = _
DirectCast(container.QueryByExample(New Item(1, "title1"))(0), Item)
```

```
                System.Console.WriteLine(item)
            Finally
                container.Close()
            End Try
            System.Console.WriteLine([String].Format("Time to create a database: {0} ms", _
(DateTime.Now.Ticks - startTime) / TimeSpan.TicksPerMillisecond))
        End Sub
```

```
CachingStorageExample.vb: QueryDatabase
Private Shared Sub QueryDatabase(ByVal config As IEmbeddedConfiguration)

            Dim container As IObjectContainer = _
Db4oEmbedded.OpenFile(config, Db4oFileName)
            Try
                Dim temp As New List(Of Item)()
                Dim startTime As Long = DateTime.Now.Ticks
                Dim q As IQuery = container.Query()
                q.Constrain(GetType(Item))
                q.Descend("id").Constrain(1).Greater()
                Dim result As IObjectSet = q.Execute()
                For Each i As Item In result
                    temp.Add(i)
                Next
                System.Console.WriteLine([String].Format( _
"Time to get an objects first time: {0} ms", _
(DateTime.Now.Ticks - startTime) / TimeSpan.TicksPerMillisecond))
                '
                temp = New List(Of Item)()
                startTime = DateTime.Now.Ticks
                For Each i As Item In result
                    temp.Add(i)
                Next
                System.Console.WriteLine([String].Format(_
"Time to get an objects second time: {0} ms", _
(DateTime.Now.Ticks - startTime) / TimeSpan.TicksPerMillisecond))
            Finally
                container.Close()
            End Try
        End Sub
```

In general you should see the benefits of using cache with the majority of db4o operations. However, the effect is most obvious when a large amount of database information should be accessed fast (query) and this information is not loaded in the applications hash memory. We are trying to reproduce this situation in the second method querying database. If the amount of items in the ItemStore collection is 10000, the database size will be around 900 KB. With the default cache size, i.e. page count multiplied by page size (64 * 1024), this will mean that the whole collection won't fit into the cache (It can still fit into your hash memory, so you may want to decrease the available hash memory or increase the size of the collection to see the effect.).

We will use the following configuration methods to compare the default cache allocation and the custom one:

```
CachingStorageExample.cs: GetConfig
private static IEmbeddedConfiguration GetConfig()  {
    IEmbeddedConfiguration config = Db4oEmbedded.NewConfiguration();
    config.File.Storage = new CachingStorage(new FileStorage(), PageCount, PageSize);
    return config;
  }
```

```
CachingStorageExample.cs: GetDefaultConfig
private static IEmbeddedConfiguration GetDefaultConfig()  {
    IEmbeddedConfiguration config = Db4oEmbedded.NewConfiguration();
    return config;
  }
```

```
CachingStorageExample.vb: GetConfig
Private Shared Function GetConfig() As IEmbeddedConfiguration
            Dim config As IEmbeddedConfiguration = _
Db4oEmbedded.NewConfiguration()
            config.File.Storage = New _
CachingStorage(New FileStorage(), PageCount, PageSize)
            Return config
        End Function
```

```
CachingStorageExample.vb: GetDefaultConfig
Private Shared Function  GetDefaultConfig() As IEmbeddedConfiguration
            Dim config As IEmbeddedConfiguration = _
Db4oEmbedded.NewConfiguration()
            Return config
        End Function
```

Using custom page count = 1024 will make the cache size enough to fit the whole collection, and you should see some performance improvement browsing the retrieved collection second time (Unless it all fits into your hash memory in any case). Please, also try decreasing page size and count to see the opposite effect.

NonFlushingStorage

NonFlushingStorage is a special IO Storage, which can be used to improve commit performance. Committing is a complex operation and requires flushing to the hard drive after each stage of commit. This is necessary as most operating system try to avoid the overhead of disk access by caching disk write data and only flushing the resulting changes to the disk. In the case of db4o commit it would mean that the physical write of some commit stages will be partially skipped and the data will be irreversibly lost.

However, physical access to the hard drive is a time-consuming operation and may considerably affect the performance. That is where NonFlushingStorage comes in: it allows the operating system to keep commit data in cache and do the physical writes in a most performant order. This may sound very nice, but in fact a system shutdown while the commit data is still in cache will lead to the database corruption.

The following example shows how to use the NonFlushingStorage. You can run it and see the performance improvement on commit stage.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
IStorage fileStorage = new FileStorage();
// the non-flushing storage improves performance, but risks database corruption.
IStorage cachingStorage = new NonFlushingStorage(fileStorage);
configuration.File.Storage = cachingStorage;
IObjectContainer container = Db4oEmbedded.OpenFile(configuration, "database.db4o");
```

IOConfigurationExamples.cs: Using the non-flushing storage

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
Dim fileStorage As IStorage = New FileStorage()
' the non-flushing storage improves performance, but risks database corruption.
Dim cachingStorage As IStorage = New NonFlushingStorage(FileStorage)
configuration.File.Storage = cachingStorage
Dim container As IObjectContainer = Db4oEmbedded.OpenFile(Configuration, "database.db4o")
```

IOConfigurationExamples.vb: Using the non-flushing storage

Please, remember, that NonFlushingStorage is potentially dangerous and any unexpected system shutdown may corrupt your database. Use with caution and avoid using in production environments.

Pluggable Storage

Pluggable nature of db4o Storage adapters allows you to implement any persistent storage behind the database engine. It can be memory, native IO, encrypted storage, mirrored storage etc.

The implementation is guided by 2 interfaces:

.NET:

`IStorage` and `IBin`

Storage/IStorage interface defines the presence of this particular storage implementation and Bin/IBin implements actual physical access to the information stored. To simplify the implementation db4o provides StorageDecorator and BinDecorator classes with the base functionality, that can be extended/overridden.

To sort out the details of the implementation let's look at an example.

Logging Storage

In this example we will implement a simple file base storage, which will log messages about each IO operation. In the implementation you can see that most of the functionality is derived from StorageDecorator and BinDecorator classes with additional logging added:

```
LoggingStorage.cs
/**//* Copyright (C) 2004 - 2009  Versant Corporation http://www.versant.com */
using System;
using Db4objects.Db4o.IO;

namespace Db4odoc.Storage
 {
    class LoggingStorage : StorageDecorator
     {
        // delegate to a normal file storage
        public LoggingStorage()
            : base(new FileStorage())
         {

        }

        // use submitted storage as a delegate
        public LoggingStorage(IStorage storage)
            : base(storage)
         {

        }

        /**//**
         * opens a Bin for the given URI.
         */
        public override IBin Open(BinConfiguration config)
         {
            IBin storage = base.Open(config);
            if (config.ReadOnly())
             {
                return new ReadOnlyBin(new LoggingBin(storage));
            }
            return new LoggingBin(storage);
        }

        /**//// <summary>
        /// LoggingBin implementation. Allows to log information
      /// for each IO operation
        /// </summary>
        class LoggingBin : BinDecorator
         {

            public LoggingBin(IBin bin)
                : base(bin)
             {

            }

            // delegate to the base class and log a message
            public override void Close()
             {
                base.Close();
                System.Console.WriteLine(string.Format
("{0} LoggingStorage: File closed", DateTime.Now.ToString()));
            }

            // log a message, then delegate
            public override int Read(long pos, byte[] buffer, int length)
             {
                System.Console.WriteLine(
string.Format("{0} LoggingStorage: Reading {1} bytes and {2} position",
DateTime.Now.ToString(), length, pos));
                return base.Read(pos, buffer, length);

            }

            // log a message, then delegate
            public override void Write(long pos, byte[] buffer, int length)
             {
                System.Console.WriteLine(
string.Format("{0} LoggingStorage: Writing {1} bytes and {2} position",
DateTime.Now.ToString(), length, pos));
                base.Write(pos, buffer, length);
            }

            // log a message, then delegate
            public override void Sync()
             {
                System.Console.WriteLine(
string.Format("{0} LoggingStorage: Syncing", DateTime.Now.ToString()));
                base.Sync();
            }

        }
    }
}
```

```
LoggingStorage.vb
' Copyright (C) 2004 - 2009  Versant Corporation http://www.versant.com

Imports System
Imports Db4objects.Db4o.IO
```

```
Namespace Db4odoc.Storage
    Class LoggingStorage
        Inherits StorageDecorator
        Public Sub New()
            ' delegate to a normal file storage
            MyBase.New(New FileStorage())

        End Sub

        Public Sub New(ByVal storage As IStorage)
            ' use submitted storage as a delegate
            MyBase.New(storage)

        End Sub

        '
        ' opens a logging bin for the given URI.
        '
        Public Overloads Overrides Function Open(ByVal config _
As BinConfiguration) As IBin
            Dim storage As IBin = MyBase.Open(config)
            If config.[ReadOnly]() Then
                Return New ReadOnlyBin(New LoggingBin(storage))
            End If
            Return New LoggingBin(storage)
        End Function

        ' LoggingBin implementation. Allows to log information
        ' for each IO operation
        Private Class LoggingBin
            Inherits BinDecorator

            Public Sub New(ByVal bin As IBin)
                MyBase.New(bin)

            End Sub

            ' delegate to the base class and log a message
            Public Overloads Overrides Sub Close()
                MyBase.Close()
                System.Console.WriteLine( _
String.Format("{0} LoggingStorage: File closed", DateTime.Now.ToString()))
            End Sub

            ' log a message, then delegate
            Public Overloads Overrides Function Read(ByVal _
pos As Long, ByVal buffer As Byte(), _
ByVal length As Integer) As Integer
                System.Console.WriteLine( _
String.Format("{0} LoggingStorage: Reading {1} bytes and {2} position", _
DateTime.Now.ToString(), length, pos))

                Return MyBase.Read(pos, buffer, length)
            End Function

            ' log a message, then delegate
            Public Overloads Overrides Sub Write(ByVal pos As Long, _
ByVal buffer As Byte(), ByVal length As Integer)
```

```
                System.Console.WriteLine(_
String.Format("{0} LoggingStorage: Writing {1} bytes and {2} position", _
DateTime.Now.ToString(), length, pos))
                MyBase.Write(pos, buffer, length)
            End Sub

            ' log a message, then delegate
            Public Overloads Overrides Sub Sync()
                System.Console.WriteLine( _
String.Format("{0} LoggingStorage: Syncing", DateTime.Now.ToString()))
                MyBase.Sync()
            End Sub

        End Class
    End Class
End Namespace
```

Use the LoggingStorage implementation with the following code (you can find a working example if you download LoggingStorage class).

C#:

```
config.File.Storage(new LoggingStorage());
```

VB.NET:

```
config.File.Storage(New LoggingStorage())
```

Isolated Storage For Silverlight

In order to support Silverlight, db4o brings a IsolatedStorage-storage. This storage uses the Silverlight Isolated Storage to store the database on the client.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.File.Storage = new IsolatedStorageStorage();

IObjectContainer container = Db4oEmbedded.OpenFile(configuration, "database.db4o");
```

IOExamples.cs: use the isolated storage on silverlight

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.File.Storage = New IsolatedStorageStorage()

Dim container As IObjectContainer = Db4oEmbedded.OpenFile(configuration, "database.db4o")
```

IOExamples.vb: use the isolated storage on silverlight

Client Configuration

The client-configuration applies only to the db4o client. All the client specific configuration is directly on the client configuration interface.

You can also configure the common-and the network-settings for a client.

Overview

Here's a overview over all client-settings which you can change:

MessageSender: Gives access to the message-sender for sending messages to the server.
PrefetchDepth: Configures how depth the object-graph is prefetched from the server.

PrefetchIDCount:Configures how many object ids are prefetched for new objects from the server.
PrefetchObjectCount: Configures how many objects are prefetched from the server.
PrefetchSlotCacheSize: Configures the slot-cache size.
TimeoutClientSocket: Configures the connection timeout.

Message Sender

This gives you access to the message sender, which allows you to send messages to the server. See "Messaging" on page 159

Prefetch Depth

In a client server scenario, network latency is one of the biggest performance concerns. Instead of making lots of little requests, it's more to do bulk operations. A good way is to prefetch objects which are maybe required later.

The prefect depth influences to which depth the object-graph is loaded from the server when query for objects. Prefetched objects avoid additional roundtrip's to the server for getting the data. However more data needs to be sent to the clients.

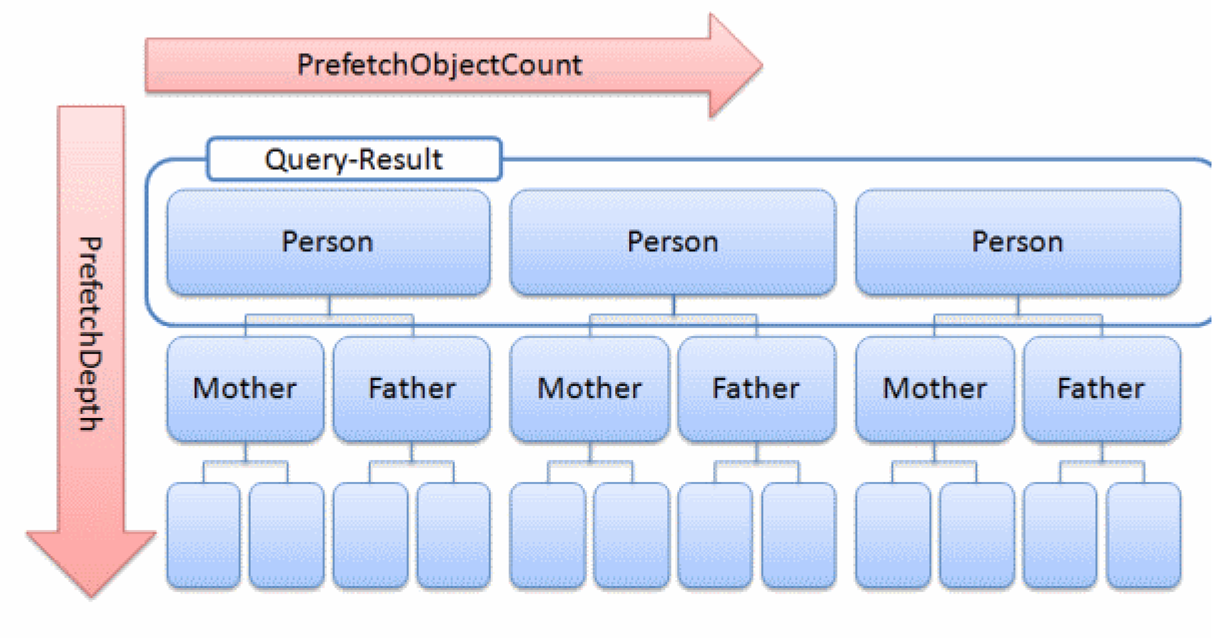
```
IClientConfiguration configuration = Db4oClientServer.NewClientConfiguration();
configuration.PrefetchDepth = 5;
```

ClientConfigurationExamples.cs: Configure the prefetch depth

```
Dim configuration As IClientConfiguration = Db4oClientServer.NewClientConfiguration()
configuration.PrefetchDepth = 5
```

ClientConfigurationExamples.vb: Configure the prefetch depth

The prefetch depth and the prefetch object count are closely related to each other. The prefetch object count configures how many objects are prefetched from a query-result. The prefetch-depth configures how deep the object-graph is fetched.



Prefetch Object Count

In a client server scenario, network latency is one of the biggest performance concerns. Instead of making lots of little requests, it's more to do bulk operations. A good way is to prefetch objects which are maybe required later.

The prefect count configures how many objects of a query-result are loaded from the server. Prefetched objects avoid additional roundtrip's to the server for getting the data. However more data needs to be sent to the clients.

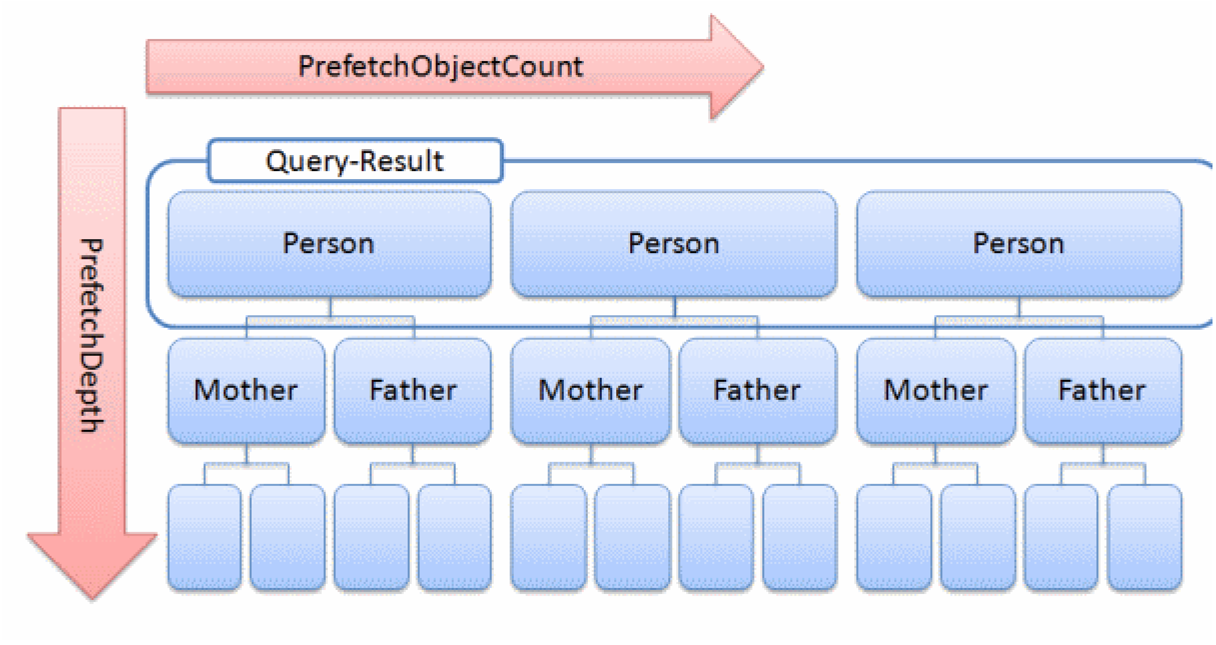
```
IClientConfiguration configuration = Db4oClientServer.NewClientConfiguration();
configuration.PrefetchObjectCount = 500;
```

ClientConfigurationExamples.cs: Configure the prefetch object count

```
Dim configuration As IClientConfiguration = Db4oClientServer.NewClientConfiguration()
configuration.PrefetchObjectCount = 500
```

ClientConfigurationExamples.vb: Configure the prefetch object count

The prefetch depth and the prefetch object count are closely related to each other. The prefetch object count configures how many objects are prefetched from a query-result. The prefetch-depth configures how deep the object-graph is fetched.



Prefetch Slot Cache Size

Configures how big the cache for prefetched data is. A larger cache can improve performance, but consumes more memory.

```
IClientConfiguration configuration = Db4oClientServer.NewClientConfiguration();
configuration.PrefetchSlotCacheSize = 1024;
```

ClientConfigurationExamples.cs: Configure the slot cache

```
Dim configuration As IClientConfiguration = Db4oClientServer.NewClientConfiguration()
configuration.PrefetchSlotCacheSize = 1024
```

ClientConfigurationExamples.vb: Configure the slot cache

Prefetching IDs For New Objects

Sets the number of IDs to be pre-allocated new objects created on the client. This avoids requests to allocate new ids when new objects are stored on a client.

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.

When the PrefetchIDCount is one, the client will have to connect to the server for each new objects created. On the other hand,, when the PrefetchIDCount is bigger than the amount of new objects the database will keep unnecessary preallocated ids.

The default PrefetchIDCount is 10.

Timeout Client Socket

Configure how long it takes, before an inactive connection is timed out.

```
IClientConfiguration configuration = Db4oClientServer.NewClientConfiguration();
configuration.TimeoutClientSocket = (1 * 60 * 1000);
```

ClientConfigurationExamples.cs: Configure the timeout

```
Dim configuration As IClientConfiguration = Db4oClientServer.NewClientConfiguration()
configuration.TimeoutClientSocket = (1 * 60 * 1000)
```

ClientConfigurationExamples.vb: Configure the timeout

Networking Configuration

The networking-configuration apply to the client- and the server-mode of db4o. All the networking configuration is accessible via the networking-property on the configuration-object.

Note that the networking configuration should be the same on the server and all clients.

Overview

Here's a overview over all networking configuration-settings which you can change:

- **BatchMessages**: Enable/Disable batch messages.
- **MessageRecipient**: Set the message receiver.
- **ClientServerFactory**: Replace the client-server factory.
- **MaxBatchQueueSize**: Set the maximum size of the batch queue.
- **SingleThreadedClient**: Set the client to single threaded.
- **SocketFactory**: Set the socket-factory.

Batch Mode

Batch mode allows to increase the performance by reducing client/server communication. It's activated by default.

```
IClientConfiguration configuration = Db4oClientServer.NewClientConfiguration();
configuration.Networking.BatchMessages = true;
```

NetworkConfigurationExample.cs: enable or disable batch mode

```
Dim configuration As IClientConfiguration = Db4oClientServer.NewClientConfiguration()
configuration.Networking.BatchMessages = True
```

NetworkConfigurationExample.vb: enable or disable batch mode

db4o communicates with the server by means of messaging. Without batch mode db4o sends a message for each operation and waits for the response. This might be quite inefficient when there are many small messages to be sent (like bulk inserts, updates, deletes). The network communication becomes a bottleneck. Batch messaging mode solves this problem by caching the messages and sending them only when required.

Advantages

- Reduced network load.
- Increased performance for bulk operations.

Disadvantages

- Increased memory consumption on both the client and the server. See "Max Batch Queue Size" on page 143

Client Server Factory

Allows you to change the behavior how a client or a server is created.

This setting should be set to the same value on the server and the clients.

```
IClientConfiguration configuration = Db4oClientServer.NewClientConfiguration();
configuration.Networking.ClientServerFactory = new StandardClientServerFactory();
```

NetworkConfigurationExample.cs: exchange the way a client or server is created

```
Dim configuration As IClientConfiguration = Db4oClientServer.NewClientConfiguration()
configuration.Networking.ClientServerFactory = New StandardClientServerFactory()
```

NetworkConfigurationExample.vb: exchange the way a client or server is created

Message Recipient

Register a message recipient for the this object container / server. See "Messaging" on page 159

Max Batch Queue Size

To avoid unnecessary network traffic, the client and server can queue up operations, which don't need to be executed immediately. As soon as a message need to be send, all queued messages are send as well. You can configure the maximum of this message queue.

A larger size allows to queue up more messages and avoid unnecessary network roundtrip's. However queued up messages consume additional memory.

```
IClientConfiguration configuration = Db4oClientServer.NewClientConfiguration();
configuration.Networking.MaxBatchQueueSize = 1024;
```

NetworkConfigurationExample.cs: change the maximum batch queue size

```
Dim configuration As IClientConfiguration = Db4oClientServer.NewClientConfiguration()
configuration.Networking.MaxBatchQueueSize = 1024
```

NetworkConfigurationExample.vb: change the maximum batch queue size

Single Threaded Client

You can configure the client to be single-threaded. When you enable this option, the client doesn't use background threads to handle the client-server communication.

```
IClientConfiguration configuration = Db4oClientServer.NewClientConfiguration();
configuration.Networking.SingleThreadedClient = true;
```

NetworkConfigurationExample.cs: single threaded client

```
Dim configuration As IClientConfiguration = Db4oClientServer.NewClientConfiguration()
configuration.Networking.SingleThreadedClient = True
```

NetworkConfigurationExample.vb: single threaded client

Advantage

On some smaller embedded systems reducing the running threads improves the performance significantly.

Disadvantage

Since all operations run in a single thread, the operations may take longer. Additionally you cannot receive messages and cannot use commit-callbacks on a single-threaded client.

Pluggable Sockets

db4o allows to customize client-server communication by using pluggable socket implementations.

```
IClientConfiguration configuration = Db4oClientServer.NewClientConfiguration();
configuration.Networking.SocketFactory = new StandardSocket4Factory();
```

NetworkConfigurationExample.cs: Exchange the socket-factory

```
Dim configuration As IClientConfiguration = Db4oClientServer.NewClientConfiguration()
configuration.Networking.SocketFactory = New StandardSocket4Factory()
```

NetworkConfigurationExample.vb: Exchange the socket-factory

One use case for changing the socket-implementation is encryption. In fact, db4o's SSL-support uses this mechanism: See "Using SSL For Client-Server Communication" on page 162

Class Specific Configuration

Some settings are object-specific and are configured for the class. It's part of the common-configuration, which is available on the client, server and embedded-mode of db4o.

Its recommended that you use the same configuration for the client and the server.

Access the Class Configuration

The configuration for a specific class follows always the same pattern. First you specify for which type the configuration applies. You pass the type, the name as string or event an instance of the specific class to the configuration.

From the class-configuration, you also can go a level deeper to the field configuration.

Overview

Here's a overview over all common configuration-settings which you can change:

CallConstructor: Configure db4o to call constructors when instantiating objects.
CascadeOnDelete: When a object is deleted, delete also referenced objects.
CascadeOnUpdate: When a object is updated, update also referenced objects.
CascadeOnActivation: When a object is activated, activate also referenced objects.
Index: Don't index the objects of this type.
~~**EnableReplication**~~: Deprecated. Generate uuids and commit timestamps to enable replication.
GenerateUUIDs: Generate UUIDs, mainly used for replication.
~~**GenerateVersionNumbers**~~: Deprecated. Generate commit timestamps instead.
MaximumActivationDepth: Set a maximum activation-depth.
MinimumActivationDeph: Set a minimum activation-depth.
PersistStaticFields: Persist also the static fields of this type.
Rename: Rename this type. Used for refactorings.
Translate: Set a translator for this type.
StoreTransientFields: Store also transient fields.
UpdateDepth: Set the update-depth for this type.

Call Constructor

By default db4o bypassed the constructor when it loads the objects from the database. When you're class relies on the constructor to be called, for example to initialize transient state, you can enable it. You can do this also on the global configuration for all types. See "Calling Constructors" on page 113

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Person)).CallConstructor(true);
```

ObjectConfigurationExamples.cs: Call constructor

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).CallConstructor(True)
```

ObjectConfigurationExamples.vb: Call constructor

Cascade on Delete

When turned on the deletion cascades to all referred objects.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Person)).CascadeOnDelete(true);
```

ObjectConfigurationExamples.cs: When deleted, delete also all references

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).CascadeOnDelete(True)
```

ObjectConfigurationExamples.vb: When deleted, delete also all references

Cascade on Update

When turned on, the update cascades to all referred objects.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Person)).CascadeOnUpdate(true);
```

ObjectConfigurationExamples.cs: When updated, update also all references

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).CascadeOnUpdate(True)
```

ObjectConfigurationExamples.vb: When updated, update also all references

Cascade on Activate

db4o uses the concept of activation to avoid loading to much data into memory. When turned on the activation cascades for this types. This means that when an instance is activated, all referenced objects are also activated.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Person)).CascadeOnActivate(true);
```

ObjectConfigurationExamples.cs: When activated, activate also all references

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).CascadeOnActivate(True)
```

ObjectConfigurationExamples.vb: When activated, activate also all references

Disable Class Index

By default there's a index which indexes all instances of a certain type. When you never query this type and only access it by navigation, you can safely disable this index.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Person)).Indexed(false);
```

ObjectConfigurationExamples.cs: Disable class index

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).Indexed(False)
```

ObjectConfigurationExamples.vb: Disable class index

Generate UUIDs

Generate UUIDs for instances of this type. UUIDs are mainly used for replication.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Person)).GenerateUUIDs(true);
```

ObjectConfigurationExamples.cs: Generate uuids for this type

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).GenerateUUIDs(True)
```

ObjectConfigurationExamples.vb: Generate uuids for this type

Maximum Activation Depth

Sets the maximum activation depth for this type. Useful to avoid accidentally activating large objects.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Person)).MaximumActivationDepth(5);
```

ObjectConfigurationExamples.cs: Set maximum activation depth

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).MaximumActivationDepth(5)
```

ObjectConfigurationExamples.vb: Set maximum activation depth

Minimum Activation Depth

Sets the minimum activation depth for this type. Useful to load always load referenced objects.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Person)).MinimumActivationDepth(2);
```

ObjectConfigurationExamples.cs: Set minimum activation depth

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).MinimumActivationDepth(2)
```

ObjectConfigurationExamples.vb: Set minimum activation depth

Persist Static Fields

By default db4o doesn't store static fields. With this setting you force db4o to also store the value of the static fields. Note that only reference types are stored. Value-types like ints, longs etc are not persisted.

In general you shouldn't store static fields, because this can lead to unexpected behaviors. It's mostly useful for enumeration-classes. See "Storing Static Fields" on page 197

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Person)).PersistStaticFieldValues();
```

ObjectConfigurationExamples.cs: Persist also the static fields

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).PersistStaticFieldValues()
```

ObjectConfigurationExamples.vb: Persist also the static fields

Rename Class

You can rename a class. Useful when you reactor you're code. See "Refactoring and Schema Evolution" on page 208

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof(Person)).Rename("Db4oDoc.NewNamespace.NewName");
```

ObjectConfigurationExamples.cs: Rename this type

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).Rename("Db4oDoc.NewNamespace.NewName")
```

ObjectConfigurationExamples.vb: Rename this type

Set a Translator

You can specify a special translator for this type. A translator is a special way to add you're custom serialization for a type. See "Translators" on page 201

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Person)).Translate(new TSerializable());
```

ObjectConfigurationExamples.cs: Use a translator for this type

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).Translate(New TSerializable())
```

ObjectConfigurationExamples.vb: Use a translator for this type

Store Transient Fields

By default db4o doesn't store transient fields. With this setting you can force db4o to event store transient fields.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Person)).StoreTransientFields(true);
```

ObjectConfigurationExamples.cs: Store also transient fields

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).StoreTransientFields(True)
```

ObjectConfigurationExamples.vb: Store also transient fields

Update Depth

By default db4o only stores changes on the updated object, but not the changes on referenced objects. With a higher update-depth db4o will traverse along the object graph to a certain depth and update all objects. See "Update Concept" on page 71. You can also specify this globally.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Person)).UpdateDepth(2);
```

ObjectConfigurationExamples.cs: Set the update depth

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).UpdateDepth(2)
```

ObjectConfigurationExamples.vb: Set the update depth

Field Specific Configuration

Some settings are field-specific. It's part of the object-configuration, which is available on the client, server and embedded-mode of db4o.

Its recommended that you use the same configuration for the client and the server.

Access the Field Configuration

The configuration for a field follows the same pattern. First you specify for which type this configuration applies. You pass the type or the name as string. Then you navigate to the specific field by passing the field name as string.

Note that you need to specify the field-name and not the property-name, also for auto-properties.

Adding a Field Index

Index dramatically speed up queries. You should index all fields which you run queries on. See "Indexing" on page 102

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Person)).ObjectField("name").Indexed(true);
```

ObjectFieldConfigurations.cs: Index a certain field

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).ObjectField("name").Indexed(True)
```

ObjectFieldConfigurations.vb: Index a certain field

As an alternative you also can use the appropriate Attribute on the field which you want to index.

```
[Indexed]
private string zipCode;
```

ObjectFieldConfigurations.cs: Index a field

```
<Indexed()> _
Private m_zipCode As String
```

ObjectFieldConfigurations.vb: Index a field

Cascade On Activate

db4o uses the concept of activation to avoid loading to much data into memory. When this setting is turned on, the object referenced by this field is activated, when the object is activated.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Person)).ObjectField("father").CascadeOnActivate(true);
```

ObjectFieldConfigurations.cs: When activated, activate also the object referenced by this field

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).ObjectField("father").CascadeOnActivate(True)
```

ObjectFieldConfigurations.vb: When activated, activate also the object referenced by this field

Cascade On Update

When the object is updated, the object referenced by this field is also updated.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Person)).ObjectField("father").CascadeOnUpdate(true);
```

ObjectFieldConfigurations.cs: When updated, update also the object referenced by this field

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).ObjectField("father").CascadeOnUpdate(True)
```

ObjectFieldConfigurations.vb: When updated, update also the object referenced by this field

Cascade On Delete

When the object is deleted, the object referenced by this field is also deleted

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Person)).ObjectField("father").CascadeOnDelete(true);
```

ObjectFieldConfigurations.cs: When deleted, delete also the object referenced by this field

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).ObjectField("father").CascadeOnDelete(True)
```

ObjectFieldConfigurations.vb: When deleted, delete also the object referenced by this field

Rename Field

Allows you to rename this field. . See "Refactoring and Schema Evolution" on page 208

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Person)).ObjectField("name").Rename("sirname");
```

ObjectFieldConfigurations.cs: Rename this field

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).ObjectField("name").Rename("sirname")
```

ObjectFieldConfigurations.vb: Rename this field

Id System

The id-system configuration applies to the embedded- and the server-mode of db4o. There are tree different id-systems-available. All the id system configuration is accessible via the id-system-property on the configuration-object.

Note that you cannot change the id-system for an existing database. You need to defragment the database in order to change the id-system.

The id-system is responsible to mapping a object id to the physical location of an object. This mapping can have significant impact on the performance.

Stacked BTree Id-System

This setting uses a stack of two BTree's on top of an InMemoryIdSystem. This system is scalable for a large number of ids.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.IdSystem.UseStackedBTreeSystem();
```

IdSystemConfigurationExamples.cs: Use stacked B-trees for storing the ids

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.IdSystem.UseStackedBTreeSystem()
```

IdSystemConfigurationExamples.vb: Use stacked B-trees for storing the ids

Single BTree Id-System

This setting uses a single BTree on top of a in memory id system. This system works great for small databases. However it cannot scale for a large number of ids.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.IdSystem.UseSingleBTreeSystem();
```

IdSystemConfigurationExamples.cs: Use a single B-tree for storing the ids.

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.IdSystem.UseSingleBTreeSystem()
```

IdSystemConfigurationExamples.vb: Use a single B-tree for storing the ids.

In Memory Id-System

This id-system keeps all ids in memory. While accessing the ids if fast, all ids have to be written to disk on every commit. Therefore it can be used only for tiny databases.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.IdSystem.UseInMemorySystem();
```

IdSystemConfigurationExamples.cs: Use a in-memory id system

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.IdSystem.UseInMemorySystem()
```

IdSystemConfigurationExamples.vb: Use a in-memory id system

Pointer Based Id-System

This id system uses pointers to handle ids. Each id represents a pointer into the database-file. This makes the id-mapping simple. However since it's a pointer, you cannot change the location. Therefore this system leads to more fragmentation and performance degradation as the database grows.

This id system is here to ensure backward-compatibility. It's not recommended to use for new databases.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.IdSystem.UseInMemorySystem();
```

IdSystemConfigurationExamples.cs: Use a in-memory id system

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.IdSystem.UseInMemorySystem()
```

IdSystemConfigurationExamples.vb: Use a in-memory id system

Custom Id-System

It's possible to implement your own id system. You can pass an factory which creates your id-system implementation.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.IdSystem.UseCustomSystem(new CustomIdSystemFactory());
```

IdSystemConfigurationExamples.cs: use a custom id system

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.IdSystem.UseCustomSystem(New CustomIdSystemFactory())
```

IdSystemConfigurationExamples.vb: use a custom id system

Server Configuration

The server-configuration applies to the db4o-server. All the server specific configuration is directly on the client configuration interface.

You can also configure the common-, file-, id-system- and the network-settings for a server.

Server Socket Timeout

Configure how long it takes, before an inactive connection is timed out.

```
IServerConfiguration configuration = Db4oClientServer.NewServerConfiguration();
configuration.TimeoutServerSocket = (10 * 60 * 1000);
```

ServerConfigurationExamples.cs: configure the socket-timeout

```
Dim configuration As IServerConfiguration = Db4oClientServer.NewServerConfiguration()
configuration.TimeoutServerSocket = (10 * 60 * 1000)
```

ServerConfigurationExamples.vb: configure the socket-timeout

Client-Server

db4o's is an embedded database which usually runs in process. Nevertheless db4o also supports a client server setup. There's no ready to use db4o server, instead you embedded the db4o server in a regular application. This gives you full how you want to run the db4o server.

In order to use the db4o client server mode you need to include the Db4objects.Db4o.CS.dll-assembly in your project. See "Dependency Overview" on page 365

Start The Server

You start the server by creating a object server instance with the client server factory. Pass the database file path and the port to the factory. After that you need to specify a user-name and password which clients can use to login to this server. You can add multiple users by calling the grant access method multiple times.

```
using (IObjectServer server = Db4oClientServer.OpenServer("database.db4o", 8080))
{
    server.GrantAccess("user", "password");

    // Let the server run.
    LetServerRun();
}
```

ServerExample.cs: Start a db4o server

```
Using server As IObjectServer = Db4oClientServer.OpenServer("database.db4o", 8080)
    server.GrantAccess("user", "password")

    ' Let the server run.
    LetServerRun()
End Using
```

ServerExample.vb: Start a db4o server

Connect To The Server

After the server is up an running you can connect to it. Pass the URL, port, user name and password to the client server factory. This will return a client object container. After that the container is ready to use.

```
using (IObjectContainer container
    = Db4oClientServer.OpenClient("localhost", 8080, "user", "password"))
{
    // Your operations
}
```

Db4oClientExample.cs: Connect to the server

```
' Your operations
Using container As IObjectContainer = Db4oClientServer.OpenClient("localhost", 8080, "user", "password")
End Using
```

Db4oClientExample.vb: Connect to the server

Reference Cache In Client-Server Mode

db4o uses an 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.

```
Person personOnClient1 = QueryForPerson(client1);
Person personOnClient2 = QueryForPerson(client2);
Console.Write(QueryForPerson(client2).Name);

personOnClient1.Name = ("New Name");
client1.Store(personOnClient1);
client1.Commit();

// The other client still has the old data in the cache
Console.Write(QueryForPerson(client2).Name);

client2.Ext().Refresh(personOnClient2, int.MaxValue);

// After refreshing the date is visible
Console.Write(QueryForPerson(client2).Name);
```

ReferenceCacheExamples.cs: Reference cache in client server

```
Dim personOnClient1 As Person = QueryForPerson(client1)
Dim personOnClient2 As Person = QueryForPerson(client2)
Console.Write(QueryForPerson(client2).Name)

personOnClient1.Name = ("New Name")
client1.Store(personOnClient1)
client1.Commit()

' The other client still has the old data in the cache
Console.Write(QueryForPerson(client2).Name)

client2.Ext().Refresh(personOnClient2, Integer.MaxValue)

' After refreshing the date is visible
Console.Write(QueryForPerson(client2).Name)
```

ReferenceCacheExamples.vb: Reference cache in client server

There are multiple strategies to deal with this.

- If you client's are not updating the same object or it very unlikely that it happens, you don't need to take any action.
- You can refresh objects as they are updated on all clients. However this need a lot of communication between the server and client. See "Refreshing Objects" on page 154
- You can do short unit of work on the client, to minimize the chance of outdated objects.

Clean Cache For Work

Often you don't want to refresh object by object. Instead you want to work with a clean cache. You can do this by using a separate 'session' on the client. This container brings its own cache with it. So you can create a new container which a clean cache.

Note that open session on the client doesn't create a separate transaction. Instead it only creates a fresh cache. The transaction is always the same.

```
using (IObjectContainer container = client.Ext().OpenSession())
{
    // do work
}
// Start with a fresh cache:
using (IObjectContainer container = client.Ext().OpenSession())
{
    // do work
}
```

ReferenceCacheExamples.cs: Clean cache for each unit of work

```
Using container As IObjectContainer = client.Ext().OpenSession()
    ' do work
End Using
' Start with a fresh cache:
Using container As IObjectContainer = client.Ext().OpenSession()
    ' do work
End Using
```

ReferenceCacheExamples.vb: Clean cache for each unit of work

Refreshing Objects

When working with multiple client object-containers each container has its own reference cache. When you query for objects, you get always the cached objects. This means, that objects probably have been updated in the mean-time but the client still sees the old state.

For some scenarios you might need to refresh to objects to bring it up to date. There are two strategies for this. You can explicit refresh a object at any time. Or you can use callbacks to refresh object on each commit. Both methods have their advantage.

	Explicit Refreshing	Using Callbacks
Advantage	• Selective refreshing possible, which reduces network traffic. • Can decide when a refresh is required. If no refresh is required, you can save the network traffic.	• The objects are always up to date. • There's no need to explicitly refresh-calls in your data-access layer.
Disadvantage	• Partial refreshed objects may lead to a inconsistent object-graph. • Probably a lot of refresh-calls spread over the code-base.	• A lot of network traffic is required to send the committing-events and object to the clients.

Explicitly Refreshing Objects

You can refresh objects with the refresh-method on the object-container. The pass the object to refresh and the refresh-depth to the method:

```
db.Ext().Refresh(objToRefresh, int.MaxValue);
```

RefreshingObjects.cs: refresh a object

```
db.Ext().Refresh(objToRefresh, Integer.MaxValue)
```

RefreshingObjects.vb: refresh a object

Using Callbacks For Refreshing Objects

You can use the committed-event to refresh objects as soon as another client commits. Take a look at the example: See "Committed Event Example" on page 191

Embedded Client Server

db4o supports embedded containers or session container. It's a separate object-container with its own transaction and own reference cache. See "Session Containers" on page 164

Now if you're using the client-server mode for db4o, you also can create such sessions directly from the server. When you pass a 0 to the open server method, the server only supports embedded clients. With any other port you can connect with regular clients and also create embedded clients.

```
using (IObjectServer server = Db4oClientServer.OpenServer(DatabaseFileName, 0))
{
    // open the db4o-embedded client. For example at the beginning for a web-request
    using (IObjectContainer container = server.OpenClient())
    {
        // do the operations on the session-container
        container.Store(new Person("Joe"));
    }
}
```

Db4oSessions.cs: Embedded client

```
Using server As IObjectServer = Db4oClientServer.OpenServer(DatabaseFileName, 0)
    ' open the db4o-embedded client. For example at the beginning for a web-request
    Using container As IObjectContainer = server.OpenClient()
        ' do the operations on the session-container
        container.Store(New Person("Joe"))
    End Using
End Using
```

Db4oSessions.vb: Embedded client

OpenSession On A Client

You might noted that the open-session is available on any object-container. Normally this creates a session-container with its own transaction and reference cache. However on a db4o-client this is not true. There it only creates a container with new cache, but shares the transaction with the client. The only use case for this is to implement connection pooling.See "Client-Container Pooling" on page 157

Native Queries

Native Queries are optimized to SODA under the hood. Therefore they can be normally used in a client-server environment. However, there is a catch: as soon as the native query is not optimized there will be two things required on the server:

- The persistent classes definitions will be required because the objects will need to be instantiated to execute the query code.
- The native predicate delegate will be required to run on the server to do the actual job.

To meet these requirements you can keep your persistent classes and **NQ**[1] predicates in separate libraries, which will make it easy to deploy it on both the client and the server side. Otherwise you may try to express the unoptimised query in SODA, this option will make it more performant, but the disadvantage is less robust and maintainable code.

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's Object Manager.

Unfortunately in a server without persistent classes mode there are still some limitations:

- Evaluation queries won't work. This also implies that native queries which cannot be optimized don't work.
- Native queries will only work if they can be optimized
- Multidimensional arrays can not be stored.

Native Queries

Native queries are usually translated to SODA queries. When a native query can be translated to SODA, it will run fine on a server without persisted classes. However when the query cannot be optimized the server needs the native query instance and the data model which isn't available on a server without persistent classes. Therefore native queries only work when it can be optimized. You can fallback to pure SODA queries in such cases.

Events In Client Server-Mode

Events of course also work in client server mode. This topic only applies to the networked client/server mode. Embedded client/server-mode isn't affected. There the events work the same way as in the embedded-mode. See "Event Registry API" on page 182

Separate Event Registry For Each Client

Each client has its own event registry. When you register to a event on the client-event-registry, events will be fired for actions on the client. You won't receive events for actions on other clients.

Furthermore you cannot register for the delete-events, the delete events is only on the server available.

The Server Event Registry

The server has its own event registry. When you register to an event on the server registry, events will be fired for all action of the clients and the server itself. So you can monitor all operations on the server.

Note that in some events the server isn't involved. For example when a client activates some object, the server is no involved, an therefore no event is fired on the server.

Use central server object container to register for the events on the server.

[1]Native Query

```
IObjectServer server =
        Db4oClientServer.OpenServer(DatabaseFileName, PortNumber);
IEventRegistry eventsOnServer =
        EventRegistryFactory.ForObjectContainer(server.Ext().ObjectContainer());
```

EventRegistryExamples.cs: register for events on the server

```
Dim server As IObjectServer = _
        Db4oClientServer.OpenServer(DatabaseFileName, PortNumber)
Dim eventsOnServer As IEventRegistry = _
        EventRegistryFactory.ForObjectContainer(server.Ext().ObjectContainer())
```

EventRegistryExamples.vb: register for events on the server

Committed Event On All Parties

The committed event is an exception. When a client or the server commits, every client and the server will fire the committed event. This way a client can inform itself that another client has made changes to the database.

However this involves communication overhead to send the message to all the clients. Therefore you should use this event only when absolutely necessary on the clients.

Events Are Fired Asynchronous

In client-server-mode the events are fired asynchronous. This means that you event-handler is invoked in a separate thread. You need to ensure that you lock any shared data-structure you access from the event-handlers.

Client-Container Pooling

When a client connects to the server, there's overhead to establish the connection. First the regular TCP-connection needs to be established. Then the server and client exchange meta data and finally the connection is ready. If you don't need a single or a few long during connections but rather short units of work on the client it is quite inefficient to open a client connection for each unit of work. In such scenarios you should consider to pool the client-containers.

Now simply pooling the raw client container might lead to issues. Each object-container has a reference-cache. When you pool the object-container and reuse it for some other work, this cache isn't cleared. This means that you might get dirty objects from the reference cache. You want to avoid this and have a clean cache when reusing the client container.

There's a way to archive that. On the client container the open session method creates a fresh object container with a clean reference cache which is sharing the transaction with the client. With this building block you can build a proper container pool.

First you need to create client connections on demand which will be pooled.

```
IObjectContainer client = Db4oClientServer.OpenClient("localhost",
    Port, UserAndPassword, UserAndPassword);
```

ConnectionPoolExamples.cs: Open clients for the pool

```
Dim client As IObjectContainer = Db4oClientServer.OpenClient("localhost", _
             Port, UserAndPassword, UserAndPassword)
```

ConnectionPoolExamples.vb: Open clients for the pool

On a request for a object container, get a client container from the pool. Rollback the transaction on it to ensure that it is in a clean state.Then open a session container on it and use this session. The session-container ensures that the reference-cache is empty. Make sure that each client container is always used only once at any time, which means that there's always only one session-container open per client-container. The session-containers share the transaction with the client and you don't want to share transactions across multiple object containers.

```
// Obtain a client container. Either take one from the pool or allocate a new one
IObjectContainer client = ObtainClientContainer();
// Make sure that the transaction is in clean state
client.Rollback();
// Then create a session on that client container and use it for the database operations.
// The client-container is now in use. Ensure that it isn't leased twice.
IObjectContainer sessionContainer = client.Ext().OpenSession();
```

ConnectionPool.cs: Obtain a pooled container

```
' Obtain a client container. Either take one from the pool or allocate a new one
Dim client As IObjectContainer = ObtainClientContainer()
' Make sure that the transaction is in clean state
client.Rollback()
' Then create a session on that client container and use it for the database operations.
' The client-container is now in use. Ensure that it isn't leased twice.
Dim sessionContainer As IObjectContainer = client.Ext().OpenSession()
```

ConnectionPool.vb: Obtain a pooled container

Now when you're done with your operations, you can return the client to your pool of object containers. First close the session-container, which commits the changes and releases the resources. Then get the underlying client-container for that session and return it to the pool.

```
// First you need to get the underlying client for the session
IObjectContainer client = ClientForSession(session);
// then the client is ready for reuse
ReturnToThePool(client);
```

ConnectionPool.cs: Returning to pool

```
' First you need to get the underlying client for the session
Dim client As IObjectContainer = ClientForSession(session)
' then the client is ready for reuse
ReturnToThePool(client)
```

ConnectionPool.vb: Returning to pool

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.

Semaphores

db4o semaphores are named flags that can only be owned by one object container 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.

```
// Grab the lock. Specify the name and a timeout in milliseconds
bool hasLock = container.Ext().SetSemaphore("LockName", 1000);
try
{
    // you need to check the lock
    if (hasLock)
    {
        Console.WriteLine("Could get lock");
    }
    else
    {
        Console.WriteLine("Couldn't get lock");
    }
}
finally
{
    // release the lock
    container.Ext().ReleaseSemaphore("LockName");
}
```

SemaphoreExamples.cs: Grab a semaphore

```
' Grab the lock. Specify the name and a timeout in milliseconds
Dim hasLock As Boolean = container.Ext().SetSemaphore("LockName", 1000)
Try
    ' you need to check the lock
    If hasLock Then
        Console.WriteLine("Could get lock")
    Else
        Console.WriteLine("Couldn't get lock")
    End If
Finally
    ' release the lock
    container.Ext().ReleaseSemaphore("LockName")
End Try
```

SemaphoreExamples.vb: Grab a semaphore

Building Block

Semaphores are a low level building block. Usually they are not used directly. Instead you can build the abstractions you need with semaphores. For example you can build object-locking, critical sections etc with semaphores.

Messaging

In client/server it possible to send messages between a client and the server. Possible use cases could be:

- Shutting down and restarting the server.
- Triggering server backup.
- Using a customized login strategy to restrict the number of allowed client connections.
- Running special code on the server. For example batch updates.

Sending and Receiving Messages

First you need to decide on a class that you want to use as the message. Any object that is storable in db4o can be used as a message. Of course you use multiple classes for representing different messages. Let's create a dedicated class.

```
public class HelloMessage
{
    private readonly string message;

    public HelloMessage(string message)
    {
        this.message = message;
    }

    public override string ToString()
    {
        return message;
    }
}
```

MessagingExample.cs: The message class

```
Public Class HelloMessage
    Private ReadOnly message As String

    Public Sub New(ByVal message As String)
        Me.message = message
    End Sub

    Public Overrides Function ToString() As String
        Return message
    End Function
End Class
```

MessagingExample.vb: The message class

Now you need to register a handler to handle arriving messages. This can be done by configuring a message recipient on the server. Let's simply print out the arriving message. Additionally we answer to the client with another message.

```
IServerConfiguration configuration = Db4oClientServer.NewServerConfiguration();
configuration.Networking.MessageRecipient = new ServerMessageReceiver();
IObjectServer server = Db4oClientServer.OpenServer(configuration, DatabaseFile, PortNumber);
```

MessagingExample.cs: configure a message receiver for the server

```
Dim configuration As IServerConfiguration = Db4oClientServer.NewServerConfiguration()
configuration.Networking.MessageRecipient = New ServerMessageReceiver()
Dim server As IObjectServer = Db4oClientServer.OpenServer(configuration, DatabaseFile, PortNumber)
```

MessagingExample.vb: configure a message receiver for the server

```
class ServerMessageReceiver : IMessageRecipient
{
    public void ProcessMessage(IMessageContext context, object message)
    {
        Console.WriteLine("The server received a '{0}' message", message);
        // you can respond to the client
        context.Sender.Send(new HelloMessage("Hi Client!"));
    }
}
```

MessagingExample.cs: The message receiver for the server

```
Private Class ServerMessageReceiver
    Implements IMessageRecipient
    Public Sub ProcessMessage(ByVal context As IMessageContext, ByVal message As Object) _
        Implements IMessageRecipient.ProcessMessage
        Console.WriteLine("The server received a '{0}' message", message)
        ' you can respond to the client
        context.Sender.Send(New HelloMessage("Hi Client!"))
    End Sub
End Class
```

MessagingExample.vb: The message receiver for the server

The same can be done on the client. Register a handler for the received messages.

```
IClientConfiguration configuration = Db4oClientServer.NewClientConfiguration();
configuration.Networking.MessageRecipient = new ClientMessageReceiver();
```

MessagingExample.cs: configure a message receiver for a client

```
Dim configuration As IClientConfiguration = Db4oClientServer.NewClientConfiguration()
configuration.Networking.MessageRecipient = New ClientMessageReceiver()
```

MessagingExample.vb: configure a message receiver for a client

```
class ClientMessageReceiver : IMessageRecipient
{
    public void ProcessMessage(IMessageContext context, object message)
    {
        Console.WriteLine("The client received a '{0}' message",message);
    }
}
```

MessagingExample.cs: The message receiver for the client

```
Private Class ClientMessageReceiver
    Implements IMessageRecipient
    Public Sub ProcessMessage(ByVal context As IMessageContext, ByVal message As Object) _
        Implements IMessageRecipient.ProcessMessage
        Console.WriteLine("The client received a '{0}' message", message)
    End Sub
End Class
```

MessagingExample.vb: The message receiver for the client

Now on the client we can get a message sender. The message sender allows you to send a message to the server. In this example we send a hello message.

```
IMessageSender sender = configuration.MessageSender;
using (IObjectContainer container = Db4oClientServer.OpenClient(configuration, "localhost",
                                                                PortNumber, UserAndPassword,
                                                                UserAndPassword))
{
    sender.Send(new HelloMessage("Hi Server!"));
    WaitForAWhile();
}
```

MessagingExample.cs: Get the message sender and use it

```
Dim sender As IMessageSender = configuration.MessageSender
Using container As IObjectContainer = Db4oClientServer.OpenClient(configuration, _
            "localhost", PortNumber, UserAndPassword, UserAndPassword)
    sender.Send(New HelloMessage("Hi Server!"))
    WaitForAWhile()
End Using
```

MessagingExample.vb: Get the message sender and use it

Using SSL For Client-Server Communication

With the default settings db4o client-server communication is not encrypted and thus can potentially be a dangerous security hole. db4o supports SSL for client server communication. The implementation uses the pluggable socket to provide secure sockets.

The SSL communication uses the .NET security libraries to implement the secure communication. Currently only one way authentication is supported, where the server needs to have a valid certificate. Clients-certificates are currently not supported.

On the server you need to add a the server SSL support. Additionally you need to specif the server certificate. This way you can choose from where you want to get the server-certificate. Take a look at the .NET documentation for a more details.

```
IServerConfiguration configuration = Db4oClientServer.NewServerConfiguration();
// For the server you need a certificate. For example using a certificate from a file
X509Certificate2 certificate = new X509Certificate2("cert.cer");
configuration.AddConfigurationItem(new ServerSslSupport(certificate));
```

SSLExample.cs: Add SSL-support to the server

```
Dim configuration As IServerConfiguration = Db4oClientServer.NewServerConfiguration()
' For the server you need a certificate. For example using a certificate from a file
Dim certificate As New X509Certificate2("cert.cer")
configuration.AddConfigurationItem(New ServerSslSupport(certificate))
```

SSLExample.vb: Add SSL-support to the server

On the client you need to use the client SSL support. Add a callback for additional certificate verification.

```
IClientConfiguration configuration = Db4oClientServer.NewClientConfiguration();
configuration.AddConfigurationItem(new ClientSslSupport(CheckCertificate));
```

SSLExample.cs: Add SSL-support to the client

```
Dim configuration As IClientConfiguration = Db4oClientServer.NewClientConfiguration()
configuration.AddConfigurationItem(New ClientSslSupport(AddressOf CheckCertificate))
```

SSLExample.vb: Add SSL-support to the client

```
private static bool CheckCertificate(object sender, X509Certificate certificate, X509Chain chain, SslPolicyErrors
{
    // here you can check the certificates of the server and accept it if it's ok.)
    return true;
}
```

SSLExample.cs: callback for validating the certificate

```
Private Shared Function CheckCertificate(ByVal sender As Object, ByVal certificate As X509Certificate, ByVal chai
    ' here you can check the certificates of the server and accept it if it's ok.
    Return True
End Function
```

SSLExample.vb: callback for validating the certificate

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 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's are small downside to this. When a server operation takes longer than the timeout, the connection will be closed. You can configure the timeouts for the client and the server.

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 instable, lower the values, so resources for disconnected clients are freed faster.

Advanced Topics

This topic is about advanced features of db4o.

If you want to have multiple unit of work or multiple transaction in embedded mode you can take a look at session containers. See "Session Containers" on page 164

db4o support unique constrains on fields. See "Unique Constraints" on page 166

You can also do backups of your database at runtime. See "Backup" on page 167

Implementing the interfaces for transparent activation/persistence is tedious. The enhancement tools can do that job for you. And can also optimize native queries at build time. See "Enhancement Tools" on page 177

The database file can fragment over time. In order to reclaim defragmentet space in the database you need to run the defragmentation. See "Defragment" on page 172

Callback allow you to perform additional logic for different database operations. See "Callbacks" on page 182

As some point in time you will change your data model. Then you probably need to also refactor you're stored data in the database. See "Refactoring and Schema Evolution" on page 208

You can access the internal ids or generate UUIDs for objects. See "IDs and UUIDs" on page 167

It's also possible to access the meta information of all stored types in the database. See "db4o Meta-Information" on page 169. And you can access system information about the database.

db4o tried hard to make persisting objects as easy as possible. However storing a object efficient and correctly is quite tricky. Read about db4o's type handling for more information. See "Type Handling" on page 193 Furthermore db4o uses an abstraction layer to encapsulate .NET-reflection. This allows you do change how reflection behaves on your objects. See "db4o Reflection API" on page 218

Unfortunately failure happen. db4o communicates failures with exceptions. See "Exception-Handling" on page 215

Session Containers

In an application there are often multiple operations running at the same time. A typical example is a web application which processes multiple requests at the same time. These operations should be isolated from each other. This means that for the database we want to have multiple transactions at the same time. Each transaction does some work and other transactions shouldn't interfere .

db4o supports this scenario with session containers. A session container is a lightweight object-container with its own transaction and reference cache, but shares the resources with its parent container. That means you can commit and rollback changes on such a session container without disturbing the work of other session containers. If you want to implement units of work, you might considers using a session container for each unit. You can create such a container with the open session call.

```
using (IObjectContainer rootContainer = Db4oEmbedded.OpenFile(DatabaseFileName))
{
    // open the db4o-session. For example at the beginning for a web-request
    using (IObjectContainer session = rootContainer.Ext().OpenSession())
    {
        // do the operations on the session-container
        session.Store(new Person("Joe"));
    }
}
```

Db4oSessions.cs: Session object container

```
Using rootContainer As IObjectContainer = Db4oEmbedded.OpenFile(DatabaseFileName)
    ' open the db4o-session. For example at the beginning for a web-request
    Using session As IObjectContainer = rootContainer.Ext().OpenSession()
        ' do the operations on the session-container
        session.Store(New Person("Joe"))
    End Using
End Using
```

Db4oSessions.vb: Session object container

Transactions And Isolation

As previously mentioned session-containers are isolated from each other. Each session container has its own transaction and its own reference system. This isolation ensures that the different session container don't interfere witch each other.

They don't share the objects loaded and stored with each other. That means you need to load and store the a object with the same session container. When you try to load a object form one session-container and store it with another, you well end up with two separate copies of that object.

Since the transactions are isolated, changes are only visible for other session containers when you've committed. Before the commit call the changes are not visible to other session containers.

```
session1.Store(new Person("Joe"));
session1.Store(new Person("Joanna"));

// the second session won't see the changes until the changes are committed
PrintAll(session2.Query<Person>());

session1.Commit();

// new the changes are visiable for the second session
PrintAll(session2.Query<Person>());
```

Db4oSessions.cs: Session are isolated from each other

```
session1.Store(New Person("Joe"))
session1.Store(New Person("Joanna"))

' the second session won't see the changes until the changes are committed
PrintAll(session2.Query(Of Person)())

session1.Commit()

' new the changes are visiable for the second session
PrintAll(session2.Query(Of Person)())
```

Db4oSessions.vb: Session are isolated from each other

Note also that sessions also have their own reference cache. So when a object is already loaded, it wont be refreshed if another transaction updates the object. You explicitly need to refresh it.

```
Person personOnSession1 = session1.Query<Person>()[0];
Person personOnSession2 = session2.Query<Person>()[0];

personOnSession1.Name = "NewName";
session1.Store(personOnSession1);
session1.Commit();


// the second session still sees the old value, because it was cached
Console.WriteLine(personOnSession2.Name);
// you can explicitly refresh it
session2.Ext().Refresh(personOnSession2,int.MaxValue);
Console.WriteLine(personOnSession2.Name);
```

Db4oSessions.cs: Each session does cache the objects

```
Dim personOnSession1 As Person = session1.Query(Of Person)()(0)
Dim personOnSession2 As Person = session2.Query(Of Person)()(0)

personOnSession1.Name = "NewName"
session1.Store(personOnSession1)
session1.Commit()


' the second session still sees the old value, because it was cached
Console.WriteLine(personOnSession2.Name)
' you can explicitly refresh it
session2.Ext().Refresh(personOnSession2, Integer.MaxValue)
Console.WriteLine(personOnSession2.Name)
```

Db4oSessions.vb: Each session does cache the objects

Unique Constraints

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.

How To Use Unique Constraints

First you need to add an index on the field which should be unique. After that you add the UniqueFieldValueConstraint to the configuration for the unique field.

```
configuration.Common.ObjectClass(typeof (UniqueId)).ObjectField("id").Indexed(true);
configuration.Common.Add(new UniqueFieldValueConstraint(typeof (UniqueId), "id"));
```

UniqueConstrainExample.cs: Add the index the field and then the unique constrain

```
configuration.Common.ObjectClass(GetType(UniqueId)).ObjectField("id").Indexed(True)
configuration.Common.Add(New UniqueFieldValueConstraint(GetType(UniqueId), "id"))
```

UniqueConstrainExample.vb: Add the index the field and then the unique constrain

After that, the unique constrain is applied. When you commit a transaction the uniqueness of the field is checked. If there's any violation, the UniqueFieldValueConstraintViolationException will be thrown.

```
container.Store(new UniqueId(42));
container.Store(new UniqueId(42));
try
{
    container.Commit();
}
catch (UniqueFieldValueConstraintViolationException e)
{
    Console.Out.WriteLine(e.StackTrace);
}
```

UniqueConstrainExample.cs: Violating the constrain throws an exception

```
container.Store(New UniqueId(42))
container.Store(New UniqueId(42))
Try
    container.Commit()
Catch e As UniqueFieldValueConstraintViolationException
    Console.Out.WriteLine(e.StackTrace)
End Try
```

UniqueConstrainExample.vb: Violating the constrain throws an exception

Backup

db4o supplies hot backup functionality to backup single-user databases and client-server databases while they are running.

```
container.Ext().Backup("backup.db4o");
```

BackupExample.cs: Store a backup while using the database

```
container.Ext().Backup("backup.db4o")
```

BackupExample.vb: Store a backup while using the database

Maybe you want to use a other storage system for the backup than the main database. You can specify the storage system for the backup directly:

```
container.Ext().Backup(new FileStorage(), "advanced-backup.db4o");
```

BackupExample.cs: Store a backup with storage

```
container.Ext().Backup(New FileStorage(), "advanced-backup.db4o")
```

BackupExample.vb: Store a backup with storage

The methods can be called while an object container is open and they will execute with low priority in a dedicated thread, with as little impact on processing performance as possible.

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. See "Identity Concept" on page 44

If you really need to have ids for you're object, take a look at this comparison. See "Comparison Of Different IDs" on page 228

Internal IDs

Each object, except value objects like ints, floats or string, do have an internal id. This id is unique within on db4o database and db4o uses it internally for managing the objects. However you also can access this id or retrieve objects by the internal id.

You can get the internal id of an object from the extended object container.

```
long interalId = container.Ext().GetID(obj);
```

Db4oInternalIdExample.cs: get the db4o internal ids

```
Dim interalId As Long = container.Ext().GetID(obj)
```

Db4oInternalIdExample.vb: get the db4o internal ids

And you can retrieve an object by the internal id. Note that when you get an object by its internal id that it won't be activated. Therefore you have to explicitly activate the object.

```
long internalId = idForObject;
object objectForID = container.Ext().GetByID(internalId);
// getting by id doesn't activate the object
// so you need to do it manually
container.Ext().Activate(objectForID);
```

Db4oInternalIdExample.cs: get an object by db4o internal id

```
Dim internalId As Long = idForObject
Dim objForID As Object = container.Ext().GetByID(internalId)
' getting by id doesn't activate the object
' so you need to do it manually
container.Ext().Activate(objForID)
```

Db4oInternalIdExample.vb: get an object by db4o internal id

db4o UUIDs

db4o can also generate a UUIDs for each object. The UUIDs main purpose is to enable replication. By default db4o doesn't assign a UUID to each object. You have to enable this globally or for certain types. For example:

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.File.GenerateUUIDs = ConfigScope.Globally;
```

FileConfiguration.cs: Enable db4o uuids globally

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.File.GenerateUUIDs = ConfigScope.Globally
```

FileConfiguration.vb: Enable db4o uuids globally

A db4o UUID consists of two parts. The first part is the database signature which is unique to the database.

The second part a unique id within the object-container for the object. Both parts together represent a unique id.

You can get the db4o uuid from the extended object container.

```
Db4oUUID uuid = container.Ext().GetObjectInfo(obj).GetUUID();
```

Db4oUuidExample.cs: get the db4o-uuid

```
Dim uuid As Db4oUUID = container.Ext().GetObjectInfo(obj).GetUUID()
```

Db4oUuidExample.vb: get the db4o-uuid

And you can get an object by its UUID. Note that when you get an object by its UUIDthat it won't be activated. Therefore you have to explicitly activate the object.

```
object objectForId = container.Ext().GetByUUID(idForObject);
// getting by uuid doesn't activate the object
// so you need to do it manually
container.Ext().Activate(objectForId);
```

Db4oUuidExample.cs: get an object by a db4o-uuid

```
Dim objForId As Object = container.Ext().GetByUUID(idForObject)
' getting by uuid doesn't activate the object
' so you need to do it manually
container.Ext().Activate(objForId)
```

Db4oUuidExample.vb: get an object by a db4o-uuid

db4o Meta-Information

Db4o meta information API provides an access to the actual structure of db4o database file. Its primary use is refactoring.

You can access the meta information via extended object container. You can ask the object container for all stored classes or for a specific class. To find the meta information for a specific class you can provide the full name, the class itself or an instance of a particular type.

Note that db4o also returns information about internal db4o instances which have been stored.

```
// Get the information about all stored classes.
IStoredClass[] classesInDB = container.Ext().StoredClasses();
foreach (IStoredClass storedClass in classesInDB)
{
    Console.WriteLine(storedClass.GetName());
}

// Information for a certain class
IStoredClass metaInfo = container.Ext().StoredClass(typeof (Person));
```

MetaInfoExample.cs: All stored classes

```
' Get the information about all stored classes.
Dim classesInDB As IStoredClass() = container.Ext().StoredClasses()
For Each storedClass As IStoredClass In classesInDB
    Console.WriteLine(storedClass.GetName())
Next

' Information for a certain class
Dim metaInfo As IStoredClass = container.Ext().StoredClass(GetType(Person))
```

MetaInfoExample.vb: All stored classes

The stored class interface provides all meta information db4o knows about. You can get the name of the class, ask for the instance count, ask for a list of the ids and get the meta info for super classes.

The most important information about the stored classes meta info is the list of the field which are stored. You can get a list of all fields or ask for specific fields. Note that the meta information might return information for fields which don't exist anymore. This is useful for refactoring.

```
IStoredClass metaInfoForPerson = container.Ext().StoredClass(typeof (Person));
// Access all existing fields
foreach (IStoredField field in metaInfoForPerson.GetStoredFields())
{
    Console.WriteLine("Field: " + field.GetName());
}
// Accessing the field 'name' of any type.
IStoredField nameField = metaInfoForPerson.StoredField("name", null);
// Accessing the string field 'name'. Important if this field had another time in previous
// versions of the class model
IStoredField ageField = metaInfoForPerson.StoredField("age", typeof (int));

// Check if the field is indexed
bool isAgeFieldIndexed = ageField.HasIndex();

// Get the type of the field
String fieldType = ageField.GetStoredType().GetName();
```

MetaInfoExample.cs: Accessing stored fields

```
Dim metaInfoForPerson As IStoredClass = container.Ext().StoredClass(GetType(Person))
' Access all existing fields
For Each field As IStoredField In metaInfoForPerson.GetStoredFields()
    Console.WriteLine("Field: " & field.GetName())
Next
' Accessing the field 'name' of any type.
Dim nameField As IStoredField = metaInfoForPerson.StoredField("name", Nothing)
' Accessing the string field 'name'. Important if this field had another time in previous
' versions of the class model
Dim ageField As IStoredField = metaInfoForPerson.StoredField("age", GetType(Integer))

' Check if the field is indexed
Dim isAgeFieldIndexed As Boolean = ageField.HasIndex()

' Get the type of the field
Dim fieldType As String = ageField.GetStoredType().GetName()
```

MetaInfoExample.vb: Accessing stored fields

On a the field meta information you can find out the name, type and if the field has an index. And you also can access the values of a object via the stored field. This allows you to access information which is stored in the database but has been removed from the class model. This is useful for refactoring.

```
IStoredClass metaForPerson = container.Ext().StoredClass(typeof (Person));
IStoredField metaNameField = metaForPerson.StoredField("name", null);

IList<Person> persons = container.Query<Person>();
foreach (Person person in persons)
{
    string name = (string) metaNameField.Get(person);
    Console.WriteLine("Name is " + name);
}
```

MetaInfoExample.cs: Access via meta data

```
Dim metaForPerson As IStoredClass = container.Ext().StoredClass(GetType(Person))
Dim metaNameField As IStoredField = metaForPerson.StoredField("name", Nothing)

Dim persons As IList(Of Person) = container.Query(Of Person)()
For Each person As Person In persons
    Dim name As String = DirectCast(metaNameField.Get(person), [String])
    Console.WriteLine("Name is " & name)
```

MetaInfoExample.vb: Access via meta data

System Info

You can ask the object container for basic system information. Following information can be accessed:

Freespace Size

```
long freeSpaceSize = container.Ext().SystemInfo().FreespaceSize();
Console.WriteLine("Freespace in bytes: {0}", freeSpaceSize);
```

SystemInfoExamples.cs: Freespace size info

```
Dim freeSpaceSize As Long = container.Ext().SystemInfo().FreespaceSize()
Console.WriteLine("Freespace in bytes: {0}", freeSpaceSize)
```

SystemInfoExamples.vb: Freespace size info

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 freespace manager, so they can be reused for other objects.

This method returns a sum of the size of all free slots in the database file.

To reclaim freespace run Defragment.

Freespace Entry Count

```
int freeSpaceEntries = container.Ext().SystemInfo().FreespaceEntryCount();
Console.WriteLine("Freespace-entries count: {0}", freeSpaceEntries);
```

SystemInfoExamples.cs: Freespace entry count info

```
Dim freeSpaceEntries As Integer = container.Ext().SystemInfo().FreespaceEntryCount()
Console.WriteLine("Freespace-entries count: {0}", freeSpaceEntries)
```

SystemInfoExamples.vb: Freespace entry count info

Returns the number of entries in the Freespace Manager.A high value for the number of freespace entries is an indication that the database is fragmented and that Defragment should be run.

Total Size

```
long databaseSize = container.Ext().SystemInfo().TotalSize();
Console.WriteLine("Database size: {0}", databaseSize);
```

SystemInfoExamples.cs: Database size info

```
Dim databaseSize As Long = container.Ext().SystemInfo().TotalSize()
```

SystemInfoExamples.vb: Database size info

Returns the total size of the database on disk.

Defragment

A db4o database file is structured as sets of free and occupied slots, similar to 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.

The defragmentation API helps to fix this problem. It creates a new database file and copies 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.

Take a look how to defragment your database. See "How To Use Defragmentation" on page 172

Take a look at the defragmentation configuration options. See "Defragmentation Configuration" on page 173

You can register a defragmentation listener which notifies you for certain issues: See "Tracking Defragmentation Errors" on page 177

How To Use Defragmentation

The simplest way to defragment a db4o file would be:

```
Defragment.Defrag("database.db4o");
```

DefragmentationExample.cs: Simplest possible defragment use case

```
Defragment.Defrag("database.db4o")
```

DefragmentationExample.vb: Simplest possible defragment use case

Ensure that the file is not opened by any other process or db4o instance.

This moves the file *filename* to *filename.backup*, then it create a defragmented version of this database at the original position. If the backup file already exists this an IOException is throw and no action will be taken.

You can also specify the backup filename manually:

```
Defragment.Defrag("database.db4o", "database.db4o.bak");
```

DefragmentationExample.cs: Specify backup file explicitly

```
Defragment.Defrag("database.db4o", "database.db4o.bak")
```

DefragmentationExample.vb: Specify backup file explicitly

For more detailed configuration of the defragmentation process, you can use a DefragmentConfig instance:

```
DefragmentConfig config = new DefragmentConfig("database.db4o");
Defragment.Defrag(config);
```

DefragmentationExample.cs: Defragment with configuration

```
Dim config As New DefragmentConfig("database.db4o")
Defragment.Defrag(config)
```

DefragmentationExample.vb: Defragment with configuration

It's possible to use a more fine grained configuration for the deframentation process. See "Defragmentation Configuration" on page 173

Defragmentation can throw IOException in the following situations:

- Backup file exists.
- Database file not found.
- Database file is opened by another process.

Defragmentation Configuration

You can configure the defragmentation processes for your needs. This topic discusses the available configuration options.

First you need to configure which database file to defragment. See "Original Database And Backup" on page 173

It's also recommended to use the db4o database configuration for the defragmentation process. This ensures that all low level settings which influence the database-file layout are used. See "Database Configuration" on page 174

When you defragment large database you should configure a commit-frequency to speed up the defragmentation process. See "Commit Frequency" on page 174

If you have refractored your classes you might want to remove old meta data. This is possible with the class filters. See "Class Filter" on page 175

By default the backup file isn't deleted after a successful defragmentation. You can change that. See "Delete The Backup" on page 175

You can force a database update before defragmenting the database-file. See "Upgrade Database File" on page 175

You can disable the read-only more the for backup file. See "Disable Read Only Mode" on page 176

If you want to have different storage implementation for the old database file and the new defragmented database file you can configure a separate storage. See "Configure Storage for Backup File" on page 176

You can change the id-mapping implementation for the defragmentation-process. See "Configure IDMapping" on page 176

Original Database And Backup

The database-file which needs to be defragmented is specified in the configuration. The first constructor parameter is the database file.

```
DefragmentConfig config = new DefragmentConfig("database.db4o");

Defragment.Defrag(config);
```

DefragmentationConfigurationExamples.cs: Configure the file

```
Dim config As New DefragmentConfig("database.db4o")

Defragment.Defrag(config)
```

DefragmentationConfigurationExamples.vb: Configure the file

The defragmentation process creates a backup of the old database. By default the back-up file has the name of the original database-file with an additional '.backup'-suffix. You can explicitly specify the backup file name with the second constructor parameter.

```
DefragmentConfig config = new DefragmentConfig("database.db4o", "database.db4o.back");

Defragment.Defrag(config);
```

DefragmentationConfigurationExamples.cs: Configure the file and backup file

```
Dim config As New DefragmentConfig("database.db4o", "database.db4o.back")

Defragment.Defrag(config)
```

DefragmentationConfigurationExamples.vb: Configure the file and backup file

Database Configuration

Perhaps you're using low level configuration settings which are file-related. In such cases it's recommended to use the database configuration for the defragmentation process. Especially settings like string-encoding and block-size need to be configured properly for the defragmentation.

```
DefragmentConfig config = new DefragmentConfig("database.db4o");
// It's best to use the very same configuration you use for the regular database
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
config.Db4oConfig(configuration);

Defragment.Defrag(config);
```

DefragmentationConfigurationExamples.cs: Use the database-configuration

```
Dim config As New DefragmentConfig("database.db4o")
' It's best to use the very same configuration you use for the regular database
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
config.Db4oConfig(configuration)

Defragment.Defrag(config)
```

DefragmentationConfigurationExamples.vb: Use the database-configuration

Commit Frequency

The defragmentation system uses db4o's transactional core to write the data to the new defragmented file. For large databases managing a long transaction can become an issue and can slow down the defragmentation. Therefore you can set a commit frequency. This means after the set number of processed objects the transaction is committed on the new file.

```
DefragmentConfig config = new DefragmentConfig("database.db4o");
config.ObjectCommitFrequency(10000);

Defragment.Defrag(config);
```

DefragmentationConfigurationExamples.cs: Set the commit frequency

```
Dim config As New DefragmentConfig("database.db4o")
config.ObjectCommitFrequency(10000)

Defragment.Defrag(config)
```

DefragmentationConfigurationExamples.vb: Set the commit frequency

Class Filter

db4o stores meta data about all classes used in the database . Even when the class doesn't exist anymore the meta-data in db4o is still there. The class filter allows you to remove class-meta data from the defragmented database. You can pass you own implementation of a class filter. Or you can use the built in AvailableTypeFilter. This filter removes all meta-data of classes which aren't present anymore.

```
DefragmentConfig config = new DefragmentConfig("database.db4o");
config.StoredClassFilter(new AvailableTypeFilter());

Defragment.Defrag(config);
```

DefragmentationConfigurationExamples.cs: Use class filter

```
Dim config As New DefragmentConfig("database.db4o")
config.StoredClassFilter(New AvailableTypeFilter())

Defragment.Defrag(config)
```

DefragmentationConfigurationExamples.vb: Use class filter

Delete The Backup

The defragmentation process copies the data from the old database to a new file. The old file is left as a backup. You can force the defragmentation process to delete the backup after a successful defragmenting.

```
DefragmentConfig config = new DefragmentConfig("database.db4o");
config.ForceBackupDelete(true);

Defragment.Defrag(config);
```

DefragmentationConfigurationExamples.cs: Delete the backup after the defragmentation process

```
Dim config As New DefragmentConfig("database.db4o")
config.ForceBackupDelete(True)

Defragment.Defrag(config)
```

DefragmentationConfigurationExamples.vb: Delete the backup after the defragmentation process

Upgrade Database File

This option will upgrade first the database file format and then defragment it. You need to specify a folder where the temporary data for this process is stored.

```
DefragmentConfig config = new DefragmentConfig("database.db4o");
config.UpgradeFile(Environment.GetEnvironmentVariable("TEMP"));

Defragment.Defrag(config);
```

DefragmentationConfigurationExamples.cs: Upgrade database version

```
Dim config As New DefragmentConfig("database.db4o")
config.UpgradeFile(Environment.GetEnvironmentVariable("TEMP"))

Defragment.Defrag(config)
```

DefragmentationConfigurationExamples.vb: Upgrade database version

Disable Read Only Mode

By default the back-up database is opened in read only mode. You can disable the read only mode if you want.

```
DefragmentConfig config = new DefragmentConfig("database.db4o");
config.ReadOnly(false);

Defragment.Defrag(config);
```

DefragmentationConfigurationExamples.cs: Disable readonly on backup

```
Dim config As New DefragmentConfig("database.db4o")
config.[ReadOnly](False)

Defragment.Defrag(config)
```

DefragmentationConfigurationExamples.vb: Disable readonly on backup

Configure Storage for Backup File

By default the defragmentation process uses the storage from the db4o configuration for both database files, the backup and the new defragmented database file. If you want to have different storage implementations you can specify a different storage for the backup-file. This way the new defragmented database-file uses the storage from the database-configuration and the backup-file uses the back-up storage.

```
DefragmentConfig config = new DefragmentConfig("database.db4o");
config.BackupStorage(new FileStorage());

Defragment.Defrag(config);
```

DefragmentationConfigurationExamples.cs: Use a separate storage for the backup

```
Dim config As New DefragmentConfig("database.db4o")
config.BackupStorage(New FileStorage())

Defragment.Defrag(config)
```

DefragmentationConfigurationExamples.vb: Use a separate storage for the backup

Configure IDMapping

When you defragment a database each object is stored at a new location. To keep track which old id maps the which new location a IDMapper is uses. You can change the IDMapper or even implement your own.

By default the InMemoryIdMapping is uses, which is the fastest version, but consumes the most memory. As alterative there's the DatabaseIdMapping available, which stores mapping in a file and therefore uses less memory.

```
IIdMapping mapping = new InMemoryIdMapping();
DefragmentConfig config = new DefragmentConfig("database.db4o", "database.db4o.back", mapping);

Defragment.Defrag(config);
```

DefragmentationConfigurationExamples.cs: Choose a id mapping system

```
Dim mapping As IIdMapping = New InMemoryIdMapping()
Dim config As New DefragmentConfig("database.db4o", "database.db4o.back", mapping)

Defragment.Defrag(config)
```

DefragmentationConfigurationExamples.vb: Choose a id mapping system

Tracking Defragmentation Errors

You can pass a defragmentation listener to the defragmentation process. This listener will be notified when there's no object for an id in the database. This means that a object has a reference to a non-existing object. This happens when you delete objects which are still referenced by other objects.

```
private class DefragmentListener : IDefragmentListener
{
    public void NotifyDefragmentInfo(DefragmentInfo defragmentInfo)
    {
        Console.WriteLine(defragmentInfo);
    }
}
```

DefragmentationExample.cs: Defragmentation listener implementation

```
Private Class DefragmentListener
    Implements IDefragmentListener
    Public Sub NotifyDefragmentInfo(ByVal defragmentInfo As DefragmentInfo) _
        Implements IDefragmentListener.NotifyDefragmentInfo

        Console.WriteLine(defragmentInfo)
    End Sub
End Class
```

DefragmentationExample.vb: Defragmentation listener implementation

```
DefragmentConfig config = new DefragmentConfig("database.db4o");
Defragment.Defrag(config, new DefragmentListener());
```

DefragmentationExample.cs: Use a defragmentation listener

```
Dim config As New DefragmentConfig("database.db4o")
Defragment.Defrag(config, New DefragmentListener())
```

DefragmentationExample.vb: Use a defragmentation listener

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. This process inserts special, usually short, sequences of bytecode at designated points in your code. It is typically used for profiling or monitoring, however the range of use of bytecode instrumentation is not limited by this tasks: it can be applied anywhere where a specific functionality should be plugged into the ready built classes.

db4o Enhancement Tools currently have these cases for bytecodeinstrumentation:

- Transparent Activation
- Transparent Persistence
- Native Query Optimization

In transparent activation/persistence case, classes are required to implement IActivatable interface to support transparent activation. In many cases you don't want to pollute your 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 bytecode instrumentation comes handy: IActivatableinterface 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 the instrumentation afterwards.

In the native query optimization case bytecode instrumentation is used as a more performant alternative to a run-time optimization. When an native query is optimized the user and compiler-friendly syntax of **NQ**[1] predicate is replaced with a query-processor-friendly code. 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.

The instrumentation can be run at build time, also known as static instrumentation. In this case a special build script calls runs the instrumentation on the classes before packaging them to assembly, or on the assembly itself. This is the fastest solution as no time is spent on bytecode instrumentation at runtime.

There are different possibilities to integrate the enhancement tools into a project.

- It's possible to integrate the enhancement step into the build process by using a MSBuild-task. See "Build Time Enhancement Example" on page 178
- It's possible to use the Db4oTool directly from the command-line. See "Db4oTool" on page 180

Build Time Enhancement Example

It possible to integrate the db4o enhancements into the build process, using a special MSBuild-Task.

Preparation

The enhance-functionality resides in the Db4oTool.exe, Db4oTool.MSBuild.dll and its dependencies. All these files are in the db4o-distribution. Ensure that all dll-files are in the same directory. Then ensure that the build-script has configured the right location for the Db4oTool.exe etc. For this example in the Db4oTool and dependencies are in the 'lib'-folder of the project

Create the Enhancement Task

First we define the enhancement-task. This task will process the assembly and enhance it.

You can add this to the existing project-files. The .csproj or .vbproj are actually MSBuild-files. Open them with a XML-Editor and add the needed parts.

```
<UsingTask AssemblyFile=".\lib\Db4oTool.MSBuild.dll" TaskName="Db4oTool.MSBuild.Db4oEnhancerMSBuildTask" />
<ItemGroup>
  <Db4oEnhance Include="$(TargetPath)" />
</ItemGroup>
```

simple-enhance-example.csproj: Define a task for the enhancement

And the execute the task after the compilation.

```
<Target Name="AfterBuild">
  <Db4oEnhancerMSBuildTask Assemblies="@(Db4oEnhance)" />
</Target>
```

simple-enhance-example.csproj: Define a target which runs the task

Now Visual Studio will automatically run the tasks for each build. You don't need to change anything else.

Often it's practical to have all persistent classes in a separate project or compile unit. Then the enhancement script runs only for this project. This makes it easy to enhance only the classes for the persistent objects.

[1]Native Query

There are lot of possibilities to tweak and configure the build-time enhancement so that it fits your requirements. See "Build Time Enhancement"

Load Time Enhancement

Its possible to enhance you classes at runtime. This mechanism uses the Java class loading mechanism. Instead of using the regular class-loader, the application needs to use a db4o-classloader which enhanced the classes when loaded.

The enable load time enhancement, you need to create a special launcher for your application. This launcher sets up the class-loader and then starts the application.

The db4o instrumenting classloader has following configuration options 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**[1] optimization code unaware of **TA**[2]. 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.

A start of your application might look like this:

```
EnhancerStarter.java: main
public static void main(String[] args) throws Exception  {
    // use a class-filter to only enhance certain classes
    ClassFilter filter = new ByNameClassFilter("yourpackages.datamodel.", true);
    // specify which enhancements you need. Typically the native-query
    // optimisation and tranparent activation support
    BloatClassEdit[] edits =  { new TranslateNQToSODAEdit() , new InjectTransparentActivationEdit(filter) };
    // specify the where the jars & classes of your project are
    URL[] urls =  { new File("/work/workspaces/db4o/tatest/bin").toURI().toURL() };
    // the launch the application
    Db4oInstrumentationLauncher.launch(edits, urls, EnhancerMain.class.getName(), args);
  }
```

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 means that you need to set explicitly the class-loader for the reflection. Set the thread-context class loader in the db4o configuration.

Don't forget that you still need to enable transparent activation / persistence in order to use it.

[1]Native Query
[2]Transparent Activation

Db4oTool

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 native queries at build time. This will improve Native Query performance by cutting of query analyzing time during execution.
2. Optimizing delegate native query 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/Persistence instrumentation. This will enable you to use transparent activation without modifying your classes or to use transparent activation on third-party classes.

If you use Db4oTool for use-cases one and two you will be able to distribute your application without Db4objects.Db4o.NativeQueries.dll (the assembly where the Native Query runtime optimizer lives).

See how the command line options of the Db4oTool. See "Db4oTool Usage" on page 180. You can call the db4o tool also from Visual Studio. See "Including Db4oTool In The Build" on page 181

Db4oTool Usage

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.

-by-attribute:PARAM	Filter types to be instrumented by attribute: `Db4oTool -ta -byattribute:Activatable MyAssembly.exe`
-by-filter:PARAM	Custom type filter: `Db4oTool -ta -byfilter:IActivatable MyAssembly.exe`
-by-name:PARAM	Filter types by name (with regular expression syntax): `Db4oTool -ta -byname:MyCompany.MyProduct MyAssembly.exe`
-not	Negate the last filter `Db4oTool -ta -byname:Db4objects.Db4o -not 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`
-collections	Instrument native collections for transparent activation/persistence. `Db4oTool -collections -tp MyAssembly.exe`
-debug	Preserves the debug information, to step through enhanced assemblies with the debugger. `Db4oTool -debug -tp -byattribute:Activatable 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`
-install-performance-counters	Enables performance counters for this assembly: `Db4oTool -install-performance-counters MyAssembly.exe`
-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`
-statistics:PARAM	Shows statistic information about the database. For example `Db4oTool -statistics databaseFile.db4o`
-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`

Including Db4oTool In The Build

The easiest way to use Db4oTool 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 Visual Studio 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.

Callbacks

Callbacks, also known as events, allow you to be notified on certain database operations. This is useful to trigger additional operations during a database operation. There are two kind of callbacks. First you can certain methods to your objects. When the method matches certain signature it will be called by db4o. See "Object Callbacks" on page 182

Additionally you can register event listener to a object-container which will be called on certain operations. See "Event Registry API" on page 182

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.

Event Registry API

You can register to events of the db4o-database. You can used these events to implement all kinds of additional functionality. Take a look a few example use-cases. See "Possible Usecases" on page 186

There's an event for each database operation. Most of the time there are two events for an operation. One is fired before the operation starts, the other when the operation ends.

Register to an Event

You can gain access to the events via a event registry. These three steps show how to register to events.

First obtain a IEventRegistry-instance from the object container.

```
IEventRegistry events = EventRegistryFactory.ForObjectContainer(container);
```

EventRegistryExamples.cs: Obtain the event-registry

```
Dim events As IEventRegistry = EventRegistryFactory.ForObjectContainer(container)
```

EventRegistryExamples.vb: Obtain the event-registry

Now you can register your event-handlers on the event registry.

```
events.Committing += HandleCommitting;
```

EventRegistryExamples.cs: register for a event

```
AddHandler events.Committing, AddressOf HandleCommitting
```

EventRegistryExamples.vb: register for a event

Then implement your event handling.

```
private static void HandleCommitting(object sender,
    CommitEventArgs commitEventArgs)
{
    // handle the event
}
```

EventRegistryExamples.cs: implement your event handling

```
Private Shared Sub HandleCommitting(ByVal sender As Object, _
                                    ByVal commitEventArgs As CommitEventArgs)
    ' handle the event
End Sub
```

EventRegistryExamples.vb: implement your event handling

Cancelable Events

Some events can cancel the operation. All events which have a CancellableObjectEventArgs-parameter can cancel the operation. When you cancel in a event, the operation won't be executed. For example:

```
IEventRegistry events = EventRegistryFactory.ForObjectContainer(container);
events.Creating +=
    delegate(object sender, CancellableObjectEventArgs args)
        {
            if (args.Object is Person)
            {
                Person p = (Person) args.Object;
                if (p.Name.Equals("Joe Junior"))
                {
                    args.Cancel();
                }
            }
        };
```

EventRegistryExamples.cs: Cancel store operation

```
Dim events As IEventRegistry = EventRegistryFactory.ForObjectContainer(container)
AddHandler events.Creating, AddressOf HandleCreatingEvent
```

EventRegistryExamples.vb: Cancel store operation

```
Private Shared Sub HandleCreatingEvent(ByVal sender As Object, _
     ByVal args As CancellableObjectEventArgs)
    If TypeOf args.[Object] Is Person Then
        Dim p As Person = DirectCast(args.[Object], Person)
        If p.Name.Equals("Joe Junior") Then
            args.Cancel()
        End If
    End If

End Sub
```

EventRegistryExamples.vb: Cancel store operation Handler

Register Events On The Server

When you want to register for the events on the server, you should register it on the server-container.

```
IObjectServer server =
        Db4oClientServer.OpenServer(DatabaseFileName, PortNumber);
IEventRegistry eventsOnServer =
        EventRegistryFactory.ForObjectContainer(server.Ext().ObjectContainer());
```

EventRegistryExamples.cs: register for events on the server

```
Dim server As IObjectServer = _
        Db4oClientServer.OpenServer(DatabaseFileName, PortNumber)
Dim eventsOnServer As IEventRegistry = _
        EventRegistryFactory.ForObjectContainer(server.Ext().ObjectContainer())
```

EventRegistryExamples.vb: register for events on the server

Commit-Events

Commit-events bring a collection of the added, updated and deleted object with it. You can iterate over these objects. The updated- and added-collections contain LazyObjectReferences, the deleted-event a FrozenObjectInfos. Note that you may cannot get deleted object-instance anymore, but only the meta-info. Furthermore the object doesn't need to be activated. So when you need to read information out if it, ensure that you've activated it first.

```
IEventRegistry events = EventRegistryFactory.ForObjectContainer(container);
events.Committed +=
    delegate(object sender, CommitEventArgs args)
        {
            foreach (LazyObjectReference reference in args.Added)
            {
                Console.WriteLine("Added " + reference.GetObject());
            }
            foreach (LazyObjectReference reference in args.Updated)
            {
                Console.WriteLine("Updated " + reference.GetObject());
            }
            foreach (FrozenObjectInfo reference in args.Deleted)
            {
                //the deleted info might doesn't contain the object anymore and
                //return the null.
                Console.WriteLine("Deleted " + reference.GetObject());
            }
        };
```

EventRegistryExamples.cs: Commit-info

```
Dim events As IEventRegistry = EventRegistryFactory.ForObjectContainer(container)
AddHandler events.Committed, AddressOf HandlingCommitEvent
```

EventRegistryExamples.vb: Commit-info

```
Private Shared Sub HandlingCommitEvent(ByVal sender As Object, ByVal args As CommitEventArgs)
    For Each reference As LazyObjectReference In args.Added
        Console.WriteLine("Added " & reference.GetObject())
    Next
    For Each reference As LazyObjectReference In args.Updated
        Console.WriteLine("Updated " & reference.GetObject())
    Next
    For Each reference As FrozenObjectInfo In args.Deleted
        'the deleted info might doesn't contain the object anymore and
        'return the null.
        Console.WriteLine("Deleted " & reference.GetObject())
    Next
End Sub
```

EventRegistryExamples.vb: Commit-info Handler

Pitfalls and Limitations

- All embedded clients-/session share the same event registry. So you need to register the events only on one.
- You cannot call recursively the event-producing operation within the event-handler. For example in the storing-event you cannot call store. In the committing-event you cannot call commit.
- In client-server mode, each client has it's own event-registry, and therefore only sees its own events. Except the committed-event. See "Events In Client Server-Mode" on page 156

Events Overview

This overview shows you all available events. Additionally it shows on which side the event is called in client-server-mode.

Event	Explanation	Cancellable	Client	Server
activating	Fired before a object is activated.	X	X	
activated	Fired after a object is activated.		X	
creating	Fired before a object is stored to first time.	X	X	
creating	Fired after a object is stored to first time.		X	
deleting	Fired before a object is deleted.	X		X
deleted	Fired after a object is deleted.			X
updating	Fired before a object is updated.	X	X	
updated	Fired after a object is updated.		X	
deactivating	Fired before a object is deactivated.	X	X	
deactivated	Fired after a object is deactivated.	X	X	
queryStarted	Fired when a query starts.		X	
queryFinished	Fired when a query has finished.		X	

committing	Fired before a commit.			X
committed	Fired after a commit.		X[1]	X
closing	Fired when the object is closed.		X	
classRegistered	Fired when a new class is stored/loaded.		X	
instantiated	Fired when a object is instantiated.		X	

Possible Usecases

There are many use cases for external callbacks, including:

- Cascaded deletes, updates.
- Referential integrity checks.
- Gathering statistics.
- Auto assigned 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:

- Referential Integrity
- Autoincrement

Autoincrement

db4o does not deliver a field auto increment feature, which is common in **RDBMS**[2]. Normally you don't need any additional ids, since db4o manages objects by object-identity. However cases where you have disconnected objects, you need additional ids. One of then possibilities it to use auto incremented ids.

If your application logic requires this feature you can implement it using external callbacks. One of the possible solutions is presented below. Note that this example only works in embedded-mode.

This example assumes that all object which need an auto incremented id are subclasses of the IDHolder-class. This class contains the auto-incremented id.

```
private int id;

public int Id
{
    get { return id; }
    set { id = value; }
}
```

IDHolder.cs: id holder

[1]This event is asynchronously distributed across all clients
[2]Relational Database Management System

```
Private m_id As Integer

Public Property Id() As Integer
    Get
        Return m_id
    End Get
    Set(ByVal value As Integer)
        m_id = value
    End Set
End Property
```

IDHolder.vb: id holder

First create a class which keeps the state of the auto-increment numbers. For example a map which keeps the latest auto incremented id for each class.

```
private class PersistedAutoIncrements
{
    private readonly IDictionary<Type, int> currentHighestIds = new Dictionary<Type, int>();

    public int NextNumber(Type forClass)
    {
        int number;
        if (!currentHighestIds.TryGetValue(forClass, out number))
        {
            number = 0;
        }
        number += 1;
        currentHighestIds[forClass] = number;
        return number;
    }
}
```

AutoIncrement.cs: persistent auto increment

```
Private Class PersistedAutoIncrements
    Private ReadOnly currentHighestIds As IDictionary(Of Type, Integer) _
                        = New Dictionary(Of Type, Integer)()

    Public Function NextNumber(ByVal forClass As Type) As Integer
        Dim number As Integer
        If Not currentHighestIds.TryGetValue(forClass, number) Then
            number = 0
        End If
        number += 1
        currentHighestIds(forClass) = number
        Return number
    End Function
End Class
```

AutoIncrement.vb: persistent auto increment

Then create two methods, which are called later. One which returns the next auto-incremented id for a certain class. Another which stores the current state of the auto-increments.

```
public int GetNextID(Type forClass)
{
    lock (dataLock)
    {
        PersistedAutoIncrements incrementState = EnsureLoadedIncrements();
        return incrementState.NextNumber(forClass);
    }
}

public void StoreState()
{
    lock (dataLock)
    {
        if (null != state)
        {
            container.Ext().Store(state,2);
        }
    }
}
```

AutoIncrement.cs: getting the next id and storing state

```
Public Function GetNextID(ByVal forClass As Type) As Integer
    SyncLock dataLock
        Dim incrementState As PersistedAutoIncrements = EnsureLoadedIncrements()
        Return incrementState.NextNumber(forClass)
    End SyncLock
End Function

Public Sub StoreState()
    SyncLock dataLock
        If state IsNot Nothing Then
            container.Ext().Store(state, 2)
        End If
    End SyncLock
End Sub
```

AutoIncrement.vb: getting the next id and storing state

The last part is to ensure that the existing auto-increments are loaded from the database. Or if not existing a new instance is created.

```
private PersistedAutoIncrements EnsureLoadedIncrements()
{
    if (null == state)
    {
        state = LoadOrCreateState();
    }
    return state;
}

private PersistedAutoIncrements LoadOrCreateState()
{
    IList<PersistedAutoIncrements> existingState = container.Query<PersistedAutoIncrements>();
    if (0 == existingState.Count)
    {
        return new PersistedAutoIncrements();
    }
    else if (1 == existingState.Count)
    {
        return existingState[0];
    }
    else
    {
        throw new InvalidOperationException("Cannot have more than one state stored in database");
    }
}
```

AutoIncrement.cs: load the state from the database

```
Private Function EnsureLoadedIncrements() As PersistedAutoIncrements
    If state Is Nothing Then
        state = LoadOrCreateState()
    End If
    Return state
End Function

Private Function LoadOrCreateState() As PersistedAutoIncrements
    Dim existingState As IList(Of PersistedAutoIncrements) = container.Query(Of PersistedAutoIncrements)()
    If 0 = existingState.Count Then
        Return New PersistedAutoIncrements()
    ElseIf 1 = existingState.Count Then
        Return existingState(0)
    Else
        Throw New InvalidOperationException("Cannot have more than one state stored in database")
    End If
End Function
```

AutoIncrement.vb: load the state from the database

Now it's time to use the callbacks. Every time when a new object is created, assign a new id. For this the creating-event is perfect. When commiting also make the auto increment-state persistent, to ensure that no id is used twice.

```
AutoIncrement increment = new AutoIncrement(container);
IEventRegistry eventRegistry = EventRegistryFactory.ForObjectContainer(container);

eventRegistry.Creating+=
    delegate(object sender, CancellableObjectEventArgs args)
    {
        if (args.Object is IDHolder)
        {
            IDHolder idHolder = (IDHolder)args.Object;
            idHolder.Id = increment.GetNextID(idHolder.GetType());
        }
    };
eventRegistry.Committing +=
    delegate(object sender, CommitEventArgs args)
        {
            increment.StoreState();
        };
```

AutoIncrementExample.cs: use events to assign the ids

```
Dim increment As New AutoIncrement(container)
Dim eventRegistry As IEventRegistry = EventRegistryFactory.ForObjectContainer(container)

AddHandler eventRegistry.Creating, AddressOf increment.HandleCreating
AddHandler eventRegistry.Committing, AddressOf increment.HandleCommiting
```

AutoIncrementExample.vb: use events to assign the ids

Last, don't forget to index the id-field. Otherwise looks-ups will be slow.

```
configuration.Common.ObjectClass(typeof (IDHolder)).ObjectField("id").Indexed(true);
```

AutoIncrementExample.cs: index the id-field

```
configuration.Common.ObjectClass(GetType(IDHolder)).ObjectField("id").Indexed(True)
```

AutoIncrementExample.vb: index the id-field

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:

```
private static void ReferentialIntegrityCheck(object sender,
                                              CancellableObjectEventArgs eventArguments)
{
    Object toDelete = eventArguments.Object;
    if (toDelete is Pilot)
    {
        IObjectContainer container = eventArguments.ObjectContainer();
        IEnumerable<Car> cars = from Car c in container
                                where c.Pilot == toDelete
                                select c;
        if (cars.Count() > 0)
        {
            eventArguments.Cancel();
        }
    }
}
```

CallbackExamples.cs: Referential integrity

```
Private Shared Sub ReferentialIntegrityCheck(ByVal sender As Object, ByVal eventArguments As CancellableObjectEve
    Dim toDelete As Object = eventArguments.Object
    If TypeOf toDelete Is Pilot Then
        Dim container As IObjectContainer = eventArguments.ObjectContainer()
        Dim cars As IEnumerable(Of Car) = From c As Car In container _
                                          Where c.Pilot Is toDelete
        If cars.Count() > 0 Then
            eventArguments.Cancel()
        End If
    End If
End Sub
```

CallbackExamples.vb: Referential integrity

```
IEventRegistry events = EventRegistryFactory.ForObjectContainer(container);
events.Deleting += ReferentialIntegrityCheck;
```

CallbackExamples.cs: Register handler

```
Dim events As IEventRegistry = EventRegistryFactory.ForObjectContainer(container)
AddHandler events.Deleting, AddressOf ReferentialIntegrityCheck
```

CallbackExamples.vb: Register handler

You can also add handlers for creating() and updating() events for a Car object to make sure that the pilot field is not null.

Note, that in client/server mode deleting event is only raised on the server side, therefore the code above can't be used and will throw an exception.

Committed Event Example

Committed callbacks can be used in various scenarios:

- Backup on commit.
- Database replication on commit.
- Client database synchronization.

This example shows you how to refresh objects on a client on commits.

When several clients are working on the same objects it is possible that the data will be outdated on a client. You can use the committed-event refresh object on each commit.

When a client commit will trigger a committed event on all clients. In order to refresh the object, register for the committed event. In the commit-event-handler, refresh the object which have been modified.

```
IEventRegistry events = EventRegistryFactory.ForObjectContainer(container);
events.Committed +=
    delegate(object sender, CommitEventArgs args)
        {
            foreach (LazyObjectReference updated in args.Updated)
            {
                object obj = updated.GetObject();
                args.ObjectContainer().Ext().Refresh(obj, 1);
            }
        };
```

RefreshingObjects.cs: On the updated-event we refresh the objects

```
Dim events As IEventRegistry = EventRegistryFactory.ForObjectContainer(container)
AddHandler events.Committed, AddressOf HandleUpdate
```

RefreshingObjects.vb: On the updated-event we refresh the objects

```
Private Shared Sub HandleUpdate(ByVal sender As Object, ByVal args As CommitEventArgs)
    For Each updated As LazyObjectReference In args.Updated
        Dim obj As Object = updated.GetObject()
        args.ObjectContainer().Ext().Refresh(obj, 1)
    Next
End Sub
```

RefreshingObjects.vb: The refresh-handler

You can register such a event-handler for each client. The committed event is transferred to each client. Note that this requires a lot of network-traffic to notify all clients and transfer the changes.

When working with committed events you should remember that the listener is called on a separate thread, which needs to be synchronized with the rest of the application.

Commit-Time Callbacks

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 object container 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.

Type Handling

db4o tries to be simple an easy to use. One big part if this is to transparently store any object without any complex mapping or configuration. Storing a object correctly is a complex process and heavily depends on the type of object. db4o has different storing strategies for different types.

For most types the regular db4o type handling is sufficient.

Since collections are so important, db4o treats collections specially to improve efficiency. See "Collections" on page 195

Blobs

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 Memory.
- Querying interface.
- Objects' modification.
- Information backup.
- Client/Server processing.

db4o provides you with a flexibility of using 2 different solutions for this case:

1. The db4o blob-type.
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. Special code is necessary to store and load .
3. No concerns about activation depth.

byte[] array

1. Data in the same file
2. Transparent handling without special concerns.
3. 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.

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.

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 .

1. Special code is necessary to store and load.

   - It is more difficult to move objects between db4o database files.

2. No concerns about activation depth

   + Big objects won't be loaded into memory as part of the activation process.

Configuration

First, the blob storage location should be defined. If that value is not defined, db4o will use the default folder "blobs" in user directory.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.File.BlobPath = "myBlobDirectory";
```

FileConfiguration.cs: Configure the blob-path

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.File.BlobPath = "myBlobDirectory"
```

FileConfiguration.vb: Configure the blob-path

Using The db4o-Blob

There are two important operations on the blob type. The first one write a file into the db4o blob:

```
blob.ReadFrom(fileToStore);
```

BlobStorage.cs: Store the file as a db4o-blob

```
blob.ReadFrom(fileToStore)
```

BlobStorage.vb: Store the file as a db4o-blob

And then there's the operation which .loads the db4o blob into a new file.

```
blob.WriteTo(file);
```

BlobStorage.cs: Load a blob from a db4o-blob

```
blob.WriteTo(file)
```

BlobStorage.vb: Load a blob from a db4o-blob

The db4o blob-type has a status attached to it. This status tells you if the blob-file all ready has been transferred:

```
while (blob.GetStatus() > Status.Completed)
{
    Thread.Sleep(50);
}
```

BlobStorage.cs: wait until the operation is done

```
While blob.GetStatus() > Status.Completed
    Thread.Sleep(50)
End While
```

BlobStorage.vb: wait until the operation is done

Take a look a the complete example code (C#, VB.NET).

Collections

Internally the regular .NET-collections are stored as arrays. When you store a collection, db4o will store all the collection-items into a array. And when you retrieve a collection, db4o will create a new instance of the collection and add back the items. Dictionary-instances are stored as an array of key-values.

Unfortunately this implementation is not very efficient for searches/updates of a certain value in a collection, as the whole collection needs to be instantiated to access any of its elements.

db4o brings some special collections with it. There are collections which support transparent persistence. See "TA Aware Collections" on page 61

When you have a need for a huge collection, you might run into some performance bottleneck, since collections are always stored and retrieved as complete unit. You can use db4o special big-set to improve performance. See "Big Set" on page 195

You might wonder what is better to use, collection or arrays. Most of times it doesn't matter. See "Collections Or Arrays" on page 195

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 you need to consider that collections are more flexible and convenient than arrays for most operations. Additionally, collections can be **TA**[1]/**TP**[2] aware by using db4o-collections, while arrays are always fully activated. See "TA Aware Collections" on page 61

Big Set

When you need to store large sets, you can use db4o's big set. This big-set operates directly on top of B-trees, which are also used for indexes. The big-set doesn't need to activate all items to perform its operations. For example when you check if the set already contains a member, the big-set can do that without activating all its items. Especially lookup-operation like contains perform much faster with a big set.

Not that currently the big set implementation only works in embedded-mode, but not in client-server mode.

You can create a new big-set with the CollectionFactory:

```
ICollection<Person> citizen = CollectionFactory.ForObjectContainer(container).NewBigSet<Person>();
// now you can use the big-set like a normal set:
citizen.Add(new Person("Citizen Kane"));
```

BigSetExample.cs: Crate a big-set instance

[1]Transparent Activation
[2]Transparent Persistence

```
Dim citizen As ICollection(Of Person) = _
    CollectionFactory.ForObjectContainer(container).NewBigSet(Of Person)()
' now you can use the big-set like a normal set:
citizen.Add(New Person("Citizen Kane"))
```

BigSetExample.vb: Crate a big-set instance

After that, the big-set behaves just like an ordinary set. Except that the big-set used the object-identity instead of the object-equality to compare the items. So when you add a equal object with a different identity, it will be added to the set. You can add, remove and iterate over the items or check if an item is already in the set. The items will be loaded and activated on demand, for example when you iterate over the set.

```
Person aCitizen;
using (IEnumerator<Person> aCitizenEnumerator = city.Citizen.GetEnumerator())
{
    aCitizenEnumerator.MoveNext();
    aCitizen = aCitizenEnumerator.Current;
}
Console.WriteLine("The big-set uses the identity, not equality of an object");
Console.WriteLine("Therefore it .contains() on the same person-object is "
                  + city.Citizen.Contains(aCitizen));
Person equalPerson = new Person(aCitizen.Name);
Console.WriteLine("Therefore it .contains() on a equal person-object is "
                  + city.Citizen.Contains(equalPerson));
```

BigSetExample.cs: Note that the big-set compares by identity, not by equality

```
Dim aCitizen As Person
Using aCitizenEnumerator As IEnumerator(Of Person) = city.Citizen.GetEnumerator()
    aCitizenEnumerator.MoveNext()
    aCitizen = aCitizenEnumerator.Current
End Using
Console.WriteLine("The big-set uses the identity, not equality of an object")
Console.WriteLine("Therefore it .contains() on the same person-object is " & city.Citizen.Contains(aCitizen))
Dim equalPerson As New Person(aCitizen.Name)
```

BigSetExample.vb: Note that the big-set compares by identity, not by equality

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
- .NET Enumerations

.NET Enumerations

.NET enumerations are value types, just like integers, numbers, booleans etc.

.NET enumerations are based on an underlying type, which can be Byte, SByte, Int32, UInt32, Int16, UInt16, Int64, and UInt64 (i.e. values of these types will be enumerated). When underlying type is not specified in the enum declaration, Int32(c#) or Integer (VB.NET) is used by default.

db4o stores .net enums just like regular value types like ints, floats etc. It is stored together with the containing object. It can only be loaded and stored with the containing object.

Storing Static Fields

By default db4o does not persist static fields. This is not necessary because static values are set for a class, not for an object. However you can set up db4o to store static fields if you want to implement constants or enumeration: See "Persist Static Fields" on page 147

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Person)).PersistStaticFieldValues();
```

ObjectConfigurationExamples.cs: Persist also the static fields

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).PersistStaticFieldValues()
```

ObjectConfigurationExamples.vb: Persist also the static fields

When this setting is enabled, all non-primitive-typed static fields are stored the first time an instance of the class is stored. The values are restored every time a database file is opened afterwards, after the class meta information is loaded for this class (when the class objects are retrieved with a query, for example).

Use this option with caution. This option means that static fields are stored in the database. When you change the value of this field, you need to store it explicitly again. Furthermore, db4o will replace the static value at runtime, which can lead to very subtle bugs in your application.

This option does not have any effect on primitive types like ints, longs, floats etc.

Enum Class Use case

One of the use-cases is when you have an enumeration-class which you want to store. For example we have a color-class, which also has some static colors.

```
public sealed class Color
{
    public static readonly Color Black = new Color(0, 0, 0);
    public static readonly Color White = new Color(255, 255, 255);
    public static readonly Color Red = new Color(255, 0, 0);
    public static readonly Color Green = new Color(0, 255, 0);
    public static readonly Color Blue = new Color(0, 0, 255);

    private readonly int red;
    private readonly int green;
    private readonly int blue;

    private Color(int red, int green, int blue)
    {
        this.red = red;
        this.green = green;
        this.blue = blue;
    }

    public int RedValue
    {
        get { return red; }
    }

    public int GreenValue
    {
        get { return green; }
    }

    public int BlueValue
    {
        get { return blue; }
    }

    public bool Equals(Color other)
    {
        if (ReferenceEquals(null, other)) return false;
        if (ReferenceEquals(this, other)) return true;
        return other.red == red && other.green == green && other.blue == blue;
    }

    public override bool Equals(object obj)
    {
        if (ReferenceEquals(null, obj)) return false;
        if (ReferenceEquals(this, obj)) return true;
        if (obj.GetType() != typeof (Color)) return false;
        return Equals((Color) obj);
    }

    public override int GetHashCode()
    {
        unchecked
        {
            int result = red;
            result = (result*397) ^ green;
            result = (result*397) ^ blue;
            return result;
        }
    }
    public override string ToString()
    {
        return string.Format("Red: {0}, Green: {1}, Blue: {2}", red, green, blue);
    }
}
```

Color.cs: Class as enumeration

```
Public NotInheritable Class Color
    Public Shared ReadOnly Black As New Color(0, 0, 0)
    Public Shared ReadOnly White As New Color(255, 255, 255)
    Public Shared ReadOnly Red As New Color(255, 0, 0)
    Public Shared ReadOnly Green As New Color(0, 255, 0)
    Public Shared ReadOnly Blue As New Color(0, 0, 255)

    Private ReadOnly m_red As Integer
    Private ReadOnly m_green As Integer
    Private ReadOnly m_blue As Integer

    Private Sub New(ByVal red As Integer, ByVal green As Integer, ByVal blue As Integer)
        Me.m_red = red
        Me.m_green = green
        Me.m_blue = blue
    End Sub

    Public ReadOnly Property RedValue() As Integer
        Get
            Return m_red
        End Get
    End Property

    Public ReadOnly Property GreenValue() As Integer
        Get
            Return m_green
        End Get
    End Property

    Public ReadOnly Property BlueValue() As Integer
        Get
            Return m_blue
        End Get
    End Property

    Public Overloads Function Equals(ByVal other As Color) As Boolean
        If ReferenceEquals(Nothing, other) Then Return False
        If ReferenceEquals(Me, other) Then Return True
        Return other.m_red = m_red AndAlso other.m_green = m_green AndAlso other.m_blue = m_blue

    End Function

    Public Overloads Overrides Function Equals(ByVal obj As Object) As Boolean
        If ReferenceEquals(Nothing, obj) Then Return False
        If ReferenceEquals(Me, obj) Then Return True
        If Not Equals(obj.GetType(), GetType(Color)) Then Return False
        Return Equals(DirectCast(obj, Color))

    End Function

    Public Overrides Function GetHashCode() As Integer
        Dim hashCode As Integer = m_red
        hashCode = (hashCode * 397) Xor m_green
        hashCode = (hashCode * 397) Xor m_blue
        Return hashCode
    End Function

    Public Overrides Function ToString() As String
        Return String.Format("Red: {0}, Green: {1}, Blue: {2}", m_red, m_green, m_blue)
    End Function
End Class
```

Color.vb: Class as enumeration

We want to ensure reference equality on colors so that you easily can check for a certain color. But when we load the colors from the database you get new instances and not the same instance as in the static field. This means that comparing the references will fail.

```
// When you enable persist static field values, you can compare by reference
// because db4o stores the static field
if (car.Color == Color.Black)
{
    Console.WriteLine("Black cars are boring");
}
else if (car.Color == Color.Red)
{
    Console.WriteLine("Fire engine?");
}
```

StoringStaticFields.cs: Compare by reference

```
' When you enable persist static field values, you can compare by reference
' because db4o stores the static field
If car.Color Is Color.Black Then
    Console.WriteLine("Black cars are boring")
ElseIf car.Color Is Color.Red Then
    Console.WriteLine("Fire engine?")
```

StoringStaticFields.vb: Compare by reference

When you enable the persist static fields option, the static fields are stored. This means that the object referenced in the static fields are loaded from the database and therefore the same instance. And the comparing the references works again.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Color)).PersistStaticFieldValues();
```

StoringStaticFields.cs: Enable storing static fields for our color class

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Color)).PersistStaticFieldValues()
```

StoringStaticFields.vb: Enable storing static fields for our color class

Delegates And Events

Db4o rules for delegate fields are very straightforward: **delegates are simply not stored.** Events and delegates are generally used for binding user interface elements and domain models together. The Db4o team felt that not storing delegate fields by default was more appropriate than opening what could potentially be a very nasty can of worms (just think of a text box bound to a Customer.Changed event). After careful thought we can easily add delegate persistence to our domain model by either installing translators for the delegate types of interest or reconnecting the necessary objects upon activation using callbacks. For details see the specific chapters on Translators and Callbacks.

Translators

Sometimes objects cannot be stored in db4o. For example because the objects have references to other parts of the system other where never intended to be stored. This is especially for objects from third party libraries.

Now the db4o object translators is a simple mechanism which allows you to manually handle the persistence of an object. There are two important interfaces for this. The IObjectTranslator-interface and the IObjectConstructor. The first interface lets you take the control over storing and activation of the object. The second interface also allows you to control the instantiation of the object.

If you register a IObjectTranslator-instance for a certain type, it will also be applied to sub-types. This doesn't apply to IObjectConstructor-instances, because those need to create the right instance and therefore cannot handle subtypes.

Creating a Translator

First you need to create a translator for your types. Let's take a look at this example. There three distinct tasks a translator has to do. The first task is to convert the not storable object into another, storable object. Another task of the translator is to take care of the activation of the object. There it need to copy the values from the stored object into a instance of the original type. The third task it to create instances of the object. There you create a instance of the original type. And for some types you maybe also read the data at this point in time.

```
internal class ExampleTranslator : IObjectConstructor
{
    // This is called to store the object
    public Object OnStore(IObjectContainer objectContainer, Object objToStore)
    {
        NonStorableType notStorable = (NonStorableType) objToStore;
        return notStorable.Data;
    }

    // This is called when the object is activated
    public void OnActivate(IObjectContainer objectContainer, Object targetObject, Object storedObject)
    {
        NonStorableType notStorable = (NonStorableType) targetObject;
        notStorable.Data = (String) storedObject;
    }

    // Tell db4o which type we use to store the data
    public Type StoredClass()
    {
        return typeof (String);
    }

    // This method is called when a new instance is needed
    public Object OnInstantiate(IObjectContainer objectContainer, Object storedObject)
    {
        return new NonStorableType("");
    }
}
```

ExampleTranslator.cs: An example translator

```
Friend Class ExampleTranslator
    Implements IObjectConstructor
    ' This is called to store the object
    Public Function OnStore(ByVal objectContainer As IObjectContainer, _
                            ByVal objToStore As Object) As Object _
                        Implements IObjectConstructor.OnStore

        Dim notStorable As NonStorableType = DirectCast(objToStore, NonStorableType)
        Return notStorable.Data
    End Function

    ' This is called when the object is activated
    Public Sub OnActivate(ByVal objectContainer As IObjectContainer, _
                          ByVal targetObject As Object, ByVal storedObject As Object) _
                      Implements IObjectConstructor.OnActivate

        Dim notStorable As NonStorableType = DirectCast(targetObject, NonStorableType)
        notStorable.Data = DirectCast(storedObject, String)
    End Sub

    ' Tell db4o which type we use to store the data
    Public Function StoredClass() As Type _
        Implements IObjectConstructor.StoredClass

        Return GetType(String)
    End Function

    ' This method is called when a new instance is needed
    Public Function OnInstantiate(ByVal objectContainer As IObjectContainer, _
                                  ByVal storedObject As Object) As Object _
        Implements IObjectConstructor.OnInstantiate
        Return New NonStorableType("")
    End Function
End Class
```

ExampleTranslator.vb: An example translator

Registering a Translator

After that you can register the translator for you're type. If you register a IObjectTranslator-instance it will also be applied to the sub-types. However a IObjectConstructor-instance is only applied for the specific type.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof(NonStorableType)).Translate(new ExampleTranslator());
IObjectContainer container = Db4oEmbedded.OpenFile(configuration, "database.db4o");
```

ExampleTranslator.cs: Register type translator for the NonStorableType-class

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(NonStorableType)).Translate(New ExampleTranslator())
Dim container As IObjectContainer = Db4oEmbedded.OpenFile(configuration, "database.db4o")
```

ExampleTranslator.vb: Register type translator for the NonStorableType-class

Using The Translator

After that you can store and use the not storable objects like any other persistent objects. db4o will call the translator for each instance when required in order to store the object correctly.

```
container.Store(new NonStorableType("TestData"));
```

ExampleTranslator.cs: Store the non storable type

```
container.Store(New NonStorableType("TestData"))
```

ExampleTranslator.vb: Store the non storable type

```
NonStorableType data = container.Query<NonStorableType>()[0];
```

ExampleTranslator.cs: Load the non storable type

```
Dim data As NonStorableType = container.Query(Of NonStorableType)()(0)
```

ExampleTranslator.vb: Load the non storable type

Limitations

The object translator mechanism is great for types which couldn't be stored otherwise. However there are serious limitations.

- Queries into the members of a object which was stored with a object translator are extremely slow. The reason is that the object first need to be loaded and instantiated with the translator in order to run the query on it.
- You cannot index types which are translated.

Built-In Translators

db4o supplies some build-in translators, which can be used in general cases. You can use them for your classes if they are not storable or nedd special treatment.

- TTransient: Doesn't store the object at all. Usable when you don't want to store instances of certain type. If makes all instances of that type transient.
- TSerializable: Uses the built in serialisation mechanism to store this object.

There are other built in translators, which are not intended be used directly. Instead they are used by db4o internally. Nevertheless you can use them as example implementation. Look for all classes which implement the IObjectTranslator-interface

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: Translators, transient fields in Java and .NET 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 way to write a specific persistence solution for any custom object, and now db4o provides this way though a pluggable ITypeHandler4 interface. You can register any number of type handlers to the configuration. Additionally you need to register a predicate which decides which classes are handled by the type handler.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.RegisterTypeHandler(
    new SingleClassTypeHandlerPredicate(typeof(StringBuilder)), new StringBuilderHandler());
```

TypeHandlerExample.cs: Register type handler

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.RegisterTypeHandler(
    New SingleClassTypeHandlerPredicate(GetType(StringBuilder)), New StringBuilderHandler())
```

TypeHandlerExample.vb: Register type handler

Note that type handler are a low level API which doesn't protect you from any mistakes. If you make a mistake in a typehandler you can loose data.

Type handler functionality is best explained on a working example.

Usecases and other benefits of the pluggable typehandler interface are reviewed here.

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 Transientmarker, 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 need to perform a complex refactoring on-the-fly (use typehandler versioning)
- You want to cipher each object before putting it into the database

Other not so common and more difficult in realization behaviours that can be realized with the new Typehandler:

- 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.

Custom Typehandler Example

For a custom typehandler example we will try to write a very simple typehandler for the StringBuilder class. We want to handle a StringBuilder as a value type, therefore we implement the ValueTypeHandler interface. Not that there's a whole collection of interfaces for typehandlers. Take a look at the TypeHandler4 type hierarchy.

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 the write method, which determines how the object is persisted:

```
public void Write(IWriteContext writeContext, object o)
{
    StringBuilder builder = (StringBuilder) o;
    string str = builder.ToString();
    byte[] bytes = Encoding.UTF8.GetBytes(str);
    writeContext.WriteInt(bytes.Length);
    writeContext.WriteBytes(bytes);
}
```

StringBuilderHandler.cs: Write the StringBuilder

```
Public Sub Write(ByVal writeContext As IWriteContext, ByVal o As Object) _
    Implements IValueTypeHandler.Write
    Dim builder As StringBuilder = DirectCast(o, StringBuilder)
    Dim str As String = builder.ToString()
    Dim bytes As Byte() = Encoding.UTF8.GetBytes(str)
    writeContext.WriteInt(bytes.Length)
    writeContext.WriteBytes(bytes)
End Sub
```

StringBuilderHandler.vb: Write the StringBuilder

As you can see from the code above, there are 3 steps:

1. Get the buffer from WriteContext/I WriteContext
2. Convert the string-content to a byte-array using the UTF8 encoding.
3. Write the length of the resulted byte-array.
4. Write the byte array of the string.

Next step is to read the stored object. It is just opposite to the write method:

```
    public Object Read(IReadContext readContext)
    {
        int length = readContext.ReadInt();
        byte[] data = new byte[length];
        readContext.ReadBytes(data);
        return new StringBuilder(Encoding.UTF8.GetString(data));
    }
}
```

StringBuilderHandler.cs: Read the StringBuilder

```
Public Function Read(ByVal readContext As IReadContext) As Object _
    Implements IValueTypeHandler.Read
    Dim length As Integer = readContext.ReadInt()
    Dim data As Byte() = New Byte(length - 1) {}
    readContext.ReadBytes(data)
    Return New StringBuilder(Encoding.UTF8.GetString(data))
End Function
```

StringBuilderHandler.vb: Read the StringBuilder

Delete is simple - we just reposition the buffer offset to the end of the slot:

```
public void Delete(IDeleteContext deleteContext)
{
    SkipData(deleteContext);
}

private static void SkipData(IReadBuffer deleteContext)
{
    int numBytes = deleteContext.ReadInt();
    deleteContext.Seek(deleteContext.Offset() + numBytes);
}
```

StringBuilderHandler.cs: Delete the content

```
Public Sub Delete(ByVal deleteContext As IDeleteContext) _
    Implements IValueTypeHandler.Delete
    SkipData(deleteContext)
End Sub

Private Shared Sub SkipData(ByVal deleteContext As IReadBuffer)
    Dim numBytes As Integer = deleteContext.ReadInt()
    deleteContext.Seek(deleteContext.Offset() + numBytes)
End Sub
```

StringBuilderHandler.vb: Delete the content

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):

```
public void Defragment(IDefragmentContext defragmentContext)
{
    SkipData(defragmentContext);
}
```

StringBuilderHandler.cs: Defragment the content

```
Public Sub Defragment(ByVal defragmentContext As IDefragmentContext) _
    Implements IValueTypeHandler.Defragment
    SkipData(defragmentContext)
End Sub
```

StringBuilderHandler.vb: Defragment the content

Now to use this type handler we need to configure db4o. To register a typehandler you have to provide a predicate which decides if a type is handled by the typehandler and the typehandler itself.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.RegisterTypeHandler(
    new SingleClassTypeHandlerPredicate(typeof(StringBuilder)), new StringBuilderHandler());
```

TypeHandlerExample.cs: Register type handler

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.RegisterTypeHandler(
    New SingleClassTypeHandlerPredicate(GetType(StringBuilder)), New StringBuilderHandler())
```

TypeHandlerExample.vb: Register type handler

After that all string builders are handled by you're type handler.

Refactoring and Schema Evolution

Application design is a volatile thing: it changes from version to version, from one customer implementation to another. The database 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.

Simple cases like adding or removing a field and changing interfaces are handled automatically. See "Automatic Refactoring" on page 208

For renaming classes and fields you can use the renaming API. See "Renaming API" on page 208.

You can change the type of a field any time. However db4o won't migrate the data to the new type. You need to do this explicitly. See "Field Type Change" on page 209 And there are some small limitations when refactoring the field-type. See "Field Refactoring Limitation" on page 210

Unfortunatly db4o doesn't support changing the inheritance hierarchy. See "Refactoring Class Hierarchy" on page 211

Automatic Refactoring

In simple cases db4o handles schema changes automatically:

- When you **add** a new field, db4o automatically starts storing the new data. Older instances of your stored class have the default value in the new field.
- When you **remove** a field, db4o ignores the stored value for that field. The stored value is not removed from the database until you update the object or run a defragmentation. Meanwhile the old values are still accessible with the StoredClass/StoredField API.
- You can **add an interface** to a class any time. The db4o operations are not affected by interfaces.

Renaming API

db4o provides a special API to move classes between namespace, rename classes or fields.

Rename a Class

Use the configuration API to rename a class.You need to rename the class before you open the database. Remember to include the assembly name.

```
configuration.Common.ObjectClass("Db4oDoc.Code.Strategies.Refactoring.PersonOld, Db4oDoc")
    .Rename("Db4oDoc.Code.Strategies.Refactoring.PersonNew, Db4oDoc");
```

RefactoringExamples.cs: Rename a class

```
configuration.Common.ObjectClass("Db4oDoc.Code.Strategies.Refactoring.PersonOld, Db4oDoc") _
    .Rename("Db4oDoc.Code.Strategies.Refactoring.PersonNew, Db4oDoc")
```

RefactoringExamples.vb: Rename a class

Rename a Field

Use the configuration API to rename a field. You need to rename the field before you open the database. Remember to include the assembly name.

```
configuration.Common.ObjectClass("Db4oDoc.Code.Strategies.Refactoring.PersonOld, Db4oDoc")
    .ObjectField("name").Rename("sirname");
```

RefactoringExamples.cs: Rename field

```
configuration.Common.ObjectClass("Db4oDoc.Code.Strategies.Refactoring.PersonOld, Db4oDoc") _
    .ObjectField("name").Rename("sirname")
```

RefactoringExamples.vb: Rename field

Rename Step by Step

The safe order of actions for renaming is:

1. Backup your database.
2. Close all open object containers if any.
3. Rename classes or fields in your application.
4. Add the renaming call to on the configuration before opening the database.
5. Pass the configuration with the rename-information to the object container factory. Open the data-base and you're ready to work with the renamed classes or fields.

Field Type Change

db4o's default policy is to never do any damage to stored data. When you change the type of a field, db4o will not update the data in this field. Instead db4o internally creates a new field of the same name, but with the new type. For existing object, the values of the old typed field are still present, but hidden. Of course you can access the old data. When you want to convert the content from the old field type to the new field type, you have to do it yourself.

You can use the stored-class API to retrieve the data of the old typed field. An example: We decide that we want to refactor the id-field from a simple int to a special identity-class. First we change the field-type:

```
public Identity id = Identity.NewId();
//  was an int previously:
//    public int id = new Random().nextInt();
```

RefactoringExamples.cs: change type of field

```
Public m_id As Identity = Identity.NewId()
'  was an int previously:
'    public int id = new Random().nextInt();
```

RefactoringExamples.vb: change type of field

After than read the old value from the old field-type and convert it to the new type:

```
using (IObjectContainer container = Db4oEmbedded.OpenFile("database.db4o"))
{
    // first get all objects which should be updated
    IList<Person> persons = container.Query<Person>();
    foreach (Person person in persons)
    {
        // get the database-meta data about this object-type
        IStoredClass dbClass = container.Ext().StoredClass(person);
        // get the old field which was an int-type
        IStoredField oldField = dbClass.StoredField("id", typeof (int));
        if(null!=oldField)
        {
            // Access the old data and copy it to the new field!
            Object oldValue = oldField.Get(person);
            if (null != oldValue)
            {
                person.id = new Identity((int)oldValue);
                container.Store(person);
            }
        }
    }
}
```

RefactoringExamples.cs: copying the data from the old field type to the new one

```
Using container As IObjectContainer = Db4oEmbedded.OpenFile("database.db4o")
    ' first get all objects which should be updated
    Dim persons As IList(Of Person) = container.Query(Of Person)()
    For Each person As Person In persons
        ' get the database-meta data about this object-type
        Dim dbClass As IStoredClass = container.Ext().StoredClass(person)
        ' get the old field which was an int-type
        Dim oldField As IStoredField = dbClass.StoredField("id", GetType(Integer))
        If oldField IsNot Nothing Then
            ' Access the old data and copy it to the new field!
            Dim oldValue As [Object] = oldField.[Get](person)
            If oldValue IsNot Nothing Then
                person.id = New Identity(CInt(oldValue))
                container.Store(person)
            End If
        End If
    Next
End Using
```

RefactoringExamples.vb: copying the data from the old field type to the new one

db4o's approach gives you the maximum flexibility for refactoring field types. You can handle the convertion with regular code, which means it can be as complex as needed. Furthermore you can decide when you convert the values. You can update all objects in one operation, you can dynamically update and covert when you access a object or even decide not to convert the old values.

Field Refactoring Limitation

For most cases changing the field type isn't an issue. db4o keeps the old values around and you can access the old values without issues. See "Field Type Change" on page 209

However there's one limitation to this mechanism. You cannot change the type of a field to its array-type and vice versa. This only applies if it's the same array-type. For example:

- You cannot change a string field to a string array field and vice versa.
- You can change a string field to an int-, object-, etc array. Every type is possible except a string-array.
- You can change a string-array to an int-, object etc. Every type is possible except a string.

Refactoring To An Array-Field Step by Step

When you change the type of a field to its array-type equivalent, you can do this only by copying the old data to a new class. In this example we have a Person-class which has its name in a string field. Now we want to change that to a string array to support multiple names.

1. Create a copy of the Person-class with a new name.
2. Do the refactoring on the new Person class
3. Query for old instances of the old Person-class and copy the values over to the new class.

```
IList<PersonOld> oldPersons = container.Query<PersonOld>();
foreach (PersonOld old in oldPersons)
{
    PersonNew newPerson = new PersonNew();
    newPerson.Name = new string[] {old.Name};
    container.Store(newPerson);
    container.Delete(old);
}
```

ChangeArrayType.cs: Copy the string-field to the new string-array field

```
Dim oldPersons As IList(Of PersonOld) = container.Query(Of PersonOld)()
For Each old As PersonOld In oldPersons
    Dim newPerson As New PersonNew()
    newPerson.Name = New String() {old.Name}
    container.Store(newPerson)
    container.Delete(old)
Next
```

ChangeArrayType.vb: Copy the string-field to the new string-array field

Note that this example doesn't change existing references from the old instances to the new ones. You need to do this manually as well.

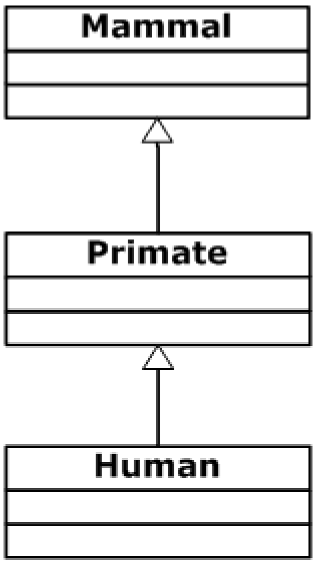
Refactoring Class Hierarchy

db4o does not directly support the following two refactorings:

- Inserting classes into an inheritance hierarchy.
- Removing classes from inheritance hierarchy.

Note that interfaces don't influence the inheritance-hierarchy and can be added and removed at any time.

For example we've following classes:



In this example you cannot introduce a 'Animal' class above the 'Mammal' or add another class between 'Mammal' and 'Primate'. Also you shouldn't remove an class form the inheritance-hierarchy.

Currently the only possible solution for this refactoring is this.

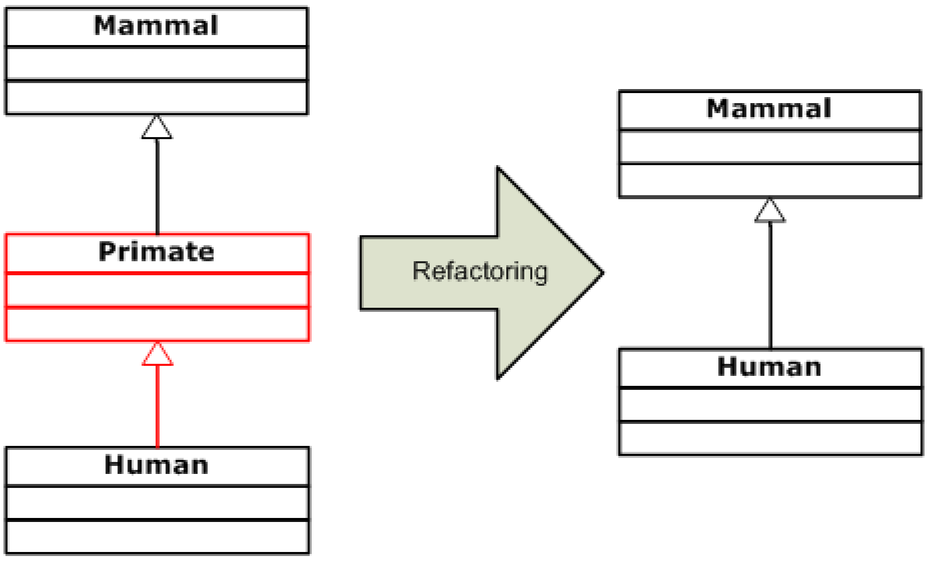
1. Create the new hierarchy with different names, preferably in a new package
2. Copy all values from the old classes to the new classes.
3. Redirect all references from existing objects to the new classes.

Take a look at the example to how to add a class into the hierarchy. See "Inserting Class Into A Hierarchy" on page 213.

Or how you can remove a class from the inheritance hierarchy. See "Removing Class From A Hierarchy" on page 212

Removing Class From A Hierarchy

In this example we have a Human class which inherits from the Primate class. Now we want to remove the Primate class and let the Human class inherit directly from the Mammal class.



Unfortunately db4o doesn't support this kind of refactoring. We need to use a work-around. Basically we create a copy of the Human class with the new Inheritance-hierarchy and the copy the existing data over.

Step by Step

1. Create a copy of the Human class, for example HumanNew!
2. Change the inheritance of the HumanNew-class to inherit directly from the Mammal-class.
3. After that, load all existing Human-instances, copy the values over to HumanNew-instances. Store the HumanNew-instance and delete the old Human-instances

Now the objects have the new inheritance hierarchy. You can delete the old Human-class.

```
IList<Human> allMammals = container.Query<Human>();
foreach (Human oldHuman in allMammals)
{
    HumanNew newHuman = new HumanNew("");
    newHuman.BodyTemperature = oldHuman.BodyTemperature;
    newHuman.IQ = oldHuman.IQ;
    newHuman.Name = oldHuman.Name;

    container.Store(newHuman);
    container.Delete(oldHuman);
}
```

RemoveClassFromHierarchy.cs: copy the data from the old type to the new one

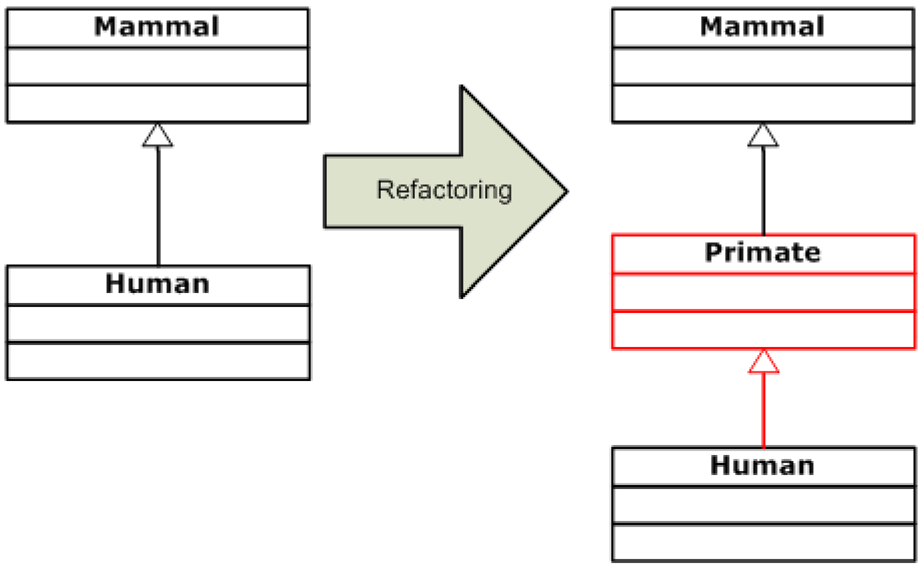
```
Dim allMammals As IList(Of Human) = container.Query(Of Human)()
For Each oldHuman As Human In allMammals
    Dim newHuman As New HumanNew("")
    newHuman.BodyTemperature = oldHuman.BodyTemperature
    newHuman.IQ = oldHuman.IQ
    newHuman.Name = oldHuman.Name

    container.Store(newHuman)
    container.Delete(oldHuman)
```

RemoveClassFromHierarchy.vb: copy the data from the old type to the new one

Note that this example doesn't change existing references from the old instances to the new ones. You need to do this manually as well.

Inserting Class Into A Hierarchy

In this example we have a Human-class witch inherits from the Mammal class. Now we want to introduce a new Primate class and let the Human class inherit from it.



Unfortunately db4o doesn't support this kind of refactoring. We need to use a work-around. Basically we create a copy of the Human-class with the new Inheritance-hierarchy and the copy the existing data over.

Step by Step

1. Create the new Primate-class
2. Create a copy of the Human-class, for example HumanNew!
3. Change the HumanNew class to inherit from the new Primate-class instead of the Mammal-class.
4. After that, load all existing Human-instances, copy the values over to HumanNew-instances. Store the HumanNew-instance and delete the old Human-instances

Now the objects have the new inheritance hierarchy. You can delete the old Human-class.

```
IList<Human> allMammals = container.Query<Human>();
foreach (Human oldHuman in allMammals)
{
    HumanNew newHuman = new HumanNew("");
    newHuman.BodyTemperature = oldHuman.BodyTemperature;
    newHuman.IQ = oldHuman.IQ;
    newHuman.Name = oldHuman.Name;

    container.Store(newHuman);
    container.Delete(oldHuman);
}
```

AddClassToHierarchy.cs: copy the data from the old type to the new one

```
Dim allMammals As IList(Of Human) = container.Query(Of Human)()
For Each oldHuman As Human In allMammals
    Dim newHuman As New HumanNew("")
    newHuman.BodyTemperature = oldHuman.BodyTemperature
    newHuman.IQ = oldHuman.IQ
    newHuman.Name = oldHuman.Name

    container.Store(newHuman)
    container.Delete(oldHuman)
Next
```

AddClassToHierarchy.vb: copy the data from the old type to the new one

Note that this example doesn't change existing references from the old instances to the new ones. You need to do this manually as well.

Exception-Handling

A part of the db4o operations is handling possible exceptions. There are two fundamental different exception-types for db4o. The recoverable exceptions and the fatal exceptions.

Recoverable Exceptions

The recoverable exceptions are all exceptions which don't endanger the consistency of the database. For example if a event callback throws an exception. After a recoverable exception you can continue to work with the object container.

Typical recoverable exceptions are:

- Exceptions in callbacks.
- Passing invalid ids to the object container.
- Unsupported schema-changes.
- Constraint-violations.

Fatal Exceptions

A fatal exception will immediately shut down the object container without committing anything. This tries to protect the database from damaging itself. Any exception which happens in the db4o core and is not expected and handled is considered as a fatal exception. Because when an exception happens in the db4o-core, it could cause a invalid state in the db4o-core and then cause further errors and lead to database corruption. That's why the policy it the stop immediately any operation after a fatal exception.

Typical fatal recoverable exceptions are:

- Exceptions related to runtime, like OutOfMemory-exceptions
- Exceptions related to corrupted database-files.

Handle-Exceptions

Take a look a the list of the most common db4o related exceptions. See "Exception Types" on page 215 Handling db4o exceptions is nothing special and complies with regular exception handling. See "How To Work With db4o Exceptions" on page 216

Exception Types

Using db4o you will have to deal with db4o-specific exceptions and system exceptions thrown directly out of db4o.

db4o-exceptions are chained; you can get the cause of the exception using:

c#:

```
db4oException.Source;
```

VB:

```
db4oException.Source;
```

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 violations.

UniqueFieldValueConstraintViolationException - An exception which will be thrown when the unique constrain is violated.

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. See "Increasing The Maximum Database File Size" on page 318

DatabaseReadOnlyException - This exception is thrown when a write operation was attempted on a database in read-only mode.
GlobalOnlyConfigException - This exception is thrown when you try to change a setting on a open object container, but this setting cannot be changed at runtime.

IncompatibleFileFormatException - 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 a client tries to connect to a server with the wrong password.

EventException - This exception is thrown when a exception is thrown in a event callback.

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.

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. Take also a look at the list of common db4o exceptions.

1. Opening a database file can throw a DatabaseFileLockedException.

```
try
{
    IObjectContainer container = Db4oEmbedded.OpenFile("database.db4o");
}
catch (DatabaseFileLockedException e)
{
    // Database is already open!
    // Use another database-file or handle this case gracefully
}
```

ImportantExceptionCases.cs: If the database is already open

```
Try
    Dim container As IObjectContainer = Db4oEmbedded.OpenFile("database.db4o")
Catch e As DatabaseFileLockedException
    ' Database is already open!
    ' Use another database-file or handle this case gracefully
End Try
```

ImportantExceptionCases.vb: If the database is already open

2. Opening a client connection can throw IOException.

```
try
{
    IObjectContainer container = Db4oClientServer.OpenClient("localhost", 1337, "sa", "sa");
}
catch (Db4oIOException e)
{
    // Couldn't connect to the server.
    // Ask for new connection-settings or handle this case gracefully
}
```

ImportantExceptionCases.cs: Cannot connect to the server

```
Try
    Dim container As IObjectContainer = Db4oClientServer.OpenClient("localhost", 1337, "sa", "sa")
Catch e As Db4oIOException
    ' Couldn't connect to the server.
    ' Ask for new connection-settings or handle this case gracefully
End Try
```

ImportantExceptionCases.vb: Cannot connect to the server

3. Working with db4o-unique constraints the commit may throw exceptions when the constraints are violated.

```
container.Store(new UniqueId(42));
container.Store(new UniqueId(42));
try
{
    container.Commit();
}
catch (UniqueFieldValueConstraintViolationException e)
{
    // Violated the unique-constraint!
    // Retry with a new value or handle this gracefully
    container.Rollback();
}
```

ImportantExceptionCases.cs: Violation of the unique constraint

```
container.Store(New UniqueId(42))
container.Store(New UniqueId(42))
Try
    container.Commit()
Catch e As UniqueFieldValueConstraintViolationException
    ' Violated the unique-constraint!
    ' Retry with a new value or handle this gracefully
    container.Rollback()
End Try
```

ImportantExceptionCases.vb: Violation of the unique constraint

db4o Reflection API

Reflection gives your code access to internal information for classes loaded into the CLR. It allows you to explore the structure of objects at runtime. In the case of reflection metadata is the description of classes and objects within the CLR, including their fields, methods and constructors. It allows the programmer to select target classes at runtime, create new objects, call their methods and operate with the fields.

In order to persist object db4o uses the reflection to read object and store their values. You can exchange this reflector layer in the configuration.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ReflectWith(new FastNetReflector());
```

CommonConfigurationExamples.cs: Change the reflector

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ReflectWith(New FastNetReflector())
```

CommonConfigurationExamples.vb: Change the reflector

There are currently two reflectors available:

- **NetReflector**: The default reflector which uses standard .NET reflection for inspecting and accessing objects.
- **FastNetReflector**: A reflector which generates special access-code for each class. Therefore it is a lot faster than the default NetReflector. However it has a higher memory-consumption due to the generated code.

If you need some special treatment you can create you're own reflector implementation. See "Creating your own reflector" on page 218

db4o has also a generic reflector which can deal with stored objects without using the original class. See "GenericReflector" on page 221

Creating your own reflector

By default db4o uses the NetReflector. As alternative you can create your own reflector and use it with db4o. In order to do so you need to implement the IReflector interface. And then pass an instance of your implementation to db4o.

Here's an example of a logging reflector. Its only difference from standard reflector is that information about loaded classes is outputted to console:

```
internal class LoggerReflector : IReflector
{
    private readonly IReflector readReflector;

    public LoggerReflector()
    {
        this.readReflector = new NetReflector();
    }

    public LoggerReflector(IReflector readReflector)
    {
        this.readReflector = readReflector;
    }

    public void Configuration(IReflectorConfiguration reflectorConfiguration)
    {
        readReflector.Configuration(reflectorConfiguration);
    }

    public IReflectArray Array()
    {
        return readReflector.Array();
    }

    public IReflectClass ForClass(Type type)
    {
        Console.WriteLine("Reflector.forClass({0})", type);
        return readReflector.ForClass(type);
    }

    public IReflectClass ForName(string className)
    {
        Console.WriteLine("Reflector.forName({0})", className);
        return readReflector.ForName(className);
    }

    public IReflectClass ForObject(object o)
    {
        Console.WriteLine("Reflector.forObject(" + o + ")");
        return readReflector.ForObject(o);
    }

    public bool IsCollection(IReflectClass reflectClass)
    {
        return readReflector.IsCollection(reflectClass);
    }

    public void SetParent(IReflector reflector)
    {
        readReflector.SetParent(reflector);
    }

    public object DeepClone(object o)
    {
        return new LoggerReflector((IReflector) readReflector.DeepClone(o));
    }
}
```

ReflectorExamples.cs: Logging reflector

```
Friend Class LoggerReflector
    Implements IReflector
    Private ReadOnly readReflector As IReflector

    Public Sub New()
        Me.readReflector = New NetReflector()
    End Sub

    Public Sub New(ByVal readReflector As IReflector)
        Me.readReflector = readReflector
    End Sub

    Public Sub Configuration(ByVal reflectorConfiguration As IReflectorConfiguration) _
        Implements IReflector.Configuration
        readReflector.Configuration(reflectorConfiguration)
    End Sub

    Public Function Array() As IReflectArray _
        Implements IReflector.Array
        Return readReflector.Array()
    End Function

    Public Function ForClass(ByVal type As Type) As IReflectClass _
        Implements IReflector.ForClass
        Console.WriteLine("Reflector.forClass({0})", type)
        Return readReflector.ForClass(type)
    End Function

    Public Function ForName(ByVal className As String) As IReflectClass _
        Implements IReflector.ForName
        Console.WriteLine("Reflector.forName({0})", className)
        Return readReflector.ForName(className)
    End Function

    Public Function ForObject(ByVal o As Object) As IReflectClass _
        Implements IReflector.ForObject
        Console.WriteLine("Reflector.forObject(" & Convert.ToString(o) & ")")
        Return readReflector.ForObject(o)
    End Function

    Public Function IsCollection(ByVal reflectClass As IReflectClass) As Boolean _
        Implements IReflector.IsCollection
        Return readReflector.IsCollection(reflectClass)
    End Function

    Public Sub SetParent(ByVal reflector As IReflector) _
        Implements IReflector.SetParent
        readReflector.SetParent(reflector)
    End Sub

    Public Function DeepClone(ByVal o As Object) As Object _
        Implements IReflector.DeepClone
        Return New LoggerReflector(DirectCast(readReflector.DeepClone(o), IReflector))
    End Function
End Class
```

ReflectorExamples.vb: Logging reflector

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 instantiation of objects using class name possible. However db4o reflection API can also work on generic objects when a class information is not available.

db4o uses a generic reflector as a decorator around specific reflector. The generic reflector is set when an object container is opened. All subsequent reflector calls are routed through this decorator.

The generic reflector keeps list of known classes in memory. When the generic reflector 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 use cases:

- Running a db4o server without deploying application classes.
- Easier access to stored objects where classes or fields are not available.
- Building interfaces to db4o from any programming language.

One of the live use cases is the ObjectManager, which uses the generic reflector to read C# objects from Java.

The generic reflector is automatically used when the class of a stored object is not found.

Best Practices

This topic is a collection of best practices for db4o.

db4o doesn't have any support for limiting the result size or skipping objects in the result set. You can do this on top of db4o with little effort. See "Paging" on page 222

How should you scope the lifetime of an object container? See this guide-line: See "Lifetime Of An Object Container" on page 224

Deleting object is always a delicate process. See "Deleting Objects" on page 225

If you upgrade to new major db4o version you should do these steps to update the file format. See "Upgrade db4o Version" on page 224

Paging

Currently db4o doesn't provide any paging mechanism at all. However all db4o query results are lazy loaded. db4o returns a result list which only contains the ids of the objects and will load the object as soon as you access it. This means you can page by only accessing the indexes of the range you're interested in.

You can access the list directly by the indexes to get the right objects. With this you can build a paging-utility methods which start at a certain index and return a certain amount of objects. Take a look a this example utility methods.

Note that LINQ brings already methods for paging with it. The skip and take methods are optimal for implementing a paging mechanism.

```
public static IList<T> Paging<T>(IList<T> listToPage, int limit)
{
    return Paging(listToPage, 0, limit);
}

public static IList<T> Paging<T>(IList<T> listToPage, int start, int limit)
{
    if (start > listToPage.Count)
    {
        throw new ArgumentException("You cannot start the paging outside the list." +
                                    " List-size: " + listToPage.Count + " start: " + start);
    }
    int end = calculateEnd(listToPage, start, limit);
    IList<T> list = new List<T>();
    for (int i = start; i < end; i++)
    {
        list.Add(listToPage[i]);
    }
    return list;
}

private static int calculateEnd<T>(IList<T> resultList, int start, int limit)
{
    int end = start + limit;
    if (end >= resultList.Count)
    {
        return resultList.Count;
    }
    return end;
}
```

PagingUtility.cs: Paging utility methods

```
Public Shared Function Paging(Of T)(ByVal listToPage As IList(Of T), ByVal limit As Integer) As IList(Of T)
    Return Paging(listToPage, 0, limit)
End Function

Public Shared Function Paging(Of T)(ByVal listToPage As IList(Of T), ByVal start As Integer, ByVal limit As Integ
    If start > listToPage.Count Then
        Throw New ArgumentException("You cannot start the paging outside the list." & " List-size: " & listToPage
    End If
    Dim endPosition As Integer = calculateEnd(listToPage, start, limit)
    Dim list As IList(Of T) = New List(Of T)()
    For i As Integer = start To endPosition - 1
        list.Add(listToPage(i))
    Next
    Return list
End Function

Private Shared Function calculateEnd(Of T)(ByVal resultList As IList(Of T), _
         ByVal start As Integer, ByVal limit As Integer) As Integer
    Dim endPosition As Integer = start + limit
    If endPosition >= resultList.Count Then
        Return resultList.Count
    End If
    Return endPosition
End Function
```

PagingUtility.vb: Paging utility methods

And then of course you can use the utility methods on the result-sets of db4o.

```
IList<StoredItems> queryResult = container.Query<StoredItems>();
IList<StoredItems> pagedResult = PagingUtility.Paging(queryResult, 2, 2);
```

TestPagingUtility.cs: Use the paging utility

```
Dim queryResult As IList(Of StoredItems) = container.Query(Of StoredItems)()
Dim pagedResult As IList(Of StoredItems) = PagingUtility.Paging(queryResult, 2, 2)
```

TestPagingUtility.vb: Use the paging utility

Upgrade db4o Version

How do you upgrade to a newer version safely? By default db4o can read an older version of a database and operate on it without any issues. However for major version updates you should consider doing this updating procedure. This ensures that the database uses the newest file format and performance optimizations.

The safest update procedure:

1. Defragment the database with the current version. This will automatically also create a backup.
2. Upgrade to the new db4o version.
3. After that immediately defragment the database with the current version. This will create another backup. After that the database file uses the most current database format.
4. Make sure that you can read the data from the database. If everything works you can delete the backups.

Lifetime Of An Object Container

The main interface to your database is the object container. You do all major operations with a object container instance. Now how long should you keep a object container open? Is it better to close the object container after each operation. Or keep the object container open? This topic is a small guideline on the lifetime of an object container.

Understanding Object Containers

The lifetime of a object container heavily depends on your application scenario. Therefore it's vital to understand what a object container represents. A object container consist of a transaction and a reference cache and is thread safe. This means that a object container is isolated from other containers. Furthermore an object container has a cache, so you don't want to throw a object container away if you don't have to.

One Object Container Per Unit Of Work

This means in fact that you should use one object container per unit of work. You can use a object container for succeeding units of work. But you should never use the same object container for concurrent or independent units of work. You can create new object containers at any time.

Desktop Application

What does this mean for a desktop application? Well in a simple desktop application you might can use only one object container. Since there only one user doing something at a time you can use the same object container for all operations.

For complexes desktop application multiple object container can be an option. For example a object container per tab, per wizard etc.

Be aware only because object container is thread safe it doesn't make your object model thread safe. If multiple threads are using the same object container they will use the same objects and modify the same object concurrently. You need to use an appropriate model for the concurrent access.

Web Application

In a web application you should use a object container per request. Each request is it unit of work. Maybe there are multiple unit of work successively in a request, but a request represents the top level unit of work.

Deleting Objects

Deleting object is always a delicate process. Deleting the wrong object can be catastrophic. Here are some best practices for deleting objects.

Delete Flag

When a end user deletes a object it's often better to use a deleted-flag instead of actually deleting the data. This has the advantage that you can undo the delete operation at any time. Also you don't break the model in cases where the user deleted the wrong object. However it has also some disadvantages. You need to honor the deleted-flag in your queries.

You can set the delete flag in a callback and use the regular db4o delete operation:

```
Private Shared Sub HandleDeleteEvent(ByVal sender As Object, ByVal args As CancellableObjectEventArgs)
    Dim obj As Object = args.Object
    ' if the object has a deletion-flag:
    ' set the flag instead of deleting the object
    If TypeOf obj Is Deletable Then
        DirectCast(obj, Deletable).Delete()
        args.ObjectContainer().Store(obj)
        args.Cancel()
    End If
End Sub
```

DeletionStrategies.vb: The delete event handler

```
IEventRegistry events = EventRegistryFactory.ForObjectContainer(container);
events.Deleting +=
    (sender, args) =>
        {
            object obj = args.Object;
            // if the object has a deletion-flag:
            // set the flag instead of deleting the object
            if (obj is Deletable)
            {
                ((Deletable) obj).Delete();
                args.ObjectContainer().Store(obj);
                args.Cancel();
            }
        };
```

DeletionStrategies.cs: Deletion-Flag

```
Dim events As IEventRegistry = EventRegistryFactory.ForObjectContainer(container)
AddHandler events.Deleting, AddressOf HandleDeleteEvent
```

DeletionStrategies.vb: Deletion-Flag

However you need to filter the deleted objects in every query.

Be Very Careful

db4o doesn't support any referential integrity. When you delete a object and there's still a reference to that object this reference is set to null. This means if you delete a object you may break the consistency of you're object model.

This means also that you need to implement any consistency check yourself on top of db4o. You can use db4o callbacks for doing so.

Use Cascade Deletion Wisely

You can configure db4o the cascade delete referenced objects. You can configure that for certain type or certain fields. As said there's no referential integrity checks for db4o, so you have to extreamly conscious where to use this feature. It makes sense to configure cascade deletion for composition roots, where you are sure that the children cannot be referenced from another location. In all other places it's a bad idea most of the time.

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 5 or newer
- .NET 3.5 or newer
- .NET 3.5 - .NET 4.0 CompactFramework
- Mono (you need to compile db4o yourself)
- Android
- Silverlight

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:

- Android
- Windows CE or Windows Mobile (.NET Compact Framework)

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 .NET Platforms
- Security Requirements On .NET Platform
- ASP.NET
- Xml Import-Export In .NET
- Automatic Properties
- Isolated Storage

Disconnected Objects

db4o manages objects by object-identity. db4o ensures that each stored object in the database is always represented by the same object in memory. When you load an object, change it and then store it, db4o recognizes the object by its identity and will update it.

This model works wonderful as long as the object-identity is preserved. However there are a lot of scenarios, where the object-identity is lost. As soon as serialize objects, the object-identity is lost. This is typical for web-scenarios or web-services, where a object needs to be identified across requests. For such scenarios objects need additional ids to identify object across requests- and object-container-boundaries.

There are several possibilities for such additional ids. You can use db4o internal ids, db4o uuids, or additional ids-fields on objects. Each has its advantages and disadvantages, so take a looks at this comparison: See "Comparison Of Different IDs" on page 228

Only identifying the object across object container boundaries is often not enough. You actually need to update a disconnected object. This is done by coping the new values to the existing object. See "Merging Changes" on page 232

Comparison Of Different IDs

There are a lot of possibilities for additional ids to identify objects across the object container boundary.

Internal db4o ids

db4o has internal ids to identify each object in the database. You can access these ids and use them yourself. See "Internal IDs". Take a look at the example. See "Example Internal Id" on page 229

Advantages	Disadvantages
• Internal ids are fast. • No additional field on the class required. • No additional configuration required.	• The id may change when defragmentating the database.

db4o UUIDs

db4o supports special UUIDs. You can enable them by configuration. See "Unique Universal IDs". Take a look at the example. See "Example db4o UUID" on page 229

Advantages	Disadvantages
• A UUID is a worldwide unique id. • No additional field on the class required.	• db4o UUIDs are large. • db4o UUID is db4o-specific type.

Guid-Fields on Classes

You can add Guid-fields to your classes. In the constructor of the object you assign a new Guid to the object. Then you can find the object by a regular query. . Don't forget to index the id-field. Take a look at the example. See "Example Guid" on page 230

Advantages	Disadvantages
• A Guid is a worldwide unique id . • Guid are easy to generate and portable.	• You need an id-field on your objects. • Guids are quite large. • Additional index required.

ID-Field On Classes With a ID-Generator

You can add a id-field to your classes and then use an ID-Generator to assign new ids to stored objects. Don't forget to index the id-field. Take a look at the example. See "Example ID-Generator" on page 231

Advantages	Disadvantages
• A simple id on objects. • Familiar model from the **RDBMS**[1]-world.	• You need an id-field on your objects. • You need to implement an ID-Generator. Which isn't trivial. • Additional index required.

[1]Relational Database Management System

Example Internal Id

This example demonstrates how you can use internal object ids to identify objects across objects containers. Take a look advantages and disadvantages of internal ids: See "Comparison Of Different IDs" on page 228

For using the internal ids no additional configuration is required. You can get this id for any object.

You can get the internal id from the extended object container.

```
long interalId = container.Ext().GetID(obj);
```

Db4oInternalIdExample.cs: get the db4o internal ids

```
Dim interalId As Long = container.Ext().GetID(obj)
```

Db4oInternalIdExample.vb: get the db4o internal ids

Getting a object by its id is also easy. First you get the object from the container. Unlike queries this won't return a activated object. So you have to do it explicitly.

```
long internalId = idForObject;
object objectForID = container.Ext().GetByID(internalId);
// getting by id doesn't activate the object
// so you need to do it manually
container.Ext().Activate(objectForID);
```

Db4oInternalIdExample.cs: get an object by db4o internal id

```
Dim internalId As Long = idForObject
Dim objForID As Object = container.Ext().GetByID(internalId)
' getting by id doesn't activate the object
' so you need to do it manually
container.Ext().Activate(objForID)
```

Db4oInternalIdExample.vb: get an object by db4o internal id

Example db4o UUID

This example demonstrates how you can use db4o-UUIDs to identify objects across objects containers. Take a look advantages and disadvantages of db4o-UUIDs: See "Comparison Of Different IDs" on page 228

First you need to enable db4o-UUIDs in order to use it.

```
configuration.File.GenerateUUIDs = ConfigScope.Globally;
```

Db4oUuidExample.cs: db4o-uuids need to be activated

```
configuration.File.GenerateUUIDs = ConfigScope.Globally
```

Db4oUuidExample.vb: db4o-uuids need to be activated

With UUIDs turned on, db4o will create an UUID for each stored object. So you can get the UUID of the object from the object-container.

```
Db4oUUID uuid = container.Ext().GetObjectInfo(obj).GetUUID();
```

Db4oUuidExample.cs: get the db4o-uuid

```
Dim uuid As Db4oUUID = container.Ext().GetObjectInfo(obj).GetUUID()
```

Db4oUuidExample.vb: get the db4o-uuid

Getting a object by its UUID is also easy. First you get the object from the container. Unlike queries this won't activate your object. So you have to do it explicitly.

```
object objectForId = container.Ext().GetByUUID(idForObject);
// getting by uuid doesn't activate the object
// so you need to do it manually
container.Ext().Activate(objectForId);
```

Db4oUuidExample.cs: get an object by a db4o-uuid

```
Dim objForId As Object = container.Ext().GetByUUID(idForObject)
' getting by uuid doesn't activate the object
' so you need to do it manually
container.Ext().Activate(objForId)
```

Db4oUuidExample.vb: get an object by a db4o-uuid

Example Guid

This example demonstrates how you can use Guids to identify objects across objects containers. Take a look advantages and disadvantages of Guids: See "Comparison Of Different IDs" on page 228

This example assumes that all object have a common super-class, IDHolder, which holds the Guid in a field.

```
private readonly Guid guid = Guid.NewGuid();

public Guid ObjectId
{
    get { return guid; }
}
```

IDHolder.cs: generate the id

```
Private ReadOnly guid As Guid = Guid.NewGuid()

Public ReadOnly Property ObjectId() As Guid
    Get
        Return guid
    End Get
End Property
```

IDHolder.vb: generate the id

It's important to index the id-field, otherwise looking up for object by id will be slow.

```
configuration.Common.ObjectClass(typeof (IDHolder)).ObjectField("guid").Indexed(true);
```

UuidOnObject.cs: index the uuid-field

```
configuration.Common.ObjectClass(GetType(IDHolder)).ObjectField("guid").Indexed(True)
```

UuidOnObject.vb: index the uuid-field

The id is hold by the object itself, so you can get it directly.

```
IDHolder uuidHolder = (IDHolder) obj;
Guid uuid = uuidHolder.ObjectId;
```

UuidOnObject.cs: get the uuid

```
Dim uuidHolder As IDHolder = DirectCast(obj, IDHolder)
Dim uuid As Guid = uuidHolder.ObjectId
```

UuidOnObject.vb: get the uuid

You can get the object you can by a regular query.

```
IDHolder instance = container.Query(delegate(IDHolder o) { return o.ObjectId.Equals(idForObject); })[0];
```

UuidOnObject.cs: get an object its UUID

```
Dim instance As IDHolder = container.Query(Function(o As IDHolder) o.ObjectId.Equals(idForObject))(0)
```

UuidOnObject.vb: get an object its UUID

Example ID-Generator

This example demonstrates how you can use an ID-generator to identify objects across objects containers. Take a look advantages and disadvantages of ID-generators: See "Comparison Of Different IDs" on page 228

This example assumes that all object have a common super-class, IDHolder, which holds the id.

```
private int id;

public int Id
{
    get { return id; }
    set { id = value; }
}
```

IDHolder.cs: id holder

```
Private m_id As Integer

Public Property Id() As Integer
    Get
        Return m_id
    End Get
    Set(ByVal value As Integer)
        m_id = value
    End Set
End Property
```

IDHolder.vb: id holder

Don't forget to index the id-field. Otherwise finding objects by id will be slow.

```
configuration.Common.ObjectClass(typeof (IDHolder)).ObjectField("id").Indexed(true);
```

AutoIncrementExample.cs: index the id-field

```
configuration.Common.ObjectClass(GetType(IDHolder)).ObjectField("id").Indexed(True)
```

AutoIncrementExample.vb: index the id-field

The hard part is to write an efficient ID-Generator. For this example a very simple auto increment generator is used. Use the creating-callback-event in order to add the ids to the object. When committing, store the state of the id-generator.

```
AutoIncrement increment = new AutoIncrement(container);
IEventRegistry eventRegistry = EventRegistryFactory.ForObjectContainer(container);

eventRegistry.Creating+=
    delegate(object sender, CancellableObjectEventArgs args)
    {
        if (args.Object is IDHolder)
        {
            IDHolder idHolder = (IDHolder)args.Object;
            idHolder.Id = increment.GetNextID(idHolder.GetType());
        }
    };
eventRegistry.Committing +=
    delegate(object sender, CommitEventArgs args)
        {
            increment.StoreState();
        };
```

AutoIncrementExample.cs: use events to assign the ids

```
Dim increment As New AutoIncrement(container)
Dim eventRegistry As IEventRegistry = EventRegistryFactory.ForObjectContainer(container)

AddHandler eventRegistry.Creating, AddressOf increment.HandleCreating
AddHandler eventRegistry.Committing, AddressOf increment.HandleCommiting
```

AutoIncrementExample.vb: use events to assign the ids

The id is hold by the object itself, so you can get it directly.

```
IDHolder idHolder = (IDHolder) obj;
int id = idHolder.Id;
```

AutoIncrementExample.cs: get the id

```
Dim idHolder As IDHolder = DirectCast(obj, IDHolder)
Dim id As Integer = idHolder.Id
```

AutoIncrementExample.vb: get the id

You can get the object you can by a regular query.

```
object instance = container.Query(delegate(IDHolder o) { return o.Id == id; })[0];
```

AutoIncrementExample.cs: get an object by its id

```
Dim instance As Object = container.Query(Function(o As IDHolder) o.Id = id)(0)
```

AutoIncrementExample.vb: get an object by its id

Merging Changes

Merging the changes is the most challenging part when using disconnected objects. Imagine that we have a disconnected object, which contains the changes. We have to store the changes somehow. You cannot simply store the disconnected object, because db4o wouldn't recognize it and store a new object instead of updating the old one. See "Wrong Approach" on page 233

Instead you need to load the existing object and then copy the state from the disconnected object to the loaded object. Basically you traverse the object-graph and copy all changes over. See "Example Merge Changes" on page 234

db4o has no built-in merge support. However there are external libraries which can help you to merge changes, like Automapper .

Wrong Approach

The wrong approach is to try to store disconnected objects. db4o manages object by their object-identity and doesn't recognize objects which have been serialized or loaded by another object container instance. This example shows, that instead of updating the object, db4o will store a new instance of the object.

```
Pilot joe;
using (IObjectContainer container = OpenDatabase())
{
    joe = QueryByName(container, "Joe");
}
// The update on another object
joe.Name = "Joe New";
using (IObjectContainer otherContainer = OpenDatabase())
{
    otherContainer.Store(joe);
}
using (IObjectContainer container = OpenDatabase())
{
    // instead of updating the existing pilot,
    // a new instance was stored.
    IList<Pilot> pilots = container.Query<Pilot>();
    Console.WriteLine("Amount of pilots: " + pilots.Count);
    foreach (Pilot pilot in pilots)
    {
        Console.WriteLine(pilot);
    }
}
```

ObjectIdentityExamples.cs: Update doesn't works when using the different object containers

```
Dim joe As Pilot
Using container As IObjectContainer = OpenDatabase()
    joe = QueryByName(container, "Joe")
End Using
' The update on another object
joe.Name = "Joe New"
Using otherContainer As IObjectContainer = OpenDatabase()
    otherContainer.Store(joe)
End Using
Using container As IObjectContainer = OpenDatabase()
    ' instead of updating the existing pilot,
    ' a new instance was stored.
    Dim pilots As IList(Of Pilot) = container.Query(Of Pilot)()
    Console.WriteLine("Amount of pilots: " & pilots.Count)
    For Each pilot As Pilot In pilots
        Console.WriteLine(pilot)
    Next
End Using
```

ObjectIdentityExamples.vb: Update doesn't works when using the different object containers

So in order to update an object, you need to load and store it in the same object-container. If you cannot do this, you need to merge to object-changes. See "Example Merge Changes" on page 234

```
using (IObjectContainer container = OpenDatabase())
{
    Pilot joe = QueryByName(container, "Joe");
    joe.Name = "Joe New";
    container.Store(joe);
}
using (IObjectContainer container = OpenDatabase())
{
    IList<Pilot> pilots = container.Query<Pilot>();
    Console.WriteLine("Amount of pilots: " + pilots.Count);
    foreach (Pilot pilot in pilots)
    {
        Console.WriteLine(pilot);
    }
}
```

ObjectIdentityExamples.cs: Update works when using the same object container

```
Using container As IObjectContainer = OpenDatabase()
    Dim joe As Pilot = QueryByName(container, "Joe")
    joe.Name = "Joe New"
    container.Store(joe)
End Using
Using container As IObjectContainer = OpenDatabase()
    Dim pilots As IList(Of Pilot) = container.Query(Of Pilot)()
    Console.WriteLine("Amount of pilots: " & pilots.Count)
    For Each pilot As Pilot In pilots
        Console.WriteLine(pilot)
    Next
End Using
```

ObjectIdentityExamples.vb: Update works when using the same object container

Example Merge Changes

This example shows how changes are merged from the disconnected object to the object to update. To do this, traverse the object-graph and copy all value types over. All reference-types are first checked if they're an existing object. If it is,the primitives are copied over, otherwise it's a stored as a new object.

```
using (IObjectContainer container = OpenDatabase())
{
    // first get the object from the database
    Car carInDb = GetCarById(container, disconnectedCar.ObjectId);

    // copy the value-objects (int, long, double, string etc)
    carInDb.Name = disconnectedCar.Name;

    // traverse into the references
    Pilot pilotInDB = carInDb.Pilot;
    Pilot disconnectedPilot = disconnectedCar.Pilot;

    // check if the object is still the same
    if (pilotInDB.ObjectId.Equals(disconnectedPilot.ObjectId))
    {
        // if it is, copy the value-objects
        pilotInDB.Name = disconnectedPilot.Name;
        pilotInDB.Points = disconnectedPilot.Points;
    }
    else
    {
        // otherwise replace the object
        carInDb.Pilot = disconnectedPilot;
    }

    // finally store the changes
    container.Store(pilotInDB);
    container.Store(carInDb);
```

MergeExample.cs: merging

```
Using container As IObjectContainer = OpenDatabase()
    ' first get the object from the database
    Dim carInDb As Car = GetCarById(container, disconnectedCar.ObjectId)

    ' copy the value-objects (int, long, double, string etc)
    carInDb.Name = disconnectedCar.Name

    ' traverse into the references
    Dim pilotInDB As Pilot = carInDb.Pilot
    Dim disconnectedPilot As Pilot = disconnectedCar.Pilot

    ' check if the object is still the same
    If pilotInDB.ObjectId.Equals(disconnectedPilot.ObjectId) Then
        ' if it is, copy the value-objects
        pilotInDB.Name = disconnectedPilot.Name
        pilotInDB.Points = disconnectedPilot.Points
    Else
        ' otherwise replace the object
        carInDb.Pilot = disconnectedPilot
    End If

    ' finally store the changes
    container.Store(pilotInDB)
```

MergeExample.vb: merging

You can use reflection to automated this process. You can also use existing libraries like Automapper which help you to do this.

Web Environment

db4o runs perfectly well in a web environment. It can be used to build your web-application.

In most web-application multiple concurrent requests are processes. Normally you want to isolate each request from another. You can use db4o transactions to archive this isolation. See "Isolation in Web-Applications" on page 236

In most web-applications a object is only alive during a request. So you have to identify objects across requests. Therefore you need to add an additional id to your object. There are different possibilities for this. See "Disconnected Objects" on page 227

When you run in a web-environment, you often have stricter security limitations. Take a look at the security requirements. See "ASP.NET Security" on page 239

Take a look how you create a object-container for each request. See "ASP.NET Request Example" on page 237

You might run into trouble because of ASP.NET assembly naming policy. See "ASP.NET Assembly Naming" on page 239

Take a look a small example ASP.NET MVC application, which includes all these concerns. See "ASP.NET MVC Example" on page 240

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

Isolation in Web-Applications

In most web-application multiple concurrent request are processes. You want to isolate the request from each other. db4o supports transactions, which are perfect for this kind of isolation. Each unit of work gets its own transaction, for example each request. You can create a new session object container for this purpose. Such a session-container brings its own transaction and reference-system. This ensures that the session container is isolated from other operations on the database.

```
using (IObjectContainer rootContainer = Db4oEmbedded.OpenFile(DatabaseFileName))
{
    // open the db4o-session. For example at the beginning for a web-request
    using (IObjectContainer session = rootContainer.Ext().OpenSession())
    {
        // do the operations on the session-container
        session.Store(new Person("Joe"));
    }
}
```

Db4oSessions.cs: Session object container

```
Using rootContainer As IObjectContainer = Db4oEmbedded.OpenFile(DatabaseFileName)
    ' open the db4o-session. For example at the beginning for a web-request
    Using session As IObjectContainer = rootContainer.Ext().OpenSession()
        ' do the operations on the session-container
        session.Store(New Person("Joe"))
    End Using
End Using
```

Db4oSessions.vb: Session object container

Or you can use embedded clients when your embedded clients.

```
using (IObjectServer server = Db4oClientServer.OpenServer(DatabaseFileName, 0))
{
    // open the db4o-embedded client. For example at the beginning for a web-request
    using (IObjectContainer container = server.OpenClient())
    {
        // do the operations on the session-container
        container.Store(new Person("Joe"));
    }
}
```

Db4oSessions.cs: Embedded client

```
Using server As IObjectServer = Db4oClientServer.OpenServer(DatabaseFileName, 0)
    ' open the db4o-embedded client. For example at the beginning for a web-request
    Using container As IObjectContainer = server.OpenClient()
        ' do the operations on the session-container
        container.Store(New Person("Joe"))
    End Using
End Using
```

Db4oSessions.vb: Embedded client

ASP.NET Request Example

This example demonstrates how you can provide an object container for each request, to ensure that the requests are isolated from each other. For this we use session object containers. See "Isolation in Web-Applications" on page 236

First, create a new class which implements the IHttpModule-interface. Then register this interface in the Web.config configuration:

```
<httpModules>
  <add name="Db4oDatabase" type="Db4oDoc.WebApp.Infrastructure.Db4oProvider" />
  <!-- Other, existing modules-->
  <add name="ScriptModule" type="System.Web.Handlers.ScriptModule, System.Web.Extensions, Version=3.5.0.0, Cultur
  <add name="UrlRoutingModule" type="System.Web.Routing.UrlRoutingModule, System.Web.Routing, Version=3.5.0.0, Cu
</httpModules>
```

Web.config: Add the db4o-http-handler

Additionally add a key where you specify the path to the database-file. Make sure that you have the data folder inside you web-site directory. The ASP.NET user should have enough rights to create and modify files inside this folder. You should also make sure that the data folder is not accessible to anonymous user; otherwise web-server visitors will be able to download your database:

```
<appSettings>
  <add key="DatabaseFileName" value="~/App_Data/database.db4o"/>
</appSettings>
```

Web.config: Database path in the configuration

After that, implement the initializing sequence. When the server starts, it starts up db4o and adds the database to the application-context:

```
public void Init(HttpApplication context)
{
    if (null == HttpContext.Current.Application[DataBaseInstance])
    {
        HttpContext.Current.Application[DataBaseInstance] = OpenDatabase();
    }
    RegisterSessionCreation(context);
}

private IEmbeddedObjectContainer OpenDatabase()
{
    string relativePath = ConfigurationSettings.AppSettings["DatabaseFileName"];
    string filePath = HttpContext.Current.Server.MapPath(relativePath);
    return Db4oEmbedded.OpenFile(filePath);
}
```

Db4oProvider.cs: open database when the application starts

On the dispose-method add the shutdown code:

```
public void Dispose()
{
    IDisposable toDispose = HttpContext.Current.Application[DataBaseInstance] as IDisposable;
    if (null != toDispose)
    {
        toDispose.Dispose();
    }
}
```

Db4oProvider.cs: close the database when the application shuts down

Now it's time to ensure that each requests has its own object container. To archive that, use the open-session-method. Register for the request-begin and request-end events. In begin session event, open a new db4o session. In the end request event, close that session.

```
private void RegisterSessionCreation(HttpApplication httpApplication)
{
    httpApplication.BeginRequest += OpenSession;
    httpApplication.EndRequest += CloseSession;
}

private void OpenSession(object sender, EventArgs e)
{
    IObjectContainer container =
        (IObjectContainer)HttpContext.Current.Application[DataBaseInstance];
    IObjectContainer session = container.Ext().OpenSession();
    HttpContext.Current.Items[SessionKey] = session;
}

private void CloseSession(object sender, EventArgs e)
{
    IObjectContainer session = (IObjectContainer)HttpContext.Current.Items[SessionKey];
    session.Dispose();
}
```

Db4oProvider.cs: A object container per request

Finally add a static property which gives access to the request-instance:

```
public static IObjectContainer Database
{
    get { return (IObjectContainer)HttpContext.Current.Items[SessionKey]; }
}
```

Db4oProvider.cs: provide access to the database

That's it. Of course there are other ways to archive the same result. For example you can use dependency injection frameworks like Microsoft's Unity to enforce the request scope.

ASP.NET Security

It is characteristic to web-applications that 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. ASP.NET 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 by installing it in GAC.

   If full trust is not a suitable solution for, you can check the minimum security permissions that db4o-assembly needs to operate using permcalc.exe tool from your Visual Studio installation.

   `PermCalc.exe -Sandbox Db4objects.Db4o.dll`

   ```
   sandbox.PermCalc.xml
   <?xml version="1.0"?>
   <Sandbox>
     <PermissionSet version="1"
   class="System.Security.PermissionSet">
       <IPermission version="1"
   class="System.Security.Permissions.FileIOPermission, mscorlib,
   Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
   Read="*AllFiles*" PathDiscovery="*AllFiles*" />
       <IPermission version="1"
   class="System.Security.Permissions.ReflectionPermission, mscorlib,
   Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
   Flags="MemberAccess" />
       <IPermission version="1"
   class="System.Security.Permissions.SecurityPermission, mscorlib,
   Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
   Flags="UnmanagedCode, Execution, ControlEvidence" />
       <IPermission Window="SafeSubWindows" Clipboard="OwnClipboard"
   version="1" class="System.Security.Permissions.UIPermission, mscorlib,
   Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
       <IPermission version="1"
   class="System.Security.Permissions.KeyContainerPermission, mscorlib,
   Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
   Unrestricted="true" />
     </PermissionSet>
   </Sandbox>
   ```

   Consult with your web-server administrator to grant these permissions. permcalc.xml

ASP.NET Assembly Naming

If you are creating your application in ASP.NET 2.0 you should take into consideration 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 has changed, 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
  ```

  Microsoft has published a how-to for using the aspnet_compiler to generate fixed names for ASP.NET assemblies.
- 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.

ASP.NET MVC Example

This example is a tiny CRUD application which shows how to use db4o in ASP.NET MVC 2.0.

Managing Object Containers

This example uses the code from the request-example to create a object container for each request. On each new request a object container is opened. Then all operations are done on this container. When the request ends, the container is disposed.

Object Identification

This example uses a GUID for each object to identify it across requests. Persisted objects inherit from the IDHolder class which contains the id-field. Take a look at alternatives for ids. See "Comparison Of Different IDs" on page 228

Using db4o

You can use db4o as expected. Just use the request-container:

```
public ActionResult Index()
{
    IList<Pilot> allPilots = Db4oProvider.Database.Query<Pilot>();
    return View(allPilots);
}
```

HomeController.cs: Listing all pilots on the index

Using IDs

Now the ids can be used in the views and actions to identify objects. For example in a list-view you use the ids for the action-links:

```
<% foreach (var pilot in Model) { %>

    <tr>
        <td>
            <%= Html.ActionLink("Edit", "Edit", new { id=pilot.ID }) %> |
            <%= Html.ActionLink("Delete", "Delete", new { id = pilot.ID })%>
        </td>
        <td>
            <%= Html.Encode(pilot.Name) %>
        </td>
        <td>
            <%= Html.Encode(pilot.Points) %>
        </td>
    </tr>

<% } %>
```

Index.aspx: In the view use the ids to identify the objects

Another location where the ids are used is in the actions. For example when you need to store changes. First we get a object which contains all changes. Then we copy all changes to the existing object in the database and finally store it. See "Merging Changes" on page 232

```
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult Edit(Guid id, Pilot editedPilot)
{
    Pilot databasePilot = GetPilotById(id);
    databasePilot.Name = editedPilot.Name;
    databasePilot.Points = editedPilot.Points;
    Db4oProvider.Database.Store(databasePilot);

    return RedirectToAction("Index");
}
```

HomeController.cs: update the object

Automatic Properties

Automatic properties simply allow a user to skip the step of introducing private field variables when the property simply sets and gets the field value:

```
class Person
{
    public string Name { get; set; }
}
```

AutoProperties.cs: Auto property

```
Public Class Person
    Public Property Name() As String
End Class
```

AutoProperties.vb: Auto property

Behind the scenes the compiler creates a field and the regular property code. The field-name depends on the compiler. For C# the field-name of auto-property is <PropertyName>k__BackingField. For example the property FirstName will be stored in the field <FirstName>k__BackingField. For VB.Net the field-

_PropertyName. For example the property FirstName will be stored in the field _FirstName.

```
public class Person
{
    // the name is actually <Name>k__BackingField
    // but you cannot express that in C# code
    private string name;

    public string Name
    {
        get { return name; }
        set { name = value; }
    }
}
```

AutoProperties.cs: Auto property behind the scenes

```
Public Class Person
    Private _Name As String

    Public Property Name() As String
        Get
            Return _Name
        End Get
        Set(ByVal value As String)
            _Name = value
        End Set
    End Property
End Class
```

AutoProperties.vb: Auto property behind the scenes

Issues With Auto Properties

Note that db4o always uses the field name for its configuration. This also applies to auto-properties. You need to use the field-name to specify index etc.

For C# the name of the backing field is not specified and a compiler implementation detail. Therefore it's possible that it actually changes in the future.

For example to index the auto property Name:

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof(Person)).ObjectField("<Name>k__BackingField").Indexed(true);
```

AutoProperties.cs: Configure auto properties

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).ObjectField("_Name").Indexed(True)
```

AutoProperties.vb: Configure auto properties

WCF Data Services (aka ADO.NET Data Services)

The WCF Data Services allows you to easily expose your data as a web service. db4o supports this framework. Look up the MSDN documentation on WCF Data Service for more details about it and its features.

The are two things which the data service need. An IQueryable implementation and a IUpdatable implementation. db4o provides both. For this you need to reference the Db4objects.Db4o.Linq.dll and

Db4objects.Db4o.Data.Services.dll. The first one contains the LINQ-provider, the second contains the IUpdatable.

Preparing The Persistent Classes

The first thing you need to do is to add a key to your objects. And you need specify at least one key for each object.

```
[DataServiceKey("Email")]
public class Person
{
    private string name;
    private string email;

    // Note that a default constructor is required for WCF
    public Person()
    {
    }

    public Person(string email, string name)
    {
        this.email = email;
        this.name = name;
    }

    public string Name
    {
        get { return name; }
        set { name = value; }
    }

    public string Email
    {
        get { return email; }
        set { email = value; }
    }

}
```

Person.cs: Add at least one key

```
[DataServiceKey("TeamName")]
public class Team
{
    private string teamName;
    private string motivation;
    private IList<Person> members = new List<Person>();

    public Team()
    {
    }

    public Team(string teamName)
    {
        this.teamName = teamName;
    }

    public string TeamName
    {
        get { return teamName; }
        set { teamName = value; }
    }

    public string Motivation
    {
        get { return motivation; }
        set { motivation = value; }
    }

    public IList<Person> Members
    {
        get { return members; }
        set { members = value; }
    }
}
```

Team.cs: The Team has also a key

Create A Context

After that, you can build a data context. For this, inherit from the Db4oDataContext. You need to overwrite the OpenSession() to provide the right object container. A good practice is to use a object container per request. For example you can use the OpenSession-operation for creating a container per request. Take also a look how you can provide a object container for each request.

After that, you can add operations and properties you want to expose. Remember to use the IQueryable interface to expose query options to the client.

```
public class TeamDataContext : Db4oDataContext
{
    // Provide access to your data via properties
    public IQueryable<Person> Persons
    {
        get { return Container.AsQueryable<Person>(); }
    }
    public IQueryable<Team> Teams
    {
        get { return Container.AsQueryable<Team>(); }
    }



    /// You need to implement the open-session and return a object container
    /// The best practise is to use a separate object-container per request.
    /// For example use the
    /// <see cref="IObjectContainer"/>.<see cref="IObjectContainer.Ext"/>.<see cref="IExtObjectContainer.OpenSess
    /// to open session-containers for each request.
    protected override IObjectContainer OpenSession()
    {
        return Db4oEmbedded.NewSession();
    }

}
```

TeamDataContext.cs: An concrete context

Create the Service

The last step is to actually create a service. Visual Studio can assist to do this. Right click on the Project, choose 'Add'-> 'New Item'. Then Choose the 'WCF Data Service'. In older releases it's called 'ADO.NET Data Service'.

After that, you can rename the created classes and parameterize it with the previously created context. Then you need to specify which operation are allowed. Read more about how the configure the allowed operation in the data service documentation.

```
public class TeamDataService : DataService<TeamDataContext>
{
    public static void InitializeService(DataServiceConfiguration config)
    {
        config.SetEntitySetAccessRule("*", EntitySetRights.All);
        config.SetServiceOperationAccessRule("*", ServiceOperationRights.All);
    }
}
```

TeamDataService.svc.cs: Build the concrete service

Consuming The Service

Now you can startup this service and you're ready to consume it. For example you can create a simple console application. There you import the service. Right click on the project choose 'Add Service Reference'. Then point the assistant to the URL of the running web service. After that you can use the service.

```
var serviceURL = new Uri("http://localhost:8080/TeamDataService.svc");
var serviceContext = new TeamDataContext(serviceURL);

var teams = serviceContext.Teams;
foreach (var team in teams)
{
    Console.Out.WriteLine(team.TeamName);
}
```

SimpleConsoleClient.cs: Now the service can be used

Silverlight

db4o runs with some limitations on Microsoft's Silverlight. You can find the Silverlight assemblies in the db4o distribution, in the folder 'bin/silverlight'.

Feature Set

Currently only the core functionality is supported on Silverlight. The db4o core and the db4o LINQ provider run on Silverlight. Other features like the client-server mode are not supported.

Add Silverlight Support

To run db4o on Silverlight you need to use the Silverlight assemblies which are in the db4o distribution. Additionally you need to configure the Silverlight support in the db4o configuration. This also configured the isolated storage.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.AddConfigurationItem(new SilverlightSupport());

IObjectContainer container = Db4oEmbedded.OpenFile(configuration, "database.db4o");
```

IOExamples.cs: Add Silverlight support

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.AddConfigurationItem(New SilverlightSupport())

Dim container As IObjectContainer = Db4oEmbedded.OpenFile(configuration, "database.db4o")
```

IOExamples.vb: Add Silverlight support

Persistent Object Limitations

Due to the restrictions of the Silverlight sandbox, it's impossible to inspect the private fields of an object. This means that the field of persistent objects need to be public!

Also auto-properties cannot be stored. Auto properties use a regular, compiler generated private field to hold the data. Since db4o cannot access private fields, it also cannot access the data of auto-properties.

```
public class Person
{
    public string FirstName;
    public string SirName;
}
```

Person.cs: fields need to be public to be persisted

```
Public Class Person
    Public FirstName As String
    Public SirName As String
End Class
```

Person.vb: fields need to be public to be persisted

Queries

You can use LINQ to query your objects in Silverlight. Note that the Silverlight version uses the Mono.Cecil.dll and the Cecil.FlowAnalysis.dll assembly for the LINQ-implementation.

```
var persons = from Person p in container
              where p.FirstName.Contains("Roman")
              select p;

foreach (Person person in persons)
{
    // do something with the person
}
```

QueriesInSilverlight.cs: Queries in Silverlight

```
Dim persons = From p As Person In container _
              Where p.FirstName.Contains("Roman") _
              Select p
For Each p As Person In persons
    ' do something with the person
Next
```

QueriesInSilverlight.vb: Queries in Silverlight

Isolated Storage

By default you cannot access the file system in Silverlight, but only a special isolated storage. Therefore you need to use the isolated storage when you're configuring the IO system. By default this storage is limited to 1 MByte of data. You can increase it with the users consent.

When the Silverlight application runs outside the browser with enough privileges you can use the regular storage. However by default the isolated storage should be used.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.File.Storage = new IsolatedStorageStorage();

IObjectContainer container = Db4oEmbedded.OpenFile(configuration, "database.db4o");
```

IOExamples.cs: use the isolated storage on silverlight

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.File.Storage = New IsolatedStorageStorage()

Dim container As IObjectContainer = Db4oEmbedded.OpenFile(configuration, "database.db4o")
```

IOExamples.vb: use the isolated storage on silverlight

Class Name Format In .NET

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:

```
Test.cs
/**//* Copyright (C) 2004 - 2007 Versant Inc. http://www.db4o.com */
namespace Db4objects.Db4odoc.ClassNameFormat
 {
    class Test
     {
        public override string ToString()
         {
            return "Test";
        }
    }
}
```

```
Test.vb
' Copyright (C) 2004 - 2007 Versant Inc. http://www.db4o.com
Namespace Db4objects.Db4odoc.ClassNameFormat

    Class TestClass Test

        Public Overloads Overrides Function ToString() As String
            Return "Test"
        End Function
    End Class
End Namespace
```

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

```
ClassNameExample1.cs: StoreObjects
private static void StoreObjects()
         {
            File.Delete(Db4oFileName);
            IObjectContainer container = Db4oFactory.OpenFile(Db4oFileName);
            try
             {
                // Store a simple class to the database
                Test test = new Test();
                container.Store(test);
            }
            finally
             {
                container.Commit();
            }
        }
```

```
ClassNameExample1.vb: StoreObjects
Private Shared Sub StoreObjects()
            File.Delete(Db4oFileName)
            Dim container As IObjectContainer = _
Db4oFactory.OpenFile(Db4oFileName)
```

```
            Try
                ' Store a simple class to the database
                Dim test As Test = New Test
                container.Store(test)
            Finally
                container.Commit()
            End Try
        End Sub
```

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:

```
ClassNameExample2.cs: CheckDatabase
public static void CheckDatabase()
         {
            IObjectContainer container = Db4oFactory.OpenFile(Db4oFileName);
            try
             {
                // Read db4o contents from another application
                IObjectSet result = container.QueryByExample(typeof(Test));
                ListResult(result);
                // Check what classes are actualy stored in the database
                IStoredClass[] storedClasses = container.Ext().StoredClasses();
                foreach (IStoredClass storedClass in storedClasses)
                 {
                    System.Console.WriteLine("Stored class: " + storedClass.GetName());
                }
            }
            finally
             {
                container.Commit();
            }
        }
```

```
ClassNameExample2.vb: CheckDatabase
Private Shared Sub CheckDatabase()
            Dim container As IObjectContainer = Db4oFactory.OpenFile(Db4oFileName)
            Try
                ' Read db4o contents from another application
                Dim result As IObjectSet = container.QueryByExample(GetType(Test))
                ListResult(result)
                ' Check what classes are actualy stored in the database
                Dim storedClasses As IStoredClass() = container.Ext.StoredClasses
                Dim storedClass As IStoredClass
                For Each storedClass In storedClasses
                    System.Console.WriteLine("Stored class: " + storedClass.GetName)
                Next
            Finally
                container.Commit()
            End Try
        End Sub
```

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 work-arounds:

- 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.

db4o On Mono

Currently db4o is only build for .NET and tested on .NET. However it's possible to use db4o on Mono. Some features are not supported on Mono: The data services and the monitoring-support.

In the regular build, db4o also uses some API's which are not available on Mono. To avoid that, you need to build db4o from source and define the build constant 'Mono'. The source is in the 'src'-folder of the db4o distribution.

Build For Mono With Regular .NET

The easiest way to build db4o for Mono is actually to use the regular MSBuild and just define the 'Mono' built-time constant. Open the Windows console and navigate to the db4o source folder. Then you can run the MSBuild command:

```
MSBuild /property:Configuration=Release /property:DefineConstants="Mono" /target:Rebuild Db4o-2008.sln
```

The MSBuild-tool is in the C:\Windows\Microsoft.NET\Framework\{version}\-folder, where the {version} represents the .NET version. For example 'v3.5' for .NET 3.5.

The different solution-files (*.sln) represent different versions of db4o. For example:

Solution-File	db4o Version
Db4o-2005.sln	db4o for .NET 2.0 / Mono equivalent
Db4o-2008.sln	db4o for .NET 3.5 / Mono equivalent
Db4o-2010.sln	db4o for .NET 4.0 / Mono equivalent
Db4o-**CF**[1]-2005.sln	db4o for .NET 2.0 Compact Framework
Db4o-CF-2008.sln	db4o for .NET 3.5 Compact Framework
Db4o-Silverlight-2008.sln	db4o for Silverlight 3.0 / Moonlight 3.0
Db4o-Silverlight-2010.sln	db4o for Silverlight 3.0 / Moonlight 3.0

Note that you should use the MSBuild-tool for the right version. Use the MSBuild 3.5 version to build the Db4o-2008.sln Solution.

The result of the builds are in the bin/Release-folders of the different subprojects.

Build With Mono

In Mono you can use the xbuild-command instead of the MSBuild. The xbuild is part of the mono distribution. If you're using a Mono version which was packaged with your Linux-distribution, you maybe need to install is separately.

In principal the command is exactly the same as with MSBuild and the same rules apply. The different solution-files represent the different db4o versions. See on the table above.

```
xbuild /property:Configuration=Release /property:DefineConstants="Mono" /target:Rebuild Db4o-2008.sln
```

When you build with Mono and xbuild, you probably get some errors.This errors should be only in following projects.

[1].NET Compact Framework

- Db4o-Data-Services: This is not available in Mono
- Db4o-Test-Projects: The db4o Test-Suite is not ported to Mono. For example it doesn't exclude Test-Cases which only apply on Windows.

The rest of db4o can be built on Mono. The result of the builds are in the bin/Release-folders of the different subproject's.

db4o on .NET Platforms

All .NET

- .NET version of db4o uses Pascal case for method names
- Root namespace is Db4objects.Db4o
- All namespaces start with upper case letter
- Interface names have an I prefix
- .NET Reflection mechanism adds assembly name to class definition. If you use db4o database with two 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.
- Enumerations are treated as integer types

Please, refer to Security Requirements On .NET Platform for further information.

.NET Compact Framwork

Due to some platform limitations nativ queries cannot be optimized at runtime. Use LINQ as a better alternative. Or you can use the Db4oTool.exe command line utility on their assemblies prior to deploying them.

Security Requirements On .NET Platform

db4o requires certain security permissions to be granted for successful execution. It is important to know these permission requirements in a not fully trusted environment. .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 Visual Studio installation:

[Visual Studio Home]\SDK\v2.0\Bin

The following command line will calculate the minimum security permissions for Db4objects.Db4o.dll and will safe them in xml format Sandbox.Permcalc.xml document:

```
PermCalc.exe -sandbox Db4objects.Db4o.dll
```

The output should look like this:

```
Db4o.Xml
<?xml version="1.0"?>
<Sandbox>
  <PermissionSet version="1" class="System.Security.PermissionSet">
    <IPermission version="1"
class="System.Security.Permissions.FileIOPermission, mscorlib, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089"
Read="*AllFiles*" PathDiscovery="*AllFiles*" />
    <IPermission version="1"
class="System.Security.Permissions.ReflectionPermission, mscorlib, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089" Flags="MemberAccess" />
    <IPermission version="1" class="System.Security.Permissions.SecurityPermission,
mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
Flags="UnmanagedCode, Execution, ControlEvidence, SerializationFormatter,
ControlAppDomain" />
    <IPermission Window="SafeSubWindows" Clipboard="OwnClipboard" version="1"
class="System.Security.Permissions.UIPermission, mscorlib, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089" />
    <IPermission version="1"
class="System.Security.Permissions.KeyContainerPermission, mscorlib,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
Unrestricted="true" />
  </PermissionSet>
</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:	
UnmanagedCode	Unmanaged code is used internally for file access and socket operations.
Execution	Permission for the code to run. Without this permission, managed code will not be executed.
ControlEvidence	Is required internally to make use of some .NET functionality
SerializationFormatter	Used to utilize serialization services (formatters)
ControlAppDomain	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

Xml Import-Export In .NET

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:

```
SerializeExample.cs: SetObjects
private static void SetObjects()
     {
      File.Delete(Db4oFileName);
      IObjectContainer db = Db4oFactory.OpenFile(Db4oFileName);
      try
       {
        Car car = new Car("BMW", new Pilot("Rubens Barrichello"));
        db.Store(car);
        car = new Car("Ferrari", new Pilot("Michael Schumacher"));
        db.Store(car);
      }
      finally
       {
        db.Close();
      }
    }
```

```
SerializeExample.vb: SetObjects
Private Shared Sub SetObjects()
            File.Delete(Db4oFileName)
            Dim db As IObjectContainer = Db4oFactory.OpenFile(Db4oFileName)
            Try
                Dim car As Car = New Car("BMW", New Pilot("Rubens Barrichello"))
                db.Store(car)
                car = New Car("Ferrari", New Pilot("Michael Schumacher"))
                db.Store(car)
            Finally
                db.Close()
            End Try
        End Sub
```

We will save the database to XML file "formula1.xml":

```
SerializeExample.cs: ExportToXml
private static void ExportToXml()
     {
      XmlSerializer carSerializer = new XmlSerializer(typeof(Car[]));
      StreamWriter xmlWriter = new StreamWriter(XmlFileName);
      IObjectContainer db = Db4oFactory.OpenFile(Db4oFileName);
      try
       {
        IObjectSet result = db.QueryByExample(typeof(Car));
        Car[] cars = new Car[result.Size()];
        for (int i = 0; i < result.Size(); i++)
         {
          Car car = (Car)result[i];
          cars.SetValue(car,i);
        }
        carSerializer.Serialize(xmlWriter, cars);
        xmlWriter.Close();
      }
      finally
       {
        db.Close();
      }
    }
```

```
SerializeExample.vb: ExportToXml
Private Shared Sub ExportToXml()
            Dim carSerializer As XmlSerializer = _
New XmlSerializer(GetType(Car()))
            Dim xmlWriter As StreamWriter = New StreamWriter(XmlFileName)
            Dim db As IObjectContainer = Db4oFactory. _
OpenFile(Db4oFileName)
            Try
                Dim result As IObjectSet = db.QueryByExample(GetType(Car))
                Dim cars() As Car = New Car(result.Size()) {}
                Dim i As Integer
                For i = 0 To result.Size() - 1 Step i + 1
                    Dim car As Car = CType(result(i), Car)
                    cars.SetValue(car, i)
                Next
                carSerializer.Serialize(xmlWriter, cars)
                xmlWriter.Close()
            Finally
                db.Close()
            End Try
        End Sub
```

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:

```
SerializeExample.cs: ImportFromXml
private static void ImportFromXml()
     {
      File.Delete(Db4oFileName);
      XmlSerializer carSerializer = new XmlSerializer(typeof(Car[]));
      FileStream xmlFileStream = new FileStream(XmlFileName, FileMode.Open);
      Car[] cars = (Car[])carSerializer.Deserialize(xmlFileStream);
      IObjectContainer db;
      foreach (Car car in cars)
       {
        db = Db4oFactory.OpenFile(Db4oFileName);
        try
         {
          Car newCar = (Car)car;
          db.Store(newCar);
        }
        finally
         {
          db.Close();
        }
      }
      db = Db4oFactory.OpenFile(Db4oFileName);
      try
       {
        IObjectSet result = db.QueryByExample(typeof(Pilot));
        ListResult(result);
        result = db.QueryByExample(typeof(Car));
        ListResult(result);
      }
      finally
       {
        db.Close();
      }
    }
```

```
SerializeExample.vb: ImportFromXml
Private Shared Sub ImportFromXml()
            File.Delete(Db4oFileName)
            Dim carSerializer As XmlSerializer = New _
XmlSerializer(GetType(Car()))
            Dim xmlFileStream As FileStream = New _
FileStream(XmlFileName, FileMode.Open)
            Dim cars() As Car = CType(carSerializer. _
Deserialize(xmlFileStream), Car())
            Dim db As IObjectContainer
            Dim car As Car
            For Each car In cars
                db = Db4oFactory.OpenFile(Db4oFileName)
                Try
                    Dim newCar As Car = CType(car, Car)
                    db.Store(newCar)
                Finally
                    db.Close()
                End Try
            Next
            db = Db4oFactory.OpenFile(Db4oFileName)
            Try
                Dim result As IObjectSet = db.QueryByExample(GetType(Pilot))
                ListResult(result)
                result = db.QueryByExample(GetType(Car))
                ListResult(result)
            Finally
                db.Close()
            End Try
        End Sub
```

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.

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
- Selective Persistence
- Indexing
- Performance Hints
- Debugging db4o
- Diagnostics
- Native Query Optimization
- Utility Methods

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
- Delete Performance
- Update Performance
- Query Performance

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.cs: Item
public class Item
     {

        public String _name;
        public Item _child;

        public Item()
         {
        }

        public Item(String name, Item child)
         {
            _name = name;
            _child = child;
        }
    }
```

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

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 storage:

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
MemoryStorage memory = new MemoryStorage();
configuration.File.Storage = memory;
IObjectContainer container = Db4oEmbedded.OpenFile(configuration, "database.db4o");
```

IOConfigurationExamples.cs: Using memory-storage

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
Dim memory As New MemoryStorage()
configuration.File.Storage = memory
```

IOConfigurationExamples.vb: Using memory-storage

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.cs: RunClientServerTest
private void RunClientServerTest()
         {
            ConfigureClientServer();
            Init();
            Clean();
            System.Console.WriteLine("Storing " + _count + " objects of depth "
+ _depth + " locally:");
            Open();
            Store();
            Close();

            InitForClientServer();
            Clean();
            System.Console.WriteLine("Storing " + _count + " objects of depth "
+ _depth + " remotely:");
            Open();
            Store();
            Close();
        }
```

```
InsertPerformanceBenchmark.cs: ConfigureClientServer
private void ConfigureClientServer()
         {
            IConfiguration config = Db4oFactory.Configure();
            config.LockDatabaseFile(false);
            config.WeakReferences(false);
            config.FlushFileBuffers(false);
            config.ClientServer().SingleThreadedClient(true);
        }
```

```
InsertPerformanceBenchmark.cs: Init
private void Init()
         {
            _count = 10000;
            _depth = 3;
            _isClientServer = false;
        }
```

```
InsertPerformanceBenchmark.cs: InitForClientServer
private void InitForClientServer()
         {
            _count = 10000;
            _depth = 3;
            _isClientServer = true;
        }
```

```
InsertPerformanceBenchmark.cs: Store
private void Store()
         {
            StartTimer();
            for (int i = 0; i < _count; i++)
             {
                Item item = new Item("load", null);
                for (int j = 1; j < _depth; j++)
                 {
                    item = new Item("load", item);
                }
                objectContainer.Store(item);
            }
            objectContainer.Commit();
            StopTimer("Store " + TotalObjects() + " objects");
        }
```

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.

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.cs: RunCommitTest
private void RunCommitTest()
         {

            ConfigureForCommitTest();
            InitForCommitTest();

            Clean();
            System.Console.WriteLine("Storing objects as a bulk:");
            Open();
            Store();
            Close();

            Clean();
            System.Console.WriteLine("Storing objects with commit after each "
+ _commitInterval + " objects:");
            Open();
            StoreWithCommit();
            Close();
        }
```

```
InsertPerformanceBenchmark.cs: ConfigureForCommitTest
private void ConfigureForCommitTest()
         {
            IConfiguration config = Db4oFactory.Configure();
            config.LockDatabaseFile(false);
            config.WeakReferences(false);
            // FlushFileBuffers should be set to true to ensure that
            // the commit information is physically written
            // and in the correct order
            config.FlushFileBuffers(false);
        }
```

```
InsertPerformanceBenchmark.cs: InitForCommitTest
private void InitForCommitTest()
         {
            _count = 100000;
            _commitInterval = 10000;
            _depth = 3;
            _isClientServer = false;
        }
```

```
InsertPerformanceBenchmark.cs: Store
private void Store()
         {
            StartTimer();
            for (int i = 0; i < _count; i++)
```

```
              {
                 Item item = new Item("load", null);
                 for (int j = 1; j < _depth; j++)
                  {
                     item = new Item("load", item);
                 }
                 objectContainer.Store(item);
             }
             objectContainer.Commit();
             StopTimer("Store " + TotalObjects() + " objects");
         }
```

```
InsertPerformanceBenchmark.cs: StoreWithCommit
private void StoreWithCommit()
          {
             StartTimer();
             int k = 0;
             while (k < _count)
              {
                 for (int i = 0; i < _commitInterval; i++)
                  {
                     Item item = new Item("load", null);
                     k++;
                     for (int j = 1; j < _depth; j++)
                      {
                         item = new Item("load", item);
                     }
                     objectContainer.Store(item);
                 }
                 objectContainer.Commit();
             }
             objectContainer.Commit();
             StopTimer("Store " + TotalObjects() + " objects");
         }
```

The following results were achieved for the testing configuration:

.NET:

Storing objects as a bulk:

Store 300000 objects: 17748ms

Storing objects with commit after each 10000 objects:

Store 300000 objects: 18163ms

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.cs: RunDifferentObjectsTest
private void RunDifferentObjectsTest()
          {
             Configure();
             Init();
             System.Console.WriteLine("Storing " + _count + " objects with "
+ _depth + " levels of embedded objects:");

             Clean();
```

```
            System.Console.WriteLine(" - primitive object with int field");
            Open();
            StoreSimplest();
            Close();

            Open();
            System.Console.WriteLine(" - object with String field");
            Store();
            Close();

            Clean();
            Open();
            System.Console.WriteLine(" - object with StringBuilder field");
            StoreWithStringBuilder();
            Close();

            Clean();
            Open();
            System.Console.WriteLine(" - object with int array field");
            StoreWithArray();
            Close();

            Clean();
            Open();
            System.Console.WriteLine(" - object with ArrayList field");
            StoreWithArrayList();
            Close();
        }
```

```
InsertPerformanceBenchmark.cs: Configure
private void Configure()
         {
            IConfiguration config = Db4oFactory.Configure();
            config.LockDatabaseFile(false);
            config.WeakReferences(false);
            config.Io(new MemoryIoAdapter());
            config.FlushFileBuffers(false);
        }
```

```
InsertPerformanceBenchmark.cs: Init
private void Init()
         {
            _count = 10000;
            _depth = 3;
            _isClientServer = false;
        }
```

```
InsertPerformanceBenchmark.cs: StoreSimplest
private void StoreSimplest()
         {
            StartTimer();
            for (int i = 0; i < _count; i++)
             {
                SimplestItem item = new SimplestItem(i, null);
                for (int j = 1; j < _depth; j++)
                 {
                    item = new SimplestItem(i, item);
                }
                objectContainer.Store(item);
            }
            objectContainer.Commit();
            StopTimer("Store " + TotalObjects() + " objects");
        }
```

```
InsertPerformanceBenchmark.cs: Store
private void Store()
         {
            StartTimer();
            for (int i = 0; i < _count; i++)
             {
                Item item = new Item("load", null);
                for (int j = 1; j < _depth; j++)
                 {
                    item = new Item("load", item);
                }
                objectContainer.Store(item);
            }
            objectContainer.Commit();
            StopTimer("Store " + TotalObjects() + " objects");
        }
```

```
InsertPerformanceBenchmark.cs: StoreWithStringBuilder
private void StoreWithStringBuilder()
         {
            StartTimer();
            for (int i = 0; i < _count; i++)
             {
                ItemWithStringBuilder item = new ItemWithStringBuilder(
new StringBuilder("load"), null);
                for (int j = 1; j < _depth; j++)
                 {
                    item = new ItemWithStringBuilder(new StringBuilder("load"), item);
                }
                objectContainer.Store(item);
            }
            objectContainer.Commit();
            StopTimer("Store " + TotalObjects() + " objects");
        }
```

```
InsertPerformanceBenchmark.cs: StoreWithArray
private void StoreWithArray()
         {
            StartTimer();
            int[] array = new int[]  { 1, 2, 3, 4 };
            for (int i = 0; i < _count; i++)
             {
                int[] id = new int[]  { 1, 2, 3, 4 };
                ItemWithArray item = new ItemWithArray(id, null);
                for (int j = 1; j < _depth; j++)
                 {
                    int[] id1 = new int[]  { 1, 2, 3, 4 };
                    item = new ItemWithArray(id1, item);
                }
                objectContainer.Store(item);
            }
            objectContainer.Commit();
            StopTimer("Store " + TotalObjects() + " objects");
        }
```

```
InsertPerformanceBenchmark.cs: StoreWithArrayList
private void StoreWithArrayList()
         {
            StartTimer();
            ArrayList idList = new ArrayList();
            idList.Add(1);
            idList.Add(2);
            idList.Add(3);
            idList.Add(4);
            for (int i = 0; i < _count; i++)
             {
                ArrayList ids = new ArrayList();
                ids.AddRange(idList);
                ItemWithArrayList item = new ItemWithArrayList(ids, null);
                for (int j = 1; j < _depth; j++)
                 {
                    ArrayList ids1 = new ArrayList();
                    ids1.AddRange(idList);
                    item = new ItemWithArrayList(ids1, item);
                }
                objectContainer.Store(item);
            }
            objectContainer.Commit();
            StopTimer("Store " + TotalObjects() + " objects");
        }
```

```
InsertPerformanceBenchmark.cs: SimplestItem
public class SimplestItem
     {
        public int _id;
        public SimplestItem _child;

        public SimplestItem()
         {
        }

        public SimplestItem(int id, SimplestItem child)
         {
            _id = id;
            _child = child;
        }
    }
```

```
InsertPerformanceBenchmark.cs: ItemWithArray
public class ItemWithArray
     {
        public int[] _id;
        public ItemWithArray _child;

        public ItemWithArray()
         {
        }

        public ItemWithArray(int[] id, ItemWithArray child)
         {
            _id = id;
            _child = child;
        }
    }
```

```
InsertPerformanceBenchmark.cs: ItemWithArrayList
public class ItemWithArrayList
     {
        public ArrayList _ids;
        public ItemWithArrayList _child;
        public ItemWithArrayList()
         {
        }

        public ItemWithArrayList(ArrayList ids, ItemWithArrayList child)
         {
            _ids = ids;
            _child = child;
        }
    }
```

```
InsertPerformanceBenchmark.cs: ItemWithStringBuilder
public class ItemWithStringBuilder
     {
        public StringBuilder _name;
        public ItemWithStringBuilder _child;

        public ItemWithStringBuilder()
         {
        }

        public ItemWithStringBuilder(StringBuilder name, ItemWithStringBuilder child)
         {
            _name = name;
            _child = child;
        }
    }
```

The following results were achieved for the testing configuration:

.NET:

Storing 10000 objects with 3 levels of embedded objects:

- primitive object with int field

Store 30000 objects: 1123ms

- object with String field

Store 30000 objects: 1356ms

- object with StringBuilder field

Store 30000 objects: 4982ms

- object with int array field

Store 30000 objects: 1810ms

- object with ArrayList field

Store 30000 objects: 3479ms

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.cs: RunIndexTest
private void RunIndexTest()
```

```
         {
            Init();
            System.Console.WriteLine("Storing " + _count + " objects with "
+ _depth + " levels of embedded objects:");

            Clean();
            Configure();
            System.Console.WriteLine(" - no index");
            Open();
            Store();
            Close();

            ConfigureIndex();
            System.Console.WriteLine(" - index on String field");
            Open();
            Store();
            Close();
        }
```

```
InsertPerformanceBenchmark.cs: Configure
private void Configure()
         {
            IConfiguration config = Db4oFactory.Configure();
            config.LockDatabaseFile(false);
            config.WeakReferences(false);
            config.Io(new MemoryIoAdapter());
            config.FlushFileBuffers(false);
        }
```

```
InsertPerformanceBenchmark.cs: ConfigureIndex
private void ConfigureIndex()
         {
            IConfiguration config = Db4oFactory.Configure();
            config.LockDatabaseFile(false);
            config.WeakReferences(false);
            config.Io(new MemoryIoAdapter());
            config.FlushFileBuffers(false);
            config.ObjectClass(typeof(Item)).ObjectField("_name").Indexed(true);
        }
```

```
InsertPerformanceBenchmark.cs: Init
private void Init()
         {
            _count = 10000;
            _depth = 3;
            _isClientServer = false;
        }
```

```
InsertPerformanceBenchmark.cs: Store
private void Store()
         {
            StartTimer();
            for (int i = 0; i < _count; i++)
             {
                Item item = new Item("load", null);
                for (int j = 1; j < _depth; j++)
                 {
                    item = new Item("load", item);
                }
                objectContainer.Store(item);
            }
            objectContainer.Commit();
            StopTimer("Store " + TotalObjects() + " objects");
        }
```

The following results were achieved for the testing configuration:

.NET:

Storing 10000 objects with 3 levels of embedded objects:

- no index

Store 30000 objects: 1235ms

- index on String field

Store 30000 objects: 1748ms

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.cs: RunInheritanceTest
private void RunInheritanceTest()
         {
            Configure();
            Init();
            Clean();
            System.Console.WriteLine("Storing " + _count + " objects of depth " + _depth);
            Open();
            Store();
            Close();

            Clean();
            System.Console.WriteLine("Storing " + _count + " inherited objects of depth " + _depth);
            Open();
            StoreInherited();
            Close();
        }
```

```
InsertPerformanceBenchmark.cs: Configure
private void Configure()
         {
            IConfiguration config = Db4oFactory.Configure();
            config.LockDatabaseFile(false);
            config.WeakReferences(false);
            config.Io(new MemoryIoAdapter());
            config.FlushFileBuffers(false);
        }
```

```
InsertPerformanceBenchmark.cs: Init
private void Init()
         {
            _count = 10000;
            _depth = 3;
            _isClientServer = false;
        }
```

```
InsertPerformanceBenchmark.cs: Store
private void Store()
         {
            StartTimer();
            for (int i = 0; i < _count; i++)
             {
                Item item = new Item("load", null);
                for (int j = 1; j < _depth; j++)
                 {
                    item = new Item("load", item);
                }
                objectContainer.Store(item);
            }
            objectContainer.Commit();
            StopTimer("Store " + TotalObjects() + " objects");
        }
```

```
InsertPerformanceBenchmark.cs: StoreInherited
private void StoreInherited()
         {
            StartTimer();
            for (int i = 0; i < _count; i++)
             {
                ItemDerived item = new ItemDerived("load", null);
                for (int j = 1; j < _depth; j++)
                 {
                    item = new ItemDerived("load", item);
                }
                objectContainer.Store(item);
            }
            objectContainer.Commit();
            StopTimer("Store " + TotalObjects() + " objects");
        }
```

```
InsertPerformanceBenchmark.cs: ItemDerived
public class ItemDerived : Item
     {

        public ItemDerived(String name, ItemDerived child)
            : base(name, child)
         {

        }
    }
```

The following results were achieved for the testing configuration:

.NET:

Storing 10000 objects of depth 3

Store 30000 objects: 1237ms

Storing 10000 inherited objects of depth 3

Store 30000 objects: 1699ms

Configuration Options

Configuration options can also affect the insert performance. Some of them we've already came across in the previous topics:

MemoryStorage- improves the insert performance, by replacing disk access with memory access.

lockDatabaseFile- reduces the resources consumption by removing database lock thread. Should only be used for JVM versions < 1.4

weakReferences - switching weak references off during insert operation releases extra resources and removed the cleanup thread.

NonFlushingStorage- 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.

Delete Performance

Delete operation only consists of marking an object as deleted in the database file and usually is very fast. For the clarity in this case we do not include time necessary to locate the object in the database. The considerations for the best delete performance would be the same as the considerations to make db4o database generally faster and would include:

- fast storage location (hard drive or RAM)
- enough operational memory

Let's consider some of the application setups and their influence on delete performance. In order to distinguish delete time from query time, all the deletions will be done by object id.

More Reading:

- Commit Frequency
- Complexity Of Objects
- Storage Characteristics
- Client-Server
- Cascade On Delete

Commit Frequency

Commit time has a major influence on all db4o operations. Commit is generally a slow operation as it requires physical disk access. The following example shows how the frequency of commits affects the delete performance:

The following initial setup is used:

Delete procedures are as follows:

Item class used in this example has depth = 3:

The results on the test computer are:

Delete test with different commit frequency

Test delete all:

Store 30000 objects: 2432ms

Deleted all objects: 3731ms

Test delete all with commit after each 1000 objects:

Store 30000 objects: 2103ms

Deleted all objects: 8615ms

You can see that the time required for deletion increased with the amount of commits. To improve the performance commit frequency should be kept relatively low, in the same time it must ensure the integrity of the logical transactions.

Complexity Of Objects

Unlike other database operations deletion is mostly unaffected by the complexity of objects. The delete operation basically marks an object as deleted and is the same for any type of object. The main time required is to locate the object record in the database. This is demonstrated by the following example:

The following classes are used for the test:

The deletion procedure deletes several items from the whole amount of objects in the database:

Approximate results on the test computer are presented below:

Delete test with different objects

Deleting 1 of 10000 objects with 3 levels of embedded objects:

- *primitive object with int field*
  Store 30000 objects: 1720ms
  Deleted 10 objects in: 0 ms.
- *object with String field*
  Store 30000 objects: 1690ms
  Deleted 10 objects in: 1 ms.
- *object with StringBuffer field*
  Store 30000 objects: 4424ms
  Deleted 10 objects in: 1 ms.
- *object with int array field*
  Store 30000 objects: 2071ms
  Deleted 10 objects in: 0 ms.
- *object with ArrayList field*
  Store 30000 objects: 4264ms
  Deleted 10 objects in: 1 ms.

You can see that the complexity and structure of an object play little or no role in the performance.

Storage Characteristics

As any other db4o operation delete is dependent on the performance of a storage location. This can be easily compared with a database stored on a hard drive and in RAM:

Delete test: RAM disk

Store 90000 objects: 5973ms

Deleting 1 object of depth 3 on a RAM drive:

Deleted all objects: 5249ms

Delete test: hard drive

Store 90000 objects: 5043ms

Deleting 1 object of depth 3 on a hard drive:

Deleted all objects: 7475ms

The general rule is: the faster the drive is the better is the performance.

Client-Server

The comparison of delete performance in a local or networked database shows an obvious result: local mode is faster as it does not include network communication delays:

The approximate results on a test computer:

Delete test: Client/Server environment

Store 30000 objects: 1710ms

Delete 10 of 10000 objects [depth 3] locally:

Deleted 10 objects in: 1 ms.

Store 30000 objects: 2721ms

Delete 10 of 10000 objects [depth 3] remotely:

Deleted 10 objects in: 31 ms.

Cascade On Delete

If you are using cascadeOnDelete option and explicit deletion of one object causes cascaded delete on several others, you may expect slower execution:

Note that in the first part of the test each object deletion only includes a single object. Therefore, when we delete a 1000 objects the amount of the objects in the database is 1000 less. In the second part each object deletion will trigger deletion of all field objects (cascadeOnDelete option), therefore the amount of objects deleted from the database can be anywhere between 1000 (only lowest level objects) and 3000(all top level objects with their field objects).

The results from the test machine:

Delete test with objects of different depth

Deleting 1000 of 10000 objects with 1 levels of embedded objects:

Store 10000 objects: 528ms

Amount of objects left: 10000

Deleted 1000 objects in: 197 ms.

Amount of objects left: 9000

Deleting 1000 of 10000 objects with 3 levels of embedded objects:

Store 30000 objects: 1693ms

Amount of objects left: 39000

Deleted 1000 objects in: 340 ms.

Amount of objects left: 36978

Update Performance

Update performance is influenced by the similar factors as Insert Performance. The main factors include: configuration, disk access times, complexity of objects.

The following chapters provide some simple tests showing the influence of the above-mentioned factors. The test results are provided for Toshiba Sattelite Pro A120 notebook with 1,5Gb RAM 120GB ATA drive running on Vista and may be different on a different environment.

The following Item class is used in most of the tests:

```
UpdatePerformanceBenchmark.cs: Item
private void Update(Object item)
         {
            objectContainer.Store(item);
        }
        // end Update


        private void RunDifferentObjectsTest()
         {
```

```
System.Console.WriteLine("Update test with different objects");
int objectsToUpdate = 90;
int updated = objectsToUpdate;

InitDifferentObjectsTest();

Clean();
System.Console.WriteLine(" - primitive object with int field");
Open(Configure());
StoreSimplest();

IObjectSet result = objectContainer.QueryByExample(null);
StartTimer();
for (int i = 0; i < objectsToUpdate; i++)
 {
    if (result.HasNext())
     {
        SimplestItem item = (SimplestItem)result.Next();
        item._id = 1;
        Update(item);
    }
    else
     {
        updated = i;
        break;
    }
}
StopTimer("Updated " + updated + " items");
Close();

Clean();
Open(Configure());
System.Console.WriteLine(" - object with string field");
Store();
updated = objectsToUpdate;
result = objectContainer.QueryByExample(null);
StartTimer();
for (int i = 0; i < objectsToUpdate; i++)
 {
    if (result.HasNext())
     {
        Item item = (Item)result.Next();
        item._name = "Updated";
        Update(item);
    }
    else
     {
        updated = i;
        break;
    }
}
StopTimer("Updated " + updated + " items");
Close();

Clean();
Open(Configure());
System.Console.WriteLine(" - object with StringBuilder field");
StoreWithStringBuilder();

updated = objectsToUpdate;
```

```
            result = objectContainer.QueryByExample(null);
            StartTimer();
            for (int i = 0; i < objectsToUpdate; i++)
             {
                if (result.HasNext())
                 {
                    ItemWithStringBuilder item = (ItemWithStringBuilder)result.Next();
                    item._name = new StringBuilder("Updated");
                    Update(item);
                }
                else
                 {
                    updated = i;
                    break;
                }
            }
            StopTimer("Updated " + updated + " items");
            Close();

            Clean();
            Open(Configure());
            System.Console.WriteLine(" - object with int array field");
            StoreWithArray();
            updated = objectsToUpdate;
            result = objectContainer.QueryByExample(null);
            StartTimer();
            for (int i = 0; i < objectsToUpdate; i++)
             {
                if (result.HasNext())
                 {
                    ItemWithArray item = (ItemWithArray)result.Next();
                    item._id = new int[]  { 1, 2, 3 };
                    Update(item);
                }
                else
                 {
                    updated = i;
                    break;
                }
            }
            StopTimer("Updated " + updated + " items");
            Close();

            Clean();
            Open(Configure());
            System.Console.WriteLine(" - object with ArrayList field");
            StoreWithArrayList();
            updated = objectsToUpdate;
            result = objectContainer.QueryByExample(null);
            StartTimer();
            for (int i = 0; i < objectsToUpdate; i++)
             {
                if (result.HasNext())
                 {
                    ItemWithArrayList item = (ItemWithArrayList)result.Next();
                    item._ids = new ArrayList();
                    Update(item);
                }
                else
                 {
                    updated = i;
                    break;
                }
            }
            StopTimer("Updated " + updated + " items");
            Close();
        }
        // end RunDifferentObjectsTest


        private void RunIndexTest()
         {
            System.Console.WriteLine("Update test for objects with and without indexed fields");

            int objectsToUpdate = 100;
            Init();
            System.Console.WriteLine("Updating " + objectsToUpdate + " of " + _count + " objects");
            Clean();
            Open(Configure());
            Store();
            UpdateItems(objectsToUpdate);
            Close();

            Clean();
            Init();
            System.Console.WriteLine("Updating " + objectsToUpdate + " of " + _count + " objects with indexed fie
            Open(ConfigureIndexTest());
            Store();
            UpdateItems(objectsToUpdate);
            Close();
        }
        // end RunIndexTest


        private void Init()
         {
            _count = 1000;
            _depth = 90;
            _isClientServer = false;

        }
        // end Init

        private void InitDifferentObjectsTest()
         {
            _count = 1000;
            _depth = 1;
            _isClientServer = false;

        }
        // end InitDifferentObjectsTest


        private void InitForClientServer()
         {
            _count = 1000;
            _depth = 90;
            _isClientServer = true;
            _host = "localhost";
        }
        // end InitForClientServer

        private void InitForRamDriveTest()
         {
            _count = 30000;
            _depth = 1;
            _filePath = "r:\\performance.db4o";
            _isClientServer = false;

        }
        // end InitForRamDriveTest

        private void InitForHardDriveTest()
         {
            _count = 10000;
            _depth = 3;
            _filePath = "performance.db4o";
            _isClientServer = false;
        }
        // end InitForHardDriveTest

        private void InitForCommitTest()
         {
            _count = 10000;
            _commitInterval = 1000;
            _depth = 3;
            _isClientServer = false;
        }
        // end InitForCommitTest

        private void Clean()
         {
            File.Delete(_filePath);
        }
        // end Clean

        private IConfiguration Configure()
         {
            IConfiguration config = Db4oFactory.NewConfiguration();
            // using MemoryIoAdapter improves the performance
            // by replacing the costly disk IO operations with
            // memory access
            config.Io(new MemoryIoAdapter());
            return config;
        }
        // end Configure

        private IConfiguration ConfigureTP()
         {
            IConfiguration config = Db4oFactory.NewConfiguration();
            // With Transparent Persistence enabled only modified
            // objects are written to disk. This allows to achieve
            // better performance
            config.ObjectClass(typeof(Item)).CascadeOnUpdate(true);
            return config;
        }
        // end ConfigureTP

        private IConfiguration ConfigureCascade()
         {
```

```
        IConfiguration config = Db4oFactory.NewConfiguration();
        // CascadeOnUpdate can be a performance-killer for
        // deep object hierarchies
        config.ObjectClass(typeof(Item)).CascadeOnUpdate(true);
        return config;
    }
    // end ConfigureCascade

    private IConfiguration ConfigureIndexTest()
     {
        IConfiguration config = Db4oFactory.NewConfiguration();
        config.Io(new MemoryIoAdapter());
        config.ObjectClass(typeof(Item)).ObjectField("_name").Indexed(true);
        return config;
    }
    // end ConfigureIndexTest

    private IConfiguration ConfigureForCommitTest()
     {
        IConfiguration config = Db4oFactory.NewConfiguration();
        // the Commit information is physically written
        // and in the correct order
        config.FlushFileBuffers(true);
        return config;
    }
    // end ConfigureForCommitTest

    private IConfiguration ConfigureClientServer()
     {
        IConfiguration config = Db4oFactory.NewConfiguration();
        config.ClientServer().SingleThreadedClient(true);
        return config;
    }
    // end ConfigureClientServer

    private IConfiguration ConfigureDriveTest()
     {
        IConfiguration config = Db4oFactory.NewConfiguration();
        config.FlushFileBuffers(true);
        return config;
    }
    // end ConfigureDriveTest

    private void Store()
     {
        StartTimer();
        for (int i = 0; i < _count; i++)
         {
            Item item = new Item("level" + i, null);
            for (int j = 1; j < _depth; j++)
             {
                item = new Item("level" + i + "/" + j, item);
            }
            objectContainer.Store(item);
        }
        objectContainer.Commit();
        StopTimer("Store " + TotalObjects() + " objects");
    }
    // end Store
```

```
        private void StoreActivatableItems()
         {
            StartTimer();
            for (int i = 0; i < _count; i++)
             {
                ActivatableItem item = new ActivatableItem("level" + i, null);
                for (int j = 1; j < _depth; j++)
                 {
                    item = new ActivatableItem("level" + i + "/" + j, item);
                }
                objectContainer.Store(item);
            }
            objectContainer.Commit();
            StopTimer("Store " + TotalObjects() + " objects");
        }
        // end StoreActivatableItems

        private void StoreInherited()
         {
            StartTimer();
            for (int i = 0; i < _count; i++)
             {
                ItemDerived item = new ItemDerived("level" + i, null);
                for (int j = 1; j < _depth; j++)
                 {
                    item = new ItemDerived("level" + i + "/" + j, item);
                }
                objectContainer.Store(item);
            }
            objectContainer.Commit();
            StopTimer("Store " + TotalObjects() + " objects");
        }
        // end StoreInherited


        private void StoreWithStringBuilder()
         {
            StartTimer();
            for (int i = 0; i < _count; i++)
             {
                ItemWithStringBuilder item = new ItemWithStringBuilder(new StringBuilder("level" + i), null);
                for (int j = 1; j < _depth; j++)
                 {
                    item = new ItemWithStringBuilder(new StringBuilder("level" + i + "/" + j), item);
                }
                objectContainer.Store(item);
            }
            objectContainer.Commit();
            StopTimer("Store " + TotalObjects() + " objects");
        }
        // end StoreWithStringBuilder

        private void StoreSimplest()
         {
            StartTimer();
            for (int i = 0; i < _count; i++)
             {
                SimplestItem item = new SimplestItem(i, null);
                for (int j = 1; j < _depth; j++)
                 {
                    item = new SimplestItem(i, item);
                }
                objectContainer.Store(item);
            }
            objectContainer.Commit();
            StopTimer("Store " + TotalObjects() + " objects");
        }
        // end StoreSimplest

        private void StoreWithArray()
         {
            StartTimer();
            int[] array = new int[]  { 1, 2, 3, 4 };
            for (int i = 0; i < _count; i++)
             {
                int[] id = new int[]  { 1, 2, 3, 4 };
                ItemWithArray item = new ItemWithArray(id, null);
                for (int j = 1; j < _depth; j++)
                 {
                    int[] id1 = new int[]  { 1, 2, 3, 4 };
                    item = new ItemWithArray(id1, item);
                }
                objectContainer.Store(item);
            }
            objectContainer.Commit();
            StopTimer("Store " + TotalObjects() + " objects");
        }
        // end StoreWithArray

        private void StoreWithArrayList()
         {
            StartTimer();
            ArrayList idList = new ArrayList();
            idList.Add(1);
            idList.Add(2);
            idList.Add(3);
            idList.Add(4);
            for (int i = 0; i < _count; i++)
             {
                ArrayList ids = new ArrayList();
                ids.AddRange(idList);
                ItemWithArrayList item = new ItemWithArrayList(ids, null);
                for (int j = 1; j < _depth; j++)
                 {
                    ArrayList ids1 = new ArrayList();
                    ids1.AddRange(idList);
                    item = new ItemWithArrayList(ids1, item);
                }
                objectContainer.Store(item);
            }
            objectContainer.Commit();
            StopTimer("Store " + TotalObjects() + " objects");
        }
        // end StoreWithArrayList

        private int TotalObjects()
         {
            return _count * _depth;
        }
        // end TotalObjects
```

```
private void Open(IConfiguration config)
 {
    if (_isClientServer)
     {
        int port = TCP ? PORT : 0;
        string user = "db4o";
        string password = user;
        objectServer = Db4oFactory.OpenServer(_filePath, port);
        objectServer.GrantAccess(user, password);
        objectContainer = TCP ? Db4oFactory.OpenClient(_host, port, user,
                password) : objectServer.OpenClient();
    }
    else
     {
        objectContainer = Db4oFactory.OpenFile(config, _filePath);
    }
}
// end Open

private void Close()
 {
    objectContainer.Close();
    if (_isClientServer)
     {
        objectServer.Close();
    }
}
//end Close

private void StartTimer()
 {
    _startTime = DateTime.Now.Ticks;
}
// end StartTimer

private void StopTimer(string message)
 {
    long stop = DateTime.Now.Ticks;
    long duration = stop - _startTime;
    System.Console.WriteLine(message + ": " + duration + " ticks");
}
// end StopTimer

public class Item
 {

    public string _name;
    public Item _child;

    public Item()
     {

    }

    public Item(string name, Item child)
     {
        _name = name;
        _child = child;
    }
```

```
        }
```

More Reading:

- Configuration
- Object Structure
- Commit Frequency
- Hard Drive Speed
- Client-Server
- Indexes
- Inheritance

Configuration

db4o provides a wide range of configuration options to help you meet your performance and reliability requirements. The following example shows how different configurations affect update performance:

```
UpdatePerformanceBenchmark.cs: RunConfigurationTest
private void RunConfigurationTest()
         {
            System.Console.WriteLine(
"Update test with different configurations");

            //
            Clean();
            Init();
            System.Console.WriteLine("Update test: default configurations");
            Open(Db4oFactory.NewConfiguration());
            Store();
            UpdateItems(90);
            Close();
            //

            Clean();
            System.Console.WriteLine("Update test: memory IO adapter");
            Open(Configure());
            Store();
            UpdateItems(90);
            Close();
            //
            Clean();
            System.Console.WriteLine("Update test: cascade on Update");
            Open(ConfigureCascade());
            Store();
            UpdateTopLevelItems(90);
            Close();

            //
            Clean();
            System.Console.WriteLine("Update test: Transparent Persistence");
            Open(ConfigureTP());
            StoreActivatableItems();
            UpdateActivatableItems(90);
            Close();

        }
```

```
UpdatePerformanceBenchmark.cs: UpdateItems
private void UpdateItems(int count)
         {
            StartTimer();
            IObjectSet result = objectContainer.QueryByExample(null);

            for (int i = 0; i < count; i++)
             {
                if (result.HasNext())
                 {
                    Item item = (Item)result.Next();
                    item._name = "Updated";
                    Update(item);
                }
                else
                 {
                    count = i;
                    break;
                }
            }
            StopTimer("Updated " + count + " items");
        }
```

```
UpdatePerformanceBenchmark.cs: UpdateTopLevelItems
private void UpdateTopLevelItems(int count)
         {
            StartTimer();
            IQuery query = objectContainer.Query();
            query.Constrain(typeof(Item));
            query.Descend("_name").Constrain("level0").StartsWith(true);
            IObjectSet result = query.Execute();

            for (int i = 0; i < count; i++)
             {
                if (result.HasNext())
                 {
                    Item item = (Item)result.Next();
                    item._name = "Updated";
                    Update(item);
                }
                else
                 {
                    count = i;
                    break;
                }
            }
            StopTimer("Updated " + count + " items");
        }
```

```
UpdatePerformanceBenchmark.cs: UpdateActivatableItems
private void UpdateActivatableItems(int count)
         {
            StartTimer();
            IQuery Query = objectContainer.Query();
            Query.Constrain(typeof(ActivatableItem));
            Query.Descend("_name").Constrain("level0").StartsWith(true);
            IObjectSet result = Query.Execute();

            for (int i = 0; i < count; i++)
             {
                if (result.HasNext())
```

```
                 {
                    ActivatableItem item = (ActivatableItem)result.Next();
                    item.Name = "Updated";
                    Update(item);
                }
                else
                 {
                    count = i;
                    break;
                }
            }
            StopTimer("Updated " + count + " items");
        }
```

```
UpdatePerformanceBenchmark.cs: Configure
private IConfiguration Configure()
         {
            IConfiguration config = Db4oFactory.NewConfiguration();
            // using MemoryIoAdapter improves the performance
            // by replacing the costly disk IO operations with
            // memory access
            config.Io(new MemoryIoAdapter());
            return config;
        }
```

```
UpdatePerformanceBenchmark.cs: ConfigureCascade
private IConfiguration ConfigureCascade()
         {
            IConfiguration config = Db4oFactory.NewConfiguration();
            // CascadeOnUpdate can be a performance-killer for
            // deep object hierarchies
            config.ObjectClass(typeof(Item)).CascadeOnUpdate(true);
            return config;
        }
```

```
UpdatePerformanceBenchmark.cs: ConfigureTP
private IConfiguration ConfigureTP()
         {
            IConfiguration config = Db4oFactory.NewConfiguration();
            // With Transparent Persistence enabled only modified
            // objects are written to disk. This allows to achieve
            // better performance
            config.ObjectClass(typeof(Item)).CascadeOnUpdate(true);
            return config;
        }
```

```
UpdatePerformanceBenchmark.cs: ActivatableItem
public class ActivatableItem : IActivatable
         {

            private string _name;
            public ActivatableItem _child;

            [System.NonSerialized]
            IActivator _activator;

            public void Bind(IActivator activator)
             {
                if (_activator == activator)
                 {
                    return;
                }
                if (activator != null && _activator != null)
                 {
                    throw new System.InvalidOperationException();
                }
                _activator = activator;
            }

            public void Activate(ActivationPurpose purpose)
             {
                if (_activator == null) return;
                _activator.Activate(purpose);
            }


            public ActivatableItem()
             {

            }

            public ActivatableItem(string name, ActivatableItem child)
             {
                Name = name;
                _child = child;
            }

            public string Name
             {
                get
                 {
                    return _name;
                }
                set
                 {
                    _name = value;
                }
            }


        }
```

The results:

Update test with different configurations

Update test: default configurations

Store 90000 objects: 7869ms

Updated 90 items: 471ms

Update test: memory IO adapter

Store 90000 objects: 6622ms

Updated 90 items: 289ms

Update test: cascade on update

Store 90000 objects: 6848ms

Updated 90 items: 1531ms

Update test: Transparent Persistence

Store 90000 objects: 6604ms

Updated 90 items: 1297ms

From the results you can see that MemoryIoAdapter allows to improve performance, CascadeOnUpdate option results in a considerable drop of performance, and Transparent Persistence makes it better again.

Object Structure

Update performance is dependent upon the structure and complexity of objects. This is demonstrated in the following test:

```
UpdatePerformanceBenchmark.cs: RunDifferentObjectsTest
private void RunDifferentObjectsTest()
         {
            System.Console.WriteLine("Update test with different objects");
            int objectsToUpdate = 90;
            int updated = objectsToUpdate;

            InitDifferentObjectsTest();

            Clean();
            System.Console.WriteLine(" - primitive object with int field");
            Open(Configure());
            StoreSimplest();

            IObjectSet result = objectContainer.QueryByExample(null);
            StartTimer();
            for (int i = 0; i < objectsToUpdate; i++)
             {
                if (result.HasNext())
                 {
                    SimplestItem item = (SimplestItem)result.Next();
                    item._id = 1;
                    Update(item);
                }
                else
                 {
                    updated = i;
                    break;
                }
            }
            StopTimer("Updated " + updated + " items");
            Close();

            Clean();
            Open(Configure());
            System.Console.WriteLine(" - object with string field");
            Store();
            updated = objectsToUpdate;
            result = objectContainer.QueryByExample(null);
            StartTimer();
            for (int i = 0; i < objectsToUpdate; i++)
             {
                if (result.HasNext())
                 {
                    Item item = (Item)result.Next();
                    item._name = "Updated";
                    Update(item);
                }
                else
                 {
                    updated = i;
                    break;
                }
            }
            StopTimer("Updated " + updated + " items");
            Close();

            Clean();
            Open(Configure());
            System.Console.WriteLine(" - object with StringBuilder field");
            StoreWithStringBuilder();

            updated = objectsToUpdate;
            result = objectContainer.QueryByExample(null);
            StartTimer();
            for (int i = 0; i < objectsToUpdate; i++)
             {
                if (result.HasNext())
                 {
                    ItemWithStringBuilder item =
(ItemWithStringBuilder)result.Next();
                    item._name = new StringBuilder("Updated");
                    Update(item);
                }
                else
                 {
                    updated = i;
                    break;
                }
            }
            StopTimer("Updated " + updated + " items");
            Close();

            Clean();
            Open(Configure());
            System.Console.WriteLine(" - object with int array field");
            StoreWithArray();
            updated = objectsToUpdate;
            result = objectContainer.QueryByExample(null);
            StartTimer();
            for (int i = 0; i < objectsToUpdate; i++)
             {
                if (result.HasNext())
                 {
                    ItemWithArray item = (ItemWithArray)result.Next();
                    item._id = new int[]  { 1, 2, 3 };
                    Update(item);
                }
                else
                 {
                    updated = i;
                    break;
                }
            }
            StopTimer("Updated " + updated + " items");
            Close();

            Clean();
            Open(Configure());
            System.Console.WriteLine(" - object with ArrayList field");
            StoreWithArrayList();
            updated = objectsToUpdate;
            result = objectContainer.QueryByExample(null);
            StartTimer();
            for (int i = 0; i < objectsToUpdate; i++)
             {
                if (result.HasNext())
                 {
                    ItemWithArrayList item = (ItemWithArrayList)result.Next();
                    item._ids = new ArrayList();
                    Update(item);
                }
                else
                 {
                    updated = i;
                    break;
                }
            }
            StopTimer("Updated " + updated + " items");
            Close();
        }
```

```
UpdatePerformanceBenchmark.cs: SimplestItem
public class SimplestItem
         {

            public int _id;
            public SimplestItem _child;

            public SimplestItem()
             {
            }

            public SimplestItem(int id, SimplestItem child)
             {
                _id = id;
                _child = child;
            }
        }
```

```
UpdatePerformanceBenchmark.cs: ItemWithStringBuilder
public class ItemWithStringBuilder
         {

            public StringBuilder _name;
            public ItemWithStringBuilder _child;

            public ItemWithStringBuilder()
             {
            }

            public ItemWithStringBuilder(StringBuilder name,
ItemWithStringBuilder child)
             {
                _name = name;
                _child = child;
            }
        }
```

```
UpdatePerformanceBenchmark.cs: ItemWithArray
public class ItemWithArray
         {
            public int[] _id;
            public ItemWithArray _child;

            public ItemWithArray()
             {
            }

            public ItemWithArray(int[] id, ItemWithArray child)
             {
                _id = id;
                _child = child;
            }
        }
```

```
UpdatePerformanceBenchmark.cs: ItemWithArrayList
public class ItemWithArrayList
         {

            public ArrayList _ids;
            public ItemWithArrayList _child;

            public ItemWithArrayList()
             {
            }

            public ItemWithArrayList(ArrayList ids, ItemWithArrayList child)
             {
                _ids = ids;
                _child = child;
            }
        }
```

The results:

Update test with different objects

- primitive object with int field

Store 1000 objects: 273ms

Updated 90 items: 185ms

- object with String field

Store 1000 objects: 166ms

Updated 90 items: 158ms

- object with StringBuffer field

Store 1000 objects: 199ms

Updated 90 items: 488ms

- object with int array field

Store 1000 objects: 78ms

Updated 90 items: 134ms

- object with ArrayList field

Store 1000 objects: 191ms

Updated 90 items: 647ms

In general update of a more complex object takes more time, however the exact result depends on the TypeHandler implementation.

Commit Frequency

Commit frequency has a direct impact on db4o performance. Commit is an expensive operation due to physical disk access. However, commit is also the only way to ensure that the whole transaction is stored safely on the disk and no data loss will occur in case of unexpected system failure.

The following test shows how commit frequency influences the performance on update:

```
UpdatePerformanceBenchmark.cs: RunCommitTest
private void RunCommitTest()
         {
            System.Console.WriteLine(
"Update test with different Commit frequency");

            InitForCommitTest();

            Clean();
            System.Console.WriteLine("Test Update all:");
            Open(ConfigureForCommitTest());
            Store();
            UpdateItems(_count);
            Close();



            Clean();
            System.Console.WriteLine(
"Test Update all with Commit after each " + _commitInterval + " objects:");
            Open(ConfigureForCommitTest());
            Store();
            UpdateWithCommit(_count);
            Close();


        }
```

```
UpdatePerformanceBenchmark.cs: ConfigureForCommitTest
private IConfiguration ConfigureForCommitTest()
         {
            IConfiguration config = Db4oFactory.NewConfiguration();
            // the Commit information is physically written
            // and in the correct order
            config.FlushFileBuffers(true);
            return config;
        }
```

```
UpdatePerformanceBenchmark.cs: UpdateItems
private void UpdateItems(int count)
         {
            StartTimer();
            IObjectSet result = objectContainer.QueryByExample(null);

            for (int i = 0; i < count; i++)
             {
                if (result.HasNext())
                 {
                    Item item = (Item)result.Next();
                    item._name = "Updated";
                    Update(item);
                }
                else
                {
                    count = i;
                    break;
                }
            }
            StopTimer("Updated " + count + " items");
        }
```

```
UpdatePerformanceBenchmark.cs: UpdateWithCommit
private void UpdateWithCommit(int count)
         {
            StartTimer();
            IObjectSet result = objectContainer.QueryByExample(null);
            int j = 0;
            for (int i = 0; i < count; i++)
             {
                if (result.HasNext())
                 {
                    Item item = (Item)result.Next();
                    item._name = "Updated";
                    Update(item);
                    if (j >= _commitInterval)
                     {
                        j = 0;
                        objectContainer.Commit();
                    }
                    else
                     {
                        j++;
                    }
                }
                else
                 {
                    count = i;
                    break;
                }
            }
            StopTimer("Updated " + count + " items ");
        }
```

The results:

Update test with different commit frequency

Test update all:

Store 30000 objects: 2661ms

Updated 10000 items: 1402ms

Test update all with commit after each 1000 objects:

Store 30000 objects: 2250ms

Updated 10000 items : 2812ms

Hard Drive Speed

Update db4o operation requires disk access and therefore is very dependent on the disk speed. To emulate a different drive speed we will use a RAMDISK utility, which creates an alternative storage media in the memory.

```
UpdatePerformanceBenchmark.cs: RunHardDriveTest
private void RunHardDriveTest()
```

```
         {
            System.Console.WriteLine("Update test: hard drive");
            int objectsToUpdate = 90;

            InitForHardDriveTest();
            Clean();
            Open(ConfigureDriveTest());
            Store();
            System.Console.WriteLine("Updating  " + objectsToUpdate +
" objects on a hard drive:");
            UpdateItems(objectsToUpdate);
            Close();
        }
```

```
UpdatePerformanceBenchmark.cs: InitForHardDriveTest
private void InitForHardDriveTest()
         {
            _count = 10000;
            _depth = 3;
            _filePath = "performance.db4o";
            _isClientServer = false;
        }
```

```
UpdatePerformanceBenchmark.cs: RunRamDiskTest
private void RunRamDiskTest()
         {
            System.Console.WriteLine("Update test: RAM disk");
            int objectsToUpdate = 90;
            InitForRamDriveTest();
            Clean();
            Open(ConfigureDriveTest());
            Store();
            System.Console.WriteLine("Updating  " + objectsToUpdate
+ " objects on a RAM drive:");
            UpdateItems(objectsToUpdate);
            Close();
        }
```

```
UpdatePerformanceBenchmark.cs: InitForRamDriveTest
private void InitForRamDriveTest()
         {
            _count = 30000;
            _depth = 1;
            _filePath = "r:\\performance.db4o";
            _isClientServer = false;

        }
```

```
UpdatePerformanceBenchmark.cs: ConfigureDriveTest
private IConfiguration ConfigureDriveTest()
         {
            IConfiguration config = Db4oFactory.NewConfiguration();
            config.FlushFileBuffers(true);
            return config;
        }
```

The results:

Update test: hard drive

Store 30000 objects: 2884ms

Updating 90 objects on a hard drive:

Updated 90 items: 250ms

Update test: RAM disk

Store 30000 objects: 1910ms

Updating 90 objects on a RAM drive:

Updated 90 items: 105ms

The test shows that the faster media (RAMDISK) shows better performance on update.

Client-Server

Client/server performance is a bit slower than local performance. This is illustrated with the following test:

```
UpdatePerformanceBenchmark.cs: RunClientServerTest
private void RunClientServerTest()
         {
            System.Console.WriteLine("Update test: Client/Server environment");
            int objectsToUpdate = 30;


            Init();
            Clean();
            Open(ConfigureClientServer());
            Store();
            System.Console.WriteLine("Update " + objectsToUpdate + " of " +
_count + " objects locally:");
            UpdateItems(objectsToUpdate);
            Close();

            InitForClientServer();
            Clean();
            Open(ConfigureClientServer());
            Store();
            System.Console.WriteLine("Update " + objectsToUpdate + " of " +
_count + " objects remotely:");
            UpdateItems(objectsToUpdate);
            Close();
        }
```

```
UpdatePerformanceBenchmark.cs: InitForClientServer
private void InitForClientServer()
         {
            _count = 1000;
            _depth = 90;
            _isClientServer = true;
            _host = "localhost";
        }
```

```
UpdatePerformanceBenchmark.cs: Init
private void Init()
         {
            _count = 1000;
            _depth = 90;
            _isClientServer = false;

        }
```

```
UpdatePerformanceBenchmark.cs: ConfigureClientServer
private IConfiguration ConfigureClientServer()
         {
            IConfiguration config = Db4oFactory.NewConfiguration();
            config.ClientServer().SingleThreadedClient(true);
            return config;
        }
```

The results:

Update test: Client/Server environment

Store 90000 objects: 7935ms

Update 30 of 1000 objects locally:

Updated 30 items: 404ms

Store 90000 objects: 11421ms

Update 30 of 1000 objects remotely:

Updated 30 items: 436ms

You can see that the performance drop is quite insignificant in this case, however it can be much worse on slow or unreliable networks.

Indexes

Updating indexed fields always takes longer as the index should be updated as well. This is shown in the following test:

```
UpdatePerformanceBenchmark.cs: RunIndexTest
private void RunIndexTest()
         {
            System.Console.WriteLine(
"Update test for objects with and without indexed fields");

            int objectsToUpdate = 100;
            Init();
            System.Console.WriteLine("Updating " + objectsToUpdate +
" of " + _count + " objects");
            Clean();
            Open(Configure());
            Store();
            UpdateItems(objectsToUpdate);
            Close();

            Clean();
            Init();
            System.Console.WriteLine("Updating " + objectsToUpdate +
" of " + _count + " objects with indexed field");
            Open(ConfigureIndexTest());
            Store();
            UpdateItems(objectsToUpdate);
            Close();
        }
```

```
UpdatePerformanceBenchmark.cs: Init
private void Init()
         {
            _count = 1000;
            _depth = 90;
            _isClientServer = false;

        }
```

```
UpdatePerformanceBenchmark.cs: ConfigureIndexTest
private IConfiguration ConfigureIndexTest()
         {
            IConfiguration config = Db4oFactory.NewConfiguration();
            config.Io(new MemoryIoAdapter());
            config.ObjectClass(typeof(Item)).
ObjectField("_name").Indexed(true);
            return config;
        }
```

The results:

Update test for objects with and without indexed fields

Updating 100 of 1000 objects

Store 90000 objects: 7466ms

Updated 100 items: 295ms

Updating 100 of 1000 objects with indexed field

Store 90000 objects: 6839ms

Updated 100 items: 441ms

Inheritance

Inherited objects take longer to store as their parent indexes need to be updated too.

```
UpdatePerformanceBenchmark.cs: RunInheritanceTest
private void RunInheritanceTest()
         {
            System.Console.WriteLine("Update test: objects with deep inheritance");

            int objectsToUpdate = 30;
            Init();
            Clean();
            Open(Configure());
            Store();
            System.Console.WriteLine("Updating " + objectsToUpdate + " objects");
            UpdateItems(objectsToUpdate);
            Close();

            Clean();
            Open(Configure());
            StoreInherited();
            System.Console.WriteLine("Updating " + objectsToUpdate +
" inherited objects");
            UpdateItems(objectsToUpdate);
            Close();

        }
```

```
UpdatePerformanceBenchmark.cs: Configure
private IConfiguration Configure()
         {
            IConfiguration config = Db4oFactory.NewConfiguration();
            // using MemoryIoAdapter improves the performance
            // by replacing the costly disk IO operations with
            // memory access
            config.Io(new MemoryIoAdapter());
            return config;
        }
```

```
UpdatePerformanceBenchmark.cs: Init
private void Init()
         {
            _count = 1000;
            _depth = 90;
            _isClientServer = false;

        }
```

```
UpdatePerformanceBenchmark.cs: ItemDerived
public class ItemDerived : Item
         {

            public ItemDerived(string name, ItemDerived child)
                : base(name, child)
             {

            }
        }
```

The results:

Update test: objects with deep inheritance

Store 90000 objects: 6312ms

Updating 30 objects

Updated 30 items: 272ms

Store 90000 objects: 5657ms

Updating 30 inherited objects

Updated 30 items: 436ms

Query Performance

Query Performance is one of the most important characteristics of a database system. In the same time it is probably the one that can vary the most. In general query performance can be dependent on the following list of factors and their combinations:

- hardware resources (free RAM, processor speed, disk access)
- database model & complexity of objects
- query structure
- query configuration
- amount of objects in the database
- connection speed
- indexes
- query optimization
- etc

The following set of tests shows some performance dependencies. A simple Item object with a string field is used in most of the tests:

```
QueryPerformanceBenchmark.cs: Item
public class Item
         {

            public string _name;
            public Item _child;

            public Item()
```

```
             {

            }

            public Item(string name, Item child)
             {
                _name = name;
                _child = child;
            }
        }
```

Different Query Types

db4o provides different query syntaxes: Query By Example, SODA, Native Queries and LINQ (for .NET 3.5 and higher). Under the hood all these syntaxes are converted to SODA. The conversion can be very straightforward (as in case of **QBE**[1]), or pretty sophisticated (some Native queries). The fact that conversion takes place and can be more or less complex affects the performance of queries expressed with different syntax. Another factor affecting the performance can be using unoptimized Native Queries: this can happen if the query is too complex to analyze or when optimization is disabled through configuration. Optimization is also applicable to LINQ queries, i.e. some of LINQ queries are currently too complex to analyze and optimize. In cases when optimization does not happen, the query is run against all instances of an object in the database, which is quite slow and consumes a lot of RAM.

```
QueryPerformanceBenchmark.cs: RunDifferentQueriesTest
private void RunDifferentQueriesTest()
         {
            Init();

            Clean();
            System.Console.WriteLine("Storing objects as a bulk:");
            Open(Configure());
            Store();
            Close();

            Open(Configure());
            //
            System.Console.WriteLine("Query by example:");
            StartTimer();
            Item item = (Item)objectContainer.QueryByExample(
                    new Item("level1/1", null)).Next();
            StopTimer("Select 1 object QBE: " + item._name);

            //
            System.Console.WriteLine("SODA:");
            StartTimer();
            IQuery query = objectContainer.Query();
            query.Constrain(typeof(Item));
            query.Descend("_name").Constrain("level1/1");
            item = (Item)query.Execute().Next();
            StopTimer("Select 1 object SODA: " + item._name);

            //
            System.Console.WriteLine("LINQ: ");
            StartTimer();
```

[1]Query By Example

```
            var resultLINQ = from Item it in objectContainer
                         where it._name.Equals("level1/1")
                         select it;

            IEnumerator<Item> i = resultLINQ.GetEnumerator();
            if (i.MoveNext())
             {
                item = i.Current;
                StopTimer("Select 1 object LINQ: " + item._name);
            }


            //
            System.Console.WriteLine("Native Query:");
            StartTimer();
            IList<Item> result = objectContainer.Query<Item>(delegate(Item it)
             {
                return it._name.Equals("level1/1");
            });
            item = result[0];
            StopTimer("Select 1 object NQ: " + item._name);
            Close();

            //
            Open(ConfigureUnoptimizedNQ());
            System.Console.WriteLine("Native Query Unoptimized:");
            StartTimer();
            result = objectContainer.Query<Item>(delegate(Item it)
             {
                return it._name.Equals("level1/1");
            });
            item = result[0];
            StopTimer("Select 1 object NQ: " + item._name);

            Close();
        }
```

```
QueryPerformanceBenchmark.cs: Init
private void Init()
         {
            _filePath = "performance.db4o";
            // amount of objects
            _count = 10000;
            // depth of objects
            _depth = 3;
            _isClientServer = false;

        }
```

```
QueryPerformanceBenchmark.cs: Configure
private IConfiguration Configure()
         {
            IConfiguration config = Db4oFactory.NewConfiguration();
            return config;
        }
```

The following results were obtained on a test machine:

Storing objects as a bulk:

Store 30000 objects: 5337 ms

Query by example:

Select 1 object QBE: level1/1: 1021 ms

SODA:

Select 1 object SODA: level1/1: 809 ms

LINQ:

Select 1 object LINQ: level1/1: 915 ms

Native Query:

Select 1 object ***NQ***[1]*: level1/1: 1604 ms*

Native Query Unoptimized:

Select 1 object NQ: level1/1: 5008 ms

You can see that SODA query shows the best performance. The other query types are less performant due to conversion, however they can be easier to support and refactor. The worst performance is shown in the case of unoptimized Native Query: in this case all the objects from the database were instantiated and tested against the constraint.

Query Structure

More complex queries take longer to execute, as they include more constrains and can impose some additional operations as sorting, aggregate, negation etc.

```
QueryPerformanceBenchmark.cs: RunQueryStructureTest
private void RunQueryStructureTest()
         {
            Init();

            Clean();
            System.Console.WriteLine("Storing objects as a bulk:");
            Open(Configure());
            Store();
            Close();

            //
            Open(Configure());
            System.Console.WriteLine("Simple SODA query:");
            StartTimer();
            IQuery query = objectContainer.Query();
            query.Constrain(typeof(Item));
            query.Descend("_name").Constrain("level1/1");
            Item item = (Item)query.Execute().Next();
            StopTimer("Select 1 object SODA: " + item._name);
            Close();

            //
            Open(Configure());
            System.Console.WriteLine("Sorted SODA query:");
            StartTimer();
            query = objectContainer.Query();
            query.Constrain(typeof(Item));
            query.Descend("_name").OrderDescending();
            item = (Item)query.Execute().Next();
            StopTimer("Select 1 object SODA: " + item._name);
            Close();
```

[1]Native Query

```
            //
            Open(Configure());
            System.Console.WriteLine("SODA query with joins:");
            StartTimer();
            query = objectContainer.Query();
            query.Constrain(typeof(Item));
            IConstraint con = query.Constrain("level2/1");
            query.Descend("_name").OrderDescending().Constrain("level1/1").Or(con);
            IList result = query.Execute();
            StopTimer("Selected " + result.Count + " objects SODA");
            Close();

        }
```

```
QueryPerformanceBenchmark.cs: Init
private void Init()
         {
            _filePath = "performance.db4o";
            // amount of objects
            _count = 10000;
            // depth of objects
            _depth = 3;
            _isClientServer = false;

        }
```

```
QueryPerformanceBenchmark.cs: Configure
private IConfiguration Configure()
         {
            IConfiguration config = Db4oFactory.NewConfiguration();
            return config;
        }
```

Results from the test machine:

Storing objects as a bulk:

Store 30000 objects: 3049ms

Simple SODA query:

Select 1 object SODA: level1/1: 440ms

Sorted SODA query:

Select 1 object SODA: level9999/2: 1509ms

SODA query with joins:

Selected 1 objects SODA: 1735ms

From the test results you can see that sorting makes the query slower, and adding additional constraints slows things down even more.

Database Size

Query performance can degrade with the amount of objects of the queried type:

```
QueryPerformanceBenchmark.cs: RunQueryAmountOfObjectsTest
private void RunQueryAmountOfObjectsTest()
         {
            Init();
            Clean();
            System.Console.WriteLine("Storing " + _count +
" of  objects of depth " + _depth);
            Open(Configure());
            Store();
            Close();

            //
            Open(Configure());
            StartTimer();
            IQuery query = objectContainer.Query();
            query.Constrain(typeof(Item));
            query.Descend("_name").Constrain("level1/1");
            Item item = (Item)query.Execute().Next();
            StopTimer("Select 1 object SODA: " + item._name);
            System.Console.WriteLine(
"Add some objects of another type and check the query time again:");
        StoreWithArray();
        Close();
        //
        Open(Configure());
        StartTimer();
        query = objectContainer.Query();
        query.Constrain(typeof(Item));
        query.Descend("_name").Constrain("level1/1");
        item = (Item) query.Execute().Next();
        StopTimer("Select 1 object SODA: " + item._name);
        Close();


        // Add many objects of the same type
            InitLargeDb();
            Clean();
            System.Console.WriteLine("Storing " + _count +
" of  objects of depth " + _depth);
            Open(Configure());
            Store();
            Close();

            //
            Open(Configure());
            StartTimer();
            query = objectContainer.Query();
            query.Constrain(typeof(Item));
            query.Descend("_name").Constrain("level1/1");
            item = (Item)query.Execute().Next();
            StopTimer("Select 1 object SODA: " + item._name);
            Close();
        }
```

```
QueryPerformanceBenchmark.cs: Init
private void Init()
         {
            _filePath = "performance.db4o";
            // amount of objects
            _count = 10000;
            // depth of objects
            _depth = 3;
            _isClientServer = false;

        }
```

```
QueryPerformanceBenchmark.cs: InitLargeDb
private void InitLargeDb()
         {
            _filePath = "performance.db4o";
            _count = 100000;
            _depth = 3;
            _isClientServer = false;

        }
```

```
QueryPerformanceBenchmark.cs: Configure
private IConfiguration Configure()
         {
            IConfiguration config = Db4oFactory.NewConfiguration();
            return config;
        }
```

However, the general size of the database, i.e. amount of other type of objects in the database should not have any impact on the query performance.

Results from the test machine:

Storing 10000 of objects of depth 3

Store 30000 objects: 3305ms

Select 1 object SODA: level1/1: 464ms

Storing 100000 of objects of depth 3

Store 300000 objects: 21338ms

Select 1 object SODA: level1/1: 5316ms

Complexity Of Objects

More complex objects are usually more difficult not only to store, but also to query and instantiate. The following test demonstrates how query performance depends on class structure, complexity and depth:

```
QueryPerformanceBenchmark.cs: RunDifferentObjectsTest
private void RunDifferentObjectsTest()
         {

            Init();
            System.Console.WriteLine("Storing " + _count +
" objects with " + _depth
                    + " levels of embedded objects:");

            Clean();
            System.Console.WriteLine();
            System.Console.WriteLine(" - primitive object with int field");
            Open(Configure());
            StoreSimplest();
            objectContainer.Ext().Purge();
            Close();
            Open(Configure());
            StartTimer();
            IQuery query = objectContainer.Query();
            query.Constrain(typeof(SimplestItem));
            query.Descend("_id").Constrain(1);
            IList result = query.Execute();
            SimplestItem simplestItem = (SimplestItem)result[0];
            StopTimer("Querying SimplestItem: " + simplestItem._id);
            Close();
            Open(Configure());
            System.Console.WriteLine();
            System.Console.WriteLine(" - object with string field");
            Store();
            objectContainer.Ext().Purge();
            Close();
            Open(Configure());
            StartTimer();
            query = objectContainer.Query();
            query.Constrain(typeof(Item));
            query.Descend("_name").Constrain("level1/2");
            result = query.Execute();
            Item item = (Item)result[0];
            StopTimer("Querying object with string field: " + item._name);
            Close();

            Clean();
            Open(Configure());
            System.Console.WriteLine();
            System.Console.WriteLine(" - object with StringBuilder field");
            StoreWithStringBuffer();
            objectContainer.Ext().Purge();
            Close();
            Open(Configure());
            StartTimer();
            query = objectContainer.Query();
            query.Constrain(typeof(ItemWithStringBuilder));
            query.Descend("_name").Constrain(new StringBuilder("level1/2"));
            result = query.Execute();
            ItemWithStringBuilder itemWithSB = (ItemWithStringBuilder)result[0];
            StopTimer("Querying object with StringBuilder field: "
                    + itemWithSB._name);
            Close();

            Clean();
            Open(Configure());
            System.Console.WriteLine();
            System.Console.WriteLine(" - object with int array field");
            StoreWithArray();
            objectContainer.Ext().Purge();
            Close();
            Open(Configure());
            StartTimer();
            query = objectContainer.Query();
            query.Constrain(typeof(ItemWithArray));
            IQuery idQuery = query.Descend("_id");
            idQuery.Constrain(1);
            idQuery.Constrain(2);
            idQuery.Constrain(3);
            idQuery.Constrain(4);
            result = query.Execute();

            ItemWithArray itemWithArray = (ItemWithArray)result[0];
            StopTimer("Querying object with Array field: [" + itemWithArray._id[0]
                    + ", " + +itemWithArray._id[1] + ", " + +itemWithArray._id[2]
                    + ", " + +itemWithArray._id[0] + "]");
            Close();

            Clean();
            Open(Configure());
            System.Console.WriteLine();
            System.Console.WriteLine(" - object with ArrayList field");
            StoreWithArrayList();
            objectContainer.Ext().Purge();
            Close();
            Open(Configure());
            StartTimer();
            query = objectContainer.Query();
            query.Constrain(typeof(ItemWithArrayList));
            query.Descend("_ids").Constrain(1).Contains();
            result = query.Execute();
            ItemWithArrayList itemWithArrayList = (ItemWithArrayList)result[0];
            StopTimer("Querying object with ArrayList field: "
                    + itemWithArrayList._ids.ToString());
            Close();

        }
```

```
QueryPerformanceBenchmark.cs: Init
private void Init()
         {
            _filePath = "performance.db4o";
            // amount of objects
            _count = 10000;
            // depth of objects
            _depth = 3;
            _isClientServer = false;

        }
```

```
QueryPerformanceBenchmark.cs: Configure
private IConfiguration Configure()
         {
            IConfiguration config = Db4oFactory.NewConfiguration();
            return config;
        }
```

```
QueryPerformanceBenchmark.cs: SimplestItem
public class SimplestItem
         {

            public int _id;
            public SimplestItem _child;

            public SimplestItem()
             {
            }

            public SimplestItem(int id, SimplestItem child)
             {
                _id = id;
                _child = child;
            }
        }
```

```
QueryPerformanceBenchmark.cs: ItemWithStringBuilder
public class ItemWithStringBuilder
         {

            public StringBuilder _name;
            public ItemWithStringBuilder _child;
            public ItemWithStringBuilder()
             {
            }

            public ItemWithStringBuilder(StringBuilder name,
                    ItemWithStringBuilder child)
             {
                _name = name;
                _child = child;
            }
        }
```

```
QueryPerformanceBenchmark.cs: ItemWithArray
public class ItemWithArray
         {

            public int[] _id;
            public ItemWithArray _child;

            public ItemWithArray()
             {
            }

            public ItemWithArray(int[] id, ItemWithArray child)
             {
                _id = id;
                _child = child;
            }
        }
```

```
QueryPerformanceBenchmark.cs: ItemWithArrayList
public class ItemWithArrayList
         {

            public ArrayList _ids;
            public ItemWithArrayList _child;

            public ItemWithArrayList()
             {
            }

            public ItemWithArrayList(ArrayList ids, ItemWithArrayList child)
             {
                _ids = ids;
                _child = child;
            }
        }
```

Results from the test machine:

- primitive object with int field

Store 30000 objects: 1878ms

Querying SimplestItem: 1: 425ms

- object with String field

Store 30000 objects: 2599ms

Querying object with String field: level1/2: 436ms

- object with StringBuffer field

Store 30000 objects: 5658ms

Querying object with StringBuffer field: level1/2: 3489ms

- object with int array field

Store 30000 objects: 2487ms

Querying object with Array field: [1, 2, 3, 1]: 1777ms

- object with ArrayList field

Store 30000 objects: 5302ms

Querying object with ArrayList field: [1, 2, 3, 4]: 3796ms

Inherited Objects

You can use a class base to query for inherited objects. This makes a query path a bit more complex and may result in a small performance degrade.

```
QueryPerformanceBenchmark.cs: RunInheritanceTest
private void RunInheritanceTest()
         {
            Init();
            Clean();
            System.Console.WriteLine("Storing " + _count +
" objects of depth " + _depth);
            Open(Configure());
            Store();
            Close();
            Open(Configure());
            StartTimer();
            IQuery query = objectContainer.Query();
            query.Constrain(typeof(Item));
            query.Descend("_name").Constrain("level1/1");
            Item item = (Item)query.Execute().Next();
            StopTimer("Select 1 object: " + item._name);
            Close();

            Clean();
            System.Console.WriteLine("Storing " + _count +
" inherited objects of depth "
                    + _depth);
            Open(Configure());
            StoreInherited();
            Close();
            Open(Configure());
            StartTimer();
            // Query for item, inheriting objects
            // should be included in the result
            query = objectContainer.Query();
            query.Constrain(typeof(Item));
            query.Descend("_name").Constrain("level1/1");
            item = (Item)query.Execute().Next();
            StopTimer("Select 1 object: " + item._name);
            Close();
        }
```

```
QueryPerformanceBenchmark.cs: ItemDerived
public class ItemDerived : Item
         {

            public ItemDerived(string name, ItemDerived child)
```

```
                 : base(name, child)
              {
             }
         }
```

```
QueryPerformanceBenchmark.cs: Init
private void Init()
         {
            _filePath = "performance.db4o";
            // amount of objects
            _count = 10000;
            // depth of objects
            _depth = 3;
            _isClientServer = false;

         }
```

```
QueryPerformanceBenchmark.cs: Configure
private IConfiguration Configure()
         {
            IConfiguration config = Db4oFactory.NewConfiguration();
            return config;
         }
```

Results from the test machine:

Storing 10000 objects of depth 3

Store 30000 objects: 2236ms

Select 1 object: level1/1: 457ms

Storing 10000 inherited objects of depth 3

Store 30000 objects: 2790ms

Select 1 object: level1/1: 595ms

Hardware Resources

Effective querying requires enough operating memory and quick hard drive access. Hard drive access time is important as the object will be read from the physical location into the operating memory. However, hard drive speed is not so critical for querying as it is for inserting.

The following test uses a RAM drive to compare test results with the hard drive:

```
QueryPerformanceBenchmark.cs: RunRamDiskTest
private void RunRamDiskTest()
         {

            InitForHardDriveTest();
            Clean();
            System.Console.WriteLine("Storing " + _count +
" objects of depth " + _depth
                    + " on a hard drive:");
            Open(ConfigureRamDrive());
            Store();
            Close();
            Open(ConfigureRamDrive());
            StartTimer();
            IQuery query = objectContainer.Query();
            query.Constrain(typeof(Item));
            query.Descend("_name").Constrain("level1/1");
            Item item = (Item)query.Execute().Next();
            StopTimer("Select 1 object: " + item._name);
            Close();

            InitForRamDriveTest();
            Clean();
            System.Console.WriteLine("Storing " + _count +
" objects of depth " + _depth
                    + " on a RAM disk:");
            Open(ConfigureRamDrive());
            Store();
            Close();
            Open(ConfigureRamDrive());
            StartTimer();
            query = objectContainer.Query();
            query.Constrain(typeof(Item));
            query.Descend("_name").Constrain("level1/1");
            item = (Item)query.Execute().Next();
            StopTimer("Select 1 object: " + item._name);
            Close();
        }
```

```
QueryPerformanceBenchmark.cs: InitForHardDriveTest
private void InitForHardDriveTest()
         {
            _count = 10000;
            _depth = 3;
            _filePath = "performance.db4o";
            _isClientServer = false;

        }
```

```
QueryPerformanceBenchmark.cs: ConfigureRamDrive
private IConfiguration ConfigureRamDrive()
         {
            IConfiguration config = Db4oFactory.NewConfiguration();
            config.FlushFileBuffers(true);
            return config;
        }
```

Test results:

Storing 30000 objects of depth 3 on a hard drive:

Store 90000 objects: 6019ms

Select 1 object: level1/1: 1515ms

Storing 30000 objects of depth 3 on a RAM disk:

Store 90000 objects: 5264ms

Select 1 object: level1/1: 1518ms

You can see that the difference in query performance is negligible.

Client-Server

In client-server use the connection speed can play an important role in the query performance.

```
QueryPerformanceBenchmark.cs: RunClientServerTest
private void RunClientServerTest()
         {

            InitForClientServer();
            Clean();
            System.Console.WriteLine("Storing " + _count +
" objects of depth " + _depth
                    + " remotely:");
            Open(ConfigureClientServer());
            Store();
            Close();
            Open(ConfigureClientServer());
            StartTimer();
            IQuery query = objectContainer.Query();
            query.Constrain(typeof(Item));
            query.Descend("_name").Constrain("level1/1");
            Item item = (Item)query.Execute().Next();
            StopTimer("Select 1 object: " + item._name);
            Close();

            Init();
            Clean();
            System.Console.WriteLine("Storing " + _count +
" objects of depth " + _depth
                    + " locally:");
            Open(ConfigureClientServer());
            Store();
            Close();
            Open(ConfigureClientServer());
            StartTimer();
            query = objectContainer.Query();
            query.Constrain(typeof(Item));
            query.Descend("_name").Constrain("level1/1");
            item = (Item)query.Execute().Next();
            StopTimer("Select 1 object: " + item._name);
            Close();
        }
```

```
QueryPerformanceBenchmark.cs: InitForClientServer
private void InitForClientServer()
         {
            _filePath = "performance.db4o";
            _isClientServer = true;
            _host = "localhost";
        }
```

```
QueryPerformanceBenchmark.cs: ConfigureClientServer
private IConfiguration ConfigureClientServer()
         {
            IConfiguration config = Db4oFactory.NewConfiguration();
            config.Queries().EvaluationMode(QueryEvaluationMode.Immediate);
            config.ClientServer().SingleThreadedClient(true);
            return config;
        }
```

Results from the test machine:

Storing 30000 objects of depth 3 remotely:

Store 90000 objects: 10725ms

Select 1 object: level1/1: 1763ms

Storing 10000 objects of depth 3 locally:

Store 30000 objects: 2904ms

Select 1 object: level1/1: 630ms

In order to improve the performance use Lazy or Snapshot evaluation modes.

Indexing

Using indexes is always a good idea to improve query performance. The following test illustrates index performance impact:

```
QueryPerformanceBenchmark.cs: RunIndexTest
private void RunIndexTest()
         {

            Init();
            System.Console.WriteLine("Storing " + _count +
" objects with " + _depth
                    + " levels of embedded objects:");

            Clean();
            System.Console.WriteLine(" - no index");
            Open(Configure());
            Store();
            Close();
            Open(Configure());
            StartTimer();
            IQuery query = objectContainer.Query();
            query.Constrain(typeof(Item));
            query.Descend("_name").Constrain("level1/2");
            IList result = query.Execute();
            Item item = (Item)result[0];
            StopTimer("Querying object with string field: " + item._name);
            Close();


            System.Console.WriteLine(" - index on string field");
            // Open to create index
            Open(ConfigureIndex());
            Close();
            Open(Configure());
            StartTimer();
            query = objectContainer.Query();
            query.Constrain(typeof(Item));
            query.Descend("_name").Constrain("level1/2");
            result = query.Execute();
            item = (Item)result[0];
            StopTimer("Querying object with string field: " + item._name);
            Close();
        }
```

```
QueryPerformanceBenchmark.cs: InitForHardDriveTest
private void InitForHardDriveTest()
         {
            _count = 10000;
            _depth = 3;
            _filePath = "performance.db4o";
            _isClientServer = false;

        }
```

```
QueryPerformanceBenchmark.cs: ConfigureRamDrive
private IConfiguration ConfigureRamDrive()
         {
            IConfiguration config = Db4oFactory.NewConfiguration();
            config.FlushFileBuffers(true);
            return config;
        }
```

Results from the test machine:

Storing 10000 objects with 3 levels of embedded objects:

- no index

Store 30000 objects: 2228ms

Querying object with String field: level1/2: 461ms

- index on String field

Querying object with String field: level1/2: 460ms

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 .NET
- Transient Classes
- Storing Transient Fields

Transient Fields In .NET

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:

```
FieldTransient.cs
/**//* Copyright (C) 2004 - 2007 Versant Inc. http://www.db4o.com */
using System;

namespace Db4objects.Db4odoc.SelectivePersistence
 {
  [AttributeUsage(AttributeTargets.Field)]
  public class FieldTransient: Attribute
   {
  }
}
```

```
FieldTransient.vb
' Copyright (C) 2004 - 2007 Versant Inc. http://www.db4o.com
Namespace Db4objects.Db4odoc.SelectivePersistence
    <AttributeUsage(AttributeTargets.Field)> _
    Public Class FieldTransient
        Inherits Attribute
    End Class
End Namespace
```

Let's use the newly-defined FieldTransient attribute and the system-provided Transient, and compare the results:

```
Test.cs
/**//* Copyright (C) 2004 - 2007 Versant Inc. http://www.db4o.com */
using System;
namespace Db4objects.Db4odoc.SelectivePersistence
 {
  public class Test
   {
        [Db4objects.Db4o.Transient]
        // you can also use [NonSerializedAttribute]
        string _transientField;
    string _persistentField;

    public Test(string transientField, string persistentField)
     {
      _transientField = transientField;
      _persistentField = persistentField;
    }

    public override string ToString()
     {
      return "Test: persistent: " + _persistentField + ",
transient: " + _transientField ;
    }
  }
}
```

```
Test.vb
' Copyright (C) 2004 - 2007 Versant Inc. http://www.db4o.com
Imports System
Imports Db4objects.Db4o

Namespace Db4objects.Db4odoc.SelectivePersistence
    Public Class Test
        <Transient()> Dim _transientField As String
        ' You can also use <NonSerialized()> Dim _transientField As String
        ' and mark the class <Serializable()>
        Dim _persistentField As String

        Public Sub New(ByVal transientField As String, _
ByVal persistentField As String)
            _transientField = transientField
            _persistentField = persistentField
        End Sub

        Public Overrides Function ToString() As String
            Return "Test: persistent: " + _persistentField + _
", transient: " + _transientField
        End Function
    End Class
End Namespace
```

```
TestCustomized.cs
/**//* Copyright (C) 2004 - 2007 Versant Inc. http://www.db4o.com */
namespace Db4objects.Db4odoc.SelectivePersistence
 {
  public class TestCustomized
   {
        [Db4objects.Db4odoc.SelectivePersistence.FieldTransient]
        string _transientField;
    string _persistentField;
    public TestCustomized(string transientField, string persistentField)
     {
      _transientField = transientField;
      _persistentField = persistentField;
    }

    public override string ToString()
     {
      return "Customized test: persistent: " + _persistentField +
", transient: " + _transientField ;
    }
  }
}
```

```
TestCustomized.vb
' Copyright (C) 2004 - 2007 Versant Inc. http://www.db4o.com

Namespace Db4objects.Db4odoc.SelectivePersistence
    Public Class TestCusomized
        <Db4objects.Db4odoc.SelectivePersistence.FieldTransient()> _
 Dim _transientField As String
        Dim _persistentField As String

        Public Sub New(ByVal transientField As String, _
ByVal persistentField As String)
            _transientField = transientField
            _persistentField = persistentField
        End Sub

        Public Overrides Function ToString() As String
            Return "Customized test: persistent: " + _persistentField + ", _
transient: " + _transientField
        End Function
    End Class
End Namespace
```

We will save and retrieve both Test and TestCustomized objects, having transient fields defined in different manner:

```
MarkTransientExample.cs: SaveObjects
private static void SaveObjects(IConfiguration configuration)
     {
      File.Delete(Db4oFileName);
            IObjectContainer container = Db4oFactory.OpenFile(configuration, Db4oFileName);
      try
       {
        Test test = new Test("Transient string","Persistent string");
        container.Store(test);
        TestCustomized testc = new TestCustomized("Transient string","Persistent string");
        container.Store(testc);
      }
      finally
       {
        container.Close();
      }
    }
```

```
MarkTransientExample.vb: SaveObjects
```

```
Public Shared Sub  SaveObjects(ByVal configuration As IConfiguration)
            File.Delete(Db4oFileName)
            Dim container As IObjectContainer = _
Db4oFactory.OpenFile(configuration, Db4oFileName)
            Try
                Dim test As Test = New Test("Transient string", _
"Persistent string")
                container.Store(test)
                Dim testc As TestCusomized = New _
TestCusomized("Transient string", "Persistent string")
                container.Store(testc)
            Finally
                container.Close()
            End Try
        End Sub
```

You will see the identical results independently of the way the transiency is defined.

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:

c#:

```
Db4objects.Db4o.Types. ITransientClass
```

VB:

```
Db4objects.Db4o.Types. ITransientClass
```

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.cs
using Db4objects.Db4o.Types;

namespace Db4objects.Db4odoc.SelectivePersistence
 {
    class NotStorable: ITransientClass
     {
        public override string ToString()
         {
            return "NotStorable class";
        }
    }
}
```

```
NotStorable.vb
' Copyright (C) 2004 - 2007 Versant Inc. http://www.db4o.com
Imports Db4objects.Db4o.Types

Namespace Db4objects.Db4odoc.SelectivePersistence
```

```
    Class NotStorable
        Implements ITransientClass

        Public Overloads Overrides Function  ToString() As String
            Return "NotStorable class"
        End Function
    End Class
End Namespace
```

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.cs: SaveObjects
private static void SaveObjects()
         {
            File.Delete(Db4oFileName);
            IObjectContainer container = Db4oFactory.OpenFile(Db4oFileName);
            try
             {
                // Save Test1 object with a NotStorable class field
                Test1 test1 = new Test1("Test1", new NotStorable());
                container.Store(test1);
                // Save Test2 object with a NotStorable class field
                Test2 test2 = new Test2("Test2", new NotStorable(), test1);
                container.Store(test2);
            }
            finally
             {
                container.Close();
            }
        }
```

```
TransientClassExample.vb: SaveObjects
Public Shared Sub SaveObjects()
            File.Delete(Db4oFileName)
            Dim container As IObjectContainer = Db4oFactory.OpenFile(Db4oFileName)
            Try
                ' Save Test1 object with a NotStorable class field
                Dim test1 As Test1 = New Test1("Test1", New NotStorable)
                container.Store(test1)
                ' Save Test2 object with a NotStorable class field
                Dim test2 As Test2 = New Test2("Test2", New NotStorable, test1)
                container.Store(test2)
            Finally
                container.Close()
            End Try
        End Sub
```

Now let's try to retrieve the saved objects:

```
TransientClassExample.cs: RetrieveObjects
private static void RetrieveObjects()
         {
            IObjectContainer container = Db4oFactory.OpenFile(Db4oFileName);
            try
             {
                // retrieve the results and check if the NotStorable instances were saved
                IList result = container.QueryByExample(null);
                ListResult(result);
            }
            finally
             {
                container.Close();
            }
        }
```

```
TransientClassExample.vb: RetrieveObjects
Public Shared Sub RetrieveObjects()
            Dim container As IObjectContainer = Db4oFactory.OpenFile(Db4oFileName)
            Try
                ' retrieve the results and check if the
                ' NotStorable instances were saved
                Dim result As IList = container.QueryByExample(Nothing)
                ListResult(result)
            Finally
                container.Close()
            End Try
        End Sub
```

If you will run the example code you will see that all the instances of NotStorable class are set to null.

Test1

```
Test1.cs
class Test1
{
       private string name;
       private NotStorable transientClass;

       public Test1(string name, NotStorable transientClass)
       {
               this.name = name;
               this.transientClass = transientClass;
       }

       public override string ToString()
       {
               if (transientClass == null)
               {
                       return string.Format("{0}/{1}", name, "null");
               }
               else
               {
                       return string.Format("{0}/{1}", name, transientClass);
               }
       }
}
```

```
Test1.vb
Class Test1
       Private name As String
       Private transientClass As NotStorable

       Public Sub New(ByVal name As String, ByVal transientClass As NotStorable)
               Me.name = name
               Me.transientClass = transientClass
       End Sub
       Public Overloads Overrides Function ToString() As String
               If transientClass Is Nothing Then
                       Return String.Format("{0}/{1}", name, "Nothing")
               Else
                       Return String.Format("{0}/{1}", name, transientClass.ToString())
               End If
       End Function
End Class
```

Test2

```
Test2.cs
class Test2
{
       private Test1 test1;
       private string name;
       private NotStorable transientClass;


       public Test2(string name, NotStorable transientClass, Test1 test1)
       {
               this.test1 = test1;
               this.name = name;
               this.transientClass = transientClass;
       }

       public override string ToString()
       {
               if (transientClass == null)
               {
                       return string.Format("{0}/{1}; test1: {2}", name, "null", test1);
               }
               else
               {
                       return string.Format("{0}/{1}; test1: {2}", name, transientClass, test1);
               }
       }
}
```

```
Test2.vb
Class Test2
       Private test1 As Test1
       Private name As String
       Private transientClass As NotStorable

       Public Sub New(ByVal name As String, _
ByVal transientClass As NotStorable, ByVal test1 As Test1)
               Me.test1 = test1
               Me.name = name
               Me.transientClass = transientClass
       End Sub

       Public Overloads Overrides Function ToString() As String
               If transientClass Is Nothing Then
                       Return String.Format("{0}/{1}; test1: {2}", name, _
 "Nothing", test1.ToString())
```

```
               Else
                       Return String.Format("{0}/{1}; test1: {2}", name, _
 transientClass.ToString(), test1.ToString())
               End If
       End Function
End Class
```

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:

- Play with the different prefetching options on the client configuration.
- Enable Field Indexes
- Discarding Free Space
- Calling constructors
- Defragment
- Turning Off Weak References
- No callbacks
- No schema changes
- No lock file thread
- No test instances
- Increasing The Maximum Database File Size
- B-Tree tuning
- Inheritance hierarchies
- Persistent and transient fields
- Activation strategies
- Chose an appropriate string encoding.
- No Class Index
- Change the Storage mechanism used by db4o
- Commit Strategies
- Database Size
- Change the underlying Storage

Discarding Free Space

.NET: `configuration.File.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:

.NET: `Db4objetcs.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 with frequent defragment runs.

Defragment

.NET: `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.

Activation strategies

.NET:
```
configuration.Common.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:

.NET: `IObjectContainer#Activate(object,depth)`

or use specific object settings:

.NET:

```
configuration.Common.ObjectClass("yourClass").
MinimumActivationDepth(minimumDepth);
```

```
configuration.Common.ObjectClass("yourClass").
MaximumActivationDepth(maximumDepth);
```

```
configuration.Common.ObjectClass("yourClass").
CascadeOnActivate(bool);
```

```
configuration.Common.ObjectClass("yourClass").
ObjectField("field").CascadeOnActivate(bool);
```

For more information on activation strategies see Activation chapter.

No Class Index

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.ObjectClass(typeof (Person)).Indexed(false);
```

ObjectConfigurationExamples.cs: Disable class index

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.ObjectClass(GetType(Person)).Indexed(False)
```

ObjectConfigurationExamples.vb: Disable class index

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.

Commit Strategies

.NET:
```
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 object container is closed.

.NET:
```
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 object container before the application exits to make sure that all the changes will be saved to disk during implicit commit.

Increasing The Maximum Database File Size

.NET:
```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.File.BlockSize = newBlockSize;
DefragmentConfig defragConfig = new DefragmentConfig("database.db4o");
defragConfig.Db4oConfig(configuration);
Defragment.Defrag(defragConfig);
```

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, 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 object containers.

Database Size

If you are concerned about the size of your database file, it is impotrnant to understand what contributes to it and what are the strategies to keep it down.

Object Overhead

When you create a new db4o fatabase file - it contains only the header and has a fixed size. As soon as you start storing the information the file will grow. The size overhead per object depends on the type-handler implementation.

In general the object consists of internal ID and value types, i.e. integers, arrays, enums etc. Overhead per object type is ID, which is integer. The overhead for value type is an integer value showing which value type is it, i.e. int or string etc.For variable length value types, there is a long value to store the length. If object contains another complex object - the id of another object is referenced in the top-level object. If you decide to use UUIDs and version number for your objects, you will get an additional over-head:

UUID = 35 bytes (signature part) + 8 bytes (long part) version number = 8 bytes.

Additional overhead per object will appear from using indexes and will depend on the amount of indexes fields and indexes value types.

Block Size

Block Size is a configurable value, which defines the way information is stored in db4o database. Using bigger block sizes can result in unnecessary growth of the database. For motr information see Increasing The Maximum Database File Size

Freespace

Freespace appears in db4o database after unneeded objects were deleted. The amount of the freespace can be controlled from the configuration. Another option to get rid of the freespace is Defragment. It is a good practice to run Defragment regularly to maintain the minimum database file size.

Inheritance hierarchies

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.

Persistent and transient fields

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:

.NET: `public class NotStorable { [Transient] private 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.

Runtime Monitoring

The db4o runtime statistics is a monitoring feature allowing to collect various important runtime data. This data can be crucial in resolving performance issues, analyzing usage patterns, predicting resource bottlenecks etc. Runtime Statistics can be collected both in the application testing stage and in an application deployed to production system. In the first case this data can help to estimate hardware requirements and analyze the stability of the system. In the latter case, the data can be used to fine-tune performance, find problems and fix bugs. Take a look how you install the statistics and monitor them.

Install And Monitor

On the .NET platform all runtime statistics are published through the Windows Performance Counters. This means that you can use the regular windows tooling to monitor db4o statistics. See "Install and Monitor" on page 321

Available Statistics

Currently following statistics are available.

- Query Monitoring: Monitor how queries behave and perform.
- Object Lifecycle Monitoring: Monitor how many objects are stored, deleted and activated.
- IO Monitoring: Monitor the IO-operations of db4o.
- Network Monitoring: Monitor the network operations of db4o.
- Reference System Monitoring: Monitor db4o's reference system.

Install and Monitor

On the .NET platform all runtime statistics are published through the Windows Performance Counters. This means that you can use the regular windows tooling to monitor db4o statistics.

Installing the Monitoring Support

The first thing we need to do is to add the monitoring support to the db4o configuration. The monitoring-support are in the optional-assemblys. So you need to add those for the monitoring support.

Monitoring adds a small overhead to the regular db4o operations. Therefore the monitoring support is distributed across different monitoring options, so that you can add only the options you need.

Currently following options are available:

- Query Monitoring: Monitor how queries behave and perform.
- Object Lifecycle Monitoring: Monitor how many objects are stored, deleted and activated.
- IO Monitoring: Monitor the IO-operations of db4o.
- Network Monitoring: Monitor the network operations of db4o.
- Reference System Monitoring: Monitor db4o's reference system.

Install the Performance Counters

In order to monitor db4o you need to install the db4o performance counters. This needs to be done only once on the machine you want to monitor. Installing the performance counters require Administrator privileges.

You can install the performance counter either from you application with an simple method call or with the db4oTool.

Installing From Your Application

You can install it by simply calling the install-method on the Db4oPerformanceCounters class. Not that your application needs to run with Administrator privileges to successfully install the performance counters.

```
Db4oPerformanceCounters.Install();
```

InstallPerformanceCounters.cs: Install the performance counters

```
Db4oPerformanceCounters.Install()
```

InstallPerformanceCounters.vb: Install the performance counters

Installing With Db4oTool

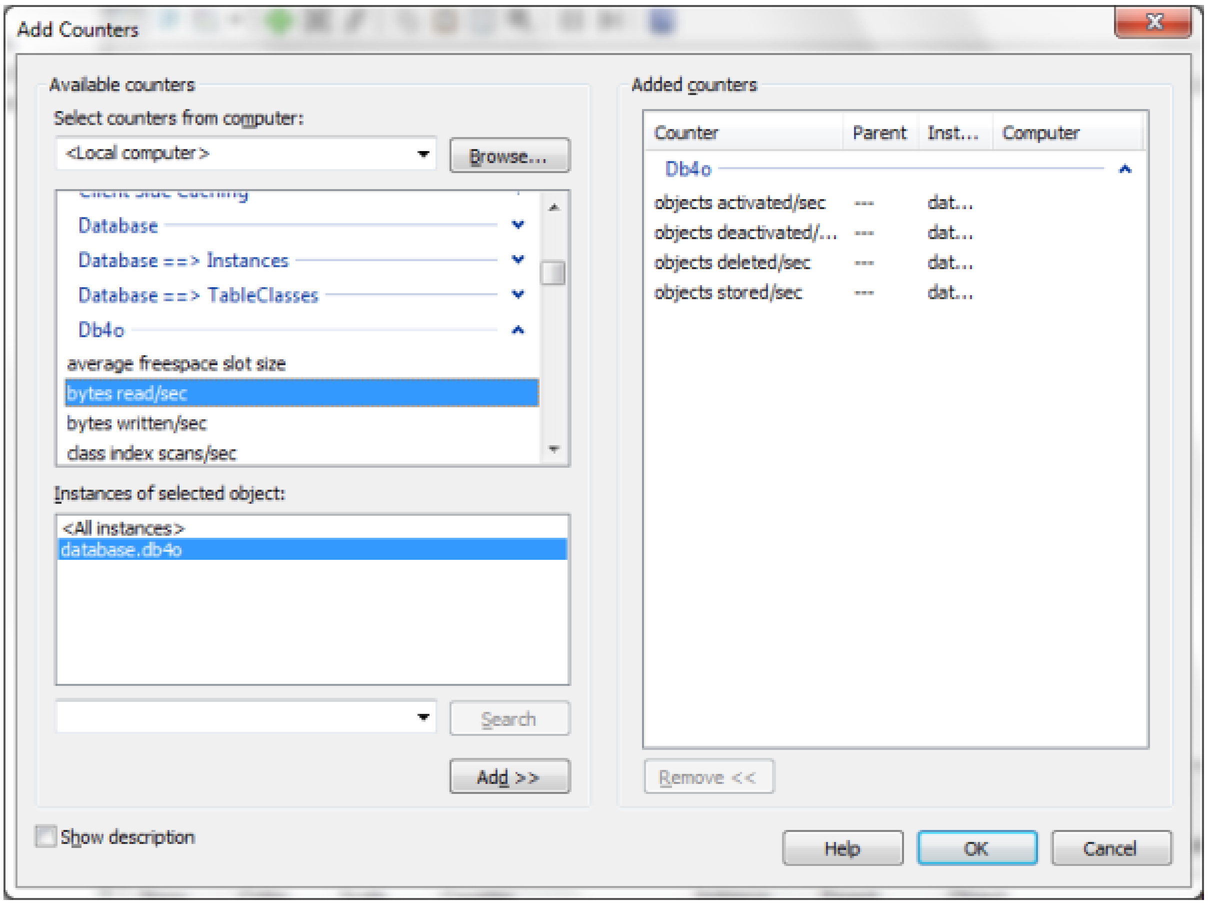
You can install it with the db4oTool. Note that the Db4oTool needs to be executed with Administrator privileges to successfully install the performance counters.

```
Db4oTool.exe -install-performance-counters
```

Monitor With Performance Monitor

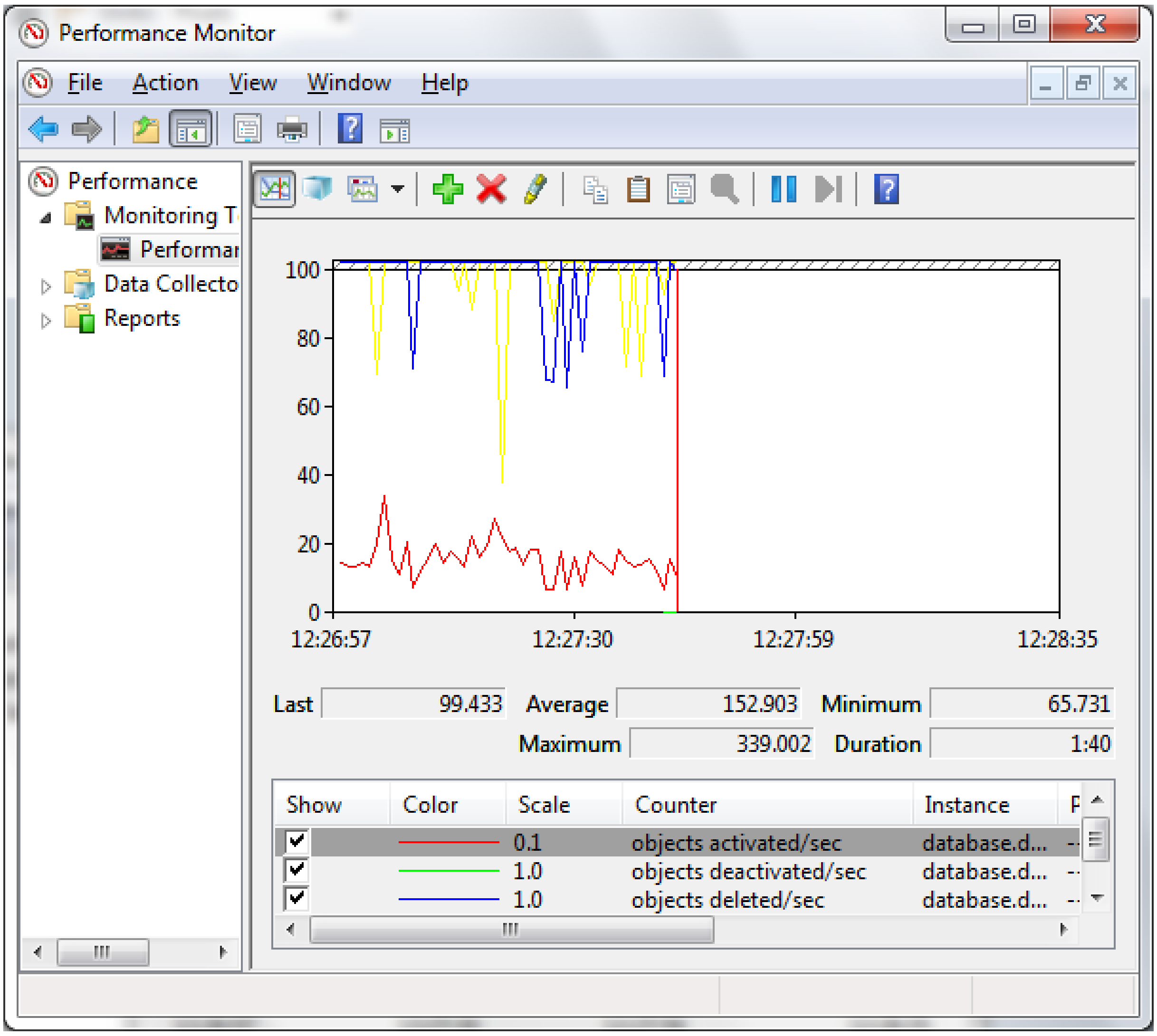
To monitor the db4o performance counters, you can use the Windows Performance Monitor (perfom.exe). You can start the Performance Monitor by opening the Run-dialog (Windows-Key + R) and enter 'perfmon' and execute.

First select which performance counters you want to monitor and for which db4o instance. Click the 'Add'-Botton on the toolbar. There scroll to the db4o-category and select the performance-counter and the instance you want to monitor.



Counter	Parent	Inst...	Computer
Db4o			
objects activated/sec	---	dat...	
objects deactivated/...	---	dat...	
objects deleted/sec	---	dat...	
objects stored/sec	---	dat...	

After that, you can watch the performance-counters. Read the Performance Monitor reference for more information.



Last	99.433	Average	152.903	Minimum	65.731
		Maximum	339.002	Duration	1:40

Show	Color	Scale	Counter	Instance
✓	🟥	0.1	objects activated/sec	database.d...
✓	🟩	1.0	objects deactivated/sec	database.d...
✓	🟦	1.0	objects deleted/sec	database.d...

Monitoring Queries

You can monitor queries to find out more about the runtime behavior of your application.

Configure the Query Monitoring Support

First you need to add the monitoring support to the db4o configuration. There are two separate items. The QueryMonitoringSupport will monitor the very basic query operations. The NativeQueryMonitoringSupport adds additional statistics about LINQ and native queries.

```
configuration.Common.Add(new QueryMonitoringSupport());
configuration.Common.Add(new NativeQueryMonitoringSupport());
```

QueryMonitoring.cs: Add query monitoring

```
configuration.Common.Add(New QueryMonitoringSupport())
configuration.Common.Add(New NativeQueryMonitoringSupport())
```

QueryMonitoring.vb: Add query monitoring

The Query Statistics

class index scans/sec: Tells you the number of queries which required to scan through all objects. This means that a query couldn't use a field index and therefore required to go through all objects. This is of course slow. You should try to keep this number low by adding the right indexes on fields. See "Indexing" on page 102

queries/sec: Tells you how many queries run per second.

The Native Query Statistics

native queries/sec: Tells you how many native queries per second run.

unoptimized native queries/sec: Tells you how many unoptimized native queries run per second. Such queries need to instantiate all objects which is a slow operation. If this number is high, you should try to simplify your queries.

linq queries/sec: Tells you how many db4o-LINQ queries run per second.

unoptimized linq queries/sec: Tells you how many unoptimized LINQ run per second . You should avoid unoptimized LINQ-queries, because these queries instantiate allobjects to perform the query and therefore are slow. Try to write a more simple LINQ-query. See "LINQ" on page 13

Monitor Object Lifecycle

You can monitor the object lifecycle statistics of db4o to find out more about the runtime behavior of your application.

Configure the Object Lifecycle Monitoring Support

In order to monitor the object lifecycle statistics, you need to add the monitoring support to the configuration.

```
configuration.Common.Add(new ObjectLifecycleMonitoringSupport());
```

ObjectLifecycleMonitoring.cs: Monitor the object lifecycle statistics

```
configuration.Common.Add(New ObjectLifecycleMonitoringSupport())
```

ObjectLifecycleMonitoring.vb: Monitor the object lifecycle statistics

The Object Lifecycle Statistics

objects activated/sec: Tells you how many objects are activated per second. Activation can consume a lot of time for complex object graphs. If there's a lot of time spend with activating objects, you may want to reduce the amount of activated objects. One of the best way to activate only the minimum set of objects is to use transparent activation.

objects deactivated/sec: Tells you how many objects are deactivate per second.

objects deleted/sec: Tells you how many objects are deleted per second.

objects stored/sec: Tells you how many objects are stored per second.

Monitor IO

You can monitor the IO-operations of db4o to find out more about the runtime behavior of your application.

Configure the IO Monitoring Support

In order to monitor the IO activities of db4o you need to configure the IO monitoring support.

```
configuration.Common.Add(new IOMonitoringSupport());
```

IOMonitoring.cs: Add IO-Monitoring

```
configuration.Common.Add(New IOMonitoringSupport())
```

IOMonitoring.vb: Add IO-Monitoring

The IO Statistics

bytes read/sec: Tells you how many bytes are read per second from the disk.

bytes written/sec: Tells you how many bytes are written to disk per second.

Monitor Freespace

You can monitor the freespace-manager to find out more about the runtime behavior of your application.

Configure the Freespace Monitoring Support

To monitor the freespace manager you need to add the monitoring support to the db4o configuration.

```
configuration.Common.Add(new FreespaceMonitoringSupport());
```

FreespaceMonitoring.cs: Monitor the free-space system

```
configuration.Common.Add(New FreespaceMonitoringSupport())
```

FreespaceMonitoring.vb: Monitor the free-space system

The Freespace Statistics

average freespace slot size: Tells you how big the average slot is. When you store larger objects, they need larger slots to fit it.

reused freespace slots/sec: Tells you how many slots can be reused. When an object is deleted or modified, its old slot is released and can be reused. If you modify and delete a lot of objects, but the slots cannot be reused, then the database-file will fragment.

When this number is low but the free-space increases and increases then you have a fragmentation issue. Defragment your database. Use the btree-id-system.

number of freespace slots: Tells you how many slots are used by the database. As more and more objects are stored, the slot count will increase.

total freespace: Tells you how much freespace there is in the database-file.A hight free-space number with a low reused slots number indicates database fragmentation. Defragment your database. Use the btree-id-system.

Monitor Network

You can monitor the network-operations of db4o in client-server mode.

Configure the Network Monitoring Support

First you need to add the monitoring support to the db4o configuration. The network monitoring support is in two separate configuration-items. The NetworkingMonitoringSupport can be used on the server and client to monitor the network statistics.

```
configuration.Common.Add(new NetworkingMonitoringSupport());
```

CSMonitoring.cs: Add the network monitoring support

```
configuration.Common.Add(New NetworkingMonitoringSupport())
```

CSMonitoring.vb: Add the network monitoring support

The ClientConnectionsMonitoringSupport can be added to the db4o server to monitor the connected clients.

```
configuration.AddConfigurationItem(new ClientConnectionsMonitoringSupport());
```

CSMonitoring.cs: Add the client connections monitoring support

```
configuration.AddConfigurationItem(New ClientConnectionsMonitoringSupport())
```

CSMonitoring.vb: Add the client connections monitoring support

The Network Statistics

network bytes sent/sec: Tells you how many bytes this client or server has receives per second.

network bytes received/sec: Tells you how many bytes this client or server sends per second.

network messages sent/sec: Tells you how many messages this client or server sends per second.

number of connected clients: Tells you how many clients are connected to this server.

Monitor Reference System

You can monitor the reference-system to find out more about the runtime behavior of your application. The reference-system ensures that each object has only one in memory representation.

Configure the Reference System Monitoring Support

First you need to add the monitoring support to the db4o configuration.

```
configuration.Common.Add(new ReferenceSystemMonitoringSupport());
```

ReferenceSystemMonitoring.cs: Add reference system monitoring

```
configuration.Common.Add(New ReferenceSystemMonitoringSupport())
```

ReferenceSystemMonitoring.vb: Add reference system monitoring

The Freespace Statistics

number of object references: Tells you how many references are currently hold in the reference-system. By default db4o uses weak references to objects. If this count is very high, you might hold unnecessary references to persisted objects in your application.

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
- Reading Db4o File

Debug Messaging System

You can turn on the db4o debug messages, which show more information about what db4o is doing.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.MessageLevel = 4;
```

CommonConfigurationExamples.cs: Change the message-level

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.MessageLevel = 4
```

CommonConfigurationExamples.vb: Change the message-level

These messages show you in detail what db4o is doing and processing. You can redirect the message output to any output stream. By default the console output stream is used.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.OutStream = Console.Out;
```

CommonConfigurationExamples.cs: Change the output stream

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.OutStream = Console.Out
```

CommonConfigurationExamples.vb: Change the output stream

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
Sub SearchOffset()
    Dim pos As Integer
    pos = Val(Selection.Text)
    If pos = 0 Then
        MsgBox ("The selection is not a number")
    Else
        ActiveDocument.Content.Characters(pos).Select
    End If
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.

Diagnostics

The db4o engine provides user with a special mechanism showing runtime diagnostics information. Diagnostics can be switched on in the configuration before opening the database file:

The DiagnosticListener is a callback interface tracking diagnostic messages from different parts of the system.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.Diagnostic.AddListener(new DiagnosticToConsole());
```

CommonConfigurationExamples.cs: Add a diagnostic listener

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.Diagnostic.AddListener(New DiagnosticToConsole())
```

CommonConfigurationExamples.vb: Add a diagnostic listener

Built-in Listeners

There are two build in listeners, which print the output to the console

- DiagnosticToConsole:Prints diagnostic messages to the console.
- DiagnosticToTrace: Only on .NET, prints diagnostic messages to the debug output window.

Messages-Types

Every diagnostic message is represented by it's own type. You can filter the messages by checking for certain instances. Take a look how you can filter for certain messages.

- **MissingClass**: Notifies you that a class couldn't be found. You should add that class in order to avoid problems. If you've renamed the class, you should rename it in the database or add an alias.
- **DefragmentRecommendation**: Notifies you that you should consider to defragment the database.
- **LoadedFromClassIndex**: Notifies you that db4o couldn't use a field-index to perform a query. This means that the query runs extremely slow on larger data sets. Consider adding a field-index.
- **DescendIntoTranslator**: Means that you a query couldn't be optimized, because the query touches a class with a translator. Therefore the query runs slow. You should consider working without a translator or changing the query.
- **ClassHasNoFields**: You stored a class which has no fields. Even when the class has no fields it need to maintain its class index and needs to be stored. So it adds overhead for storing 'empty' objects. You should consider removing the empty class.
- **DeletionFailed**: db4o failed to delete a object from the database.
- **UpdateDepthGreaterOne**: You have configured a update depth greater one. A large update depth slows down updates significantly. Consider reducing the update-depth and use another strategies to update objects correctly, like transparent persistence.
- **NativeQueryNotOptimized**: A native query couldn't be optimized. An unoptimized query runs significantly slower than a optimized query. Consider to simplify you're query.

- **NativeQueryOptimizerNotLoaded**: Couldn't load the native query optimizer. Ensure that all required assembly are added to your application.
- **NotTransparentActivationEnabled**: Notifies you when a class doesn't support transparent activation. Such object need to be fully activated and slow down the activation process.
- **ObjectFieldDoesNotExist**: A query uses a object-field which doesn't exist. Check you're queries to use only exising fields.

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 giving information about queries that are running on un-indexed fields. By 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:

```
private class DiagnosticFilter : IDiagnosticListener
{
    private readonly ICollection<Type> filterFor;
    private readonly IDiagnosticListener target;

    public DiagnosticFilter(IDiagnosticListener target, params Type[] filterFor)
    {
        this.target = target;
        this.filterFor = new List<Type>(filterFor);
    }

    public void OnDiagnostic(IDiagnostic diagnostic)
    {
        Type type = diagnostic.GetType();
        if (filterFor.Contains(type))
        {
            target.OnDiagnostic(diagnostic);
        }
    }
}
```

DiagnosticsExamples.cs: A simple message filter

```
Private Class DiagnosticFilter
    Implements IDiagnosticListener
    Private ReadOnly filterFor As ICollection(Of Type)
    Private ReadOnly target As IDiagnosticListener

    Public Sub New(ByVal target As IDiagnosticListener, ByVal ParamArray filterFor As Type())
        Me.target = target
        Me.filterFor = New List(Of Type)(filterFor)
    End Sub

    Public Sub OnDiagnostic(ByVal diagnostic As IDiagnostic) _
        Implements IDiagnosticListener.OnDiagnostic

        Dim type As Type = diagnostic.[GetType]()
        If filterFor.Contains(type) Then
            target.OnDiagnostic(diagnostic)
        End If
    End Sub
End Class
```

DiagnosticsExamples.vb: A simple message filter

After that we can use the filter-listener. It takes two arguments. The first one is a regular listener, the second is a list of all messages which are passed through.

```
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.Diagnostic
    .AddListener(new DiagnosticFilter(new DiagnosticToConsole(), typeof(LoadedFromClassIndex)));
```

DiagnosticsExamples.cs: Filter for unindexed fields

```
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.Diagnostic _
    .AddListener(New DiagnosticFilter(New DiagnosticToConsole(), GetType(LoadedFromClassIndex)))
```

DiagnosticsExamples.vb: Filter for unindexed fields

Utility Methods

In this chapter we will review utility methods provided by extended object container interface. Use API documentation for more information on ExtObjectContainer methods.

More Reading:

- PeekPersisted
- IsActive
- IsStored
- Descend

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:

c#: `IExtObjectContainer.PeekPersisted(object, depth, committed)`

VB: `IExtObjectContainer.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.cs: MeasureCarTemperature
private static void MeasureCarTemperature()
     {
      SetObjects();
      IObjectContainer db = Db4oFactory.OpenFile(Db4oFileName);
      try
       {
        IObjectSet result = db.QueryByExample(typeof(Car));
        if (result.Size() > 0)
         {
          Car car = (Car)result[0];
          Car car1  = (Car)db.Ext().PeekPersisted(car, 5, true);
          Change1 ch1 = new Change1();
          ch1.Init(car1);
          Car car2  = (Car)db.Ext().PeekPersisted(car, 5, true);
          Change2 ch2 = new Change2();
          ch2.Init(car2);
          Thread.Sleep(300);
          // We can work on the database object at the same time
          car.Model = "BMW M3Coupe";
          db.Store(car);
          ch1.Stop();
          ch2.Stop();
          System.Console.WriteLine("car1 saved to the database: "
+ db.Ext().IsStored(car1));
          System.Console.WriteLine("car2 saved to the database: "
+ db.Ext().IsStored(car1));
          int temperature = (int)((car1.Temperature + car2.Temperature)/2);
          car.Temperature = temperature;
          db.Store(car);
        }
      }
      finally
       {
        db.Close();
      }
      �heckCar();
    }
```

```
PeekPersistedExample.vb: MeasureCarTemperature
Private Shared Sub MeasureCarTemperature()
```

```
            SetObjects()
            Dim db As IObjectContainer = Db4oFactory.OpenFile(Db4oFileName)
            Try
                Dim result As IObjectSet = db.QueryByExample(GetType(Car))
                If result.Size() > 0 Then
                    Dim car As Car = CType(result(0), Car)
                    Dim car1 As Car = CType(db.Ext().PeekPersisted(car, 5, True), Car)
                    Dim ch1 As Change1 = New Change1()
                    ch1.Init(car1)
                    Dim car2 As Car = CType(db.Ext().PeekPersisted(car, 5, True), Car)
                    Dim ch2 As Change2 = New Change2()
                    ch2.Init(car2)
                    Thread.Sleep(300)
                    ' We can work on the database object at the same time
                    car.Model = "BMW M3Coupe"
                    db.Store(car)
                    ch1.Kill()
                    ch2.Kill()
                    System.Console.WriteLine("car1 saved to the database: " _
+ db.Ext().IsStored(car1).ToString())
                    System.Console.WriteLine("car2 saved to the database: " _
+ db.Ext().IsStored(car1).ToString())
                    Dim temperature As Integer = CType(((car1.Temperature _
+ car2.Temperature) / 2), Integer)
                    car.Temperature = temperature
                    db.Store(car)
                End If
            Finally
                db.Close()
            End Try
            CheckCar()
        End Sub
```

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 you will get a new object in the database.

IsActive

ExtObjectContainer.isActive method provides you with means to define if the object is active.

```
UtilityExample.cs: CheckActive
public static void CheckActive()
     {
      StoreSensorPanel();
            IConfiguration configuration = Db4oFactory.NewConfiguration();
            configuration.ActivationDepth(2);
      IObjectContainer db = Db4oFactory.OpenFile(configuration, Db4oFileName);
      try
       {
        System.Console.WriteLine("Object container activation depth = 2");
        IObjectSet result = db.QueryByExample(new SensorPanel(1));
        SensorPanel sensor = (SensorPanel)result[0];
        SensorPanel next = sensor.Next;
        while (next != null)
         {
          System.Console.WriteLine("Object " + next +" is active: "
+ db.Ext().IsActive(next));
          next = next.Next;
        }
      }
      finally
       {
        db.Close();
      }
    }
```

```
UtilityExample.vb: CheckActive
Public Shared Sub CheckActive()
            StoreSensorPanel()
            Dim configuration As IConfiguration = Db4oFactory.NewConfiguration()
            configuration.ActivationDepth(2)
            Dim db As IObjectContainer = Db4oFactory.OpenFile _
(configuration, Db4oFileName)
            Try
                System.Console.WriteLine("Object container activation depth = 2")
                Dim result As IObjectSet = db.QueryByExample(New SensorPanel(1))
                Dim sensor As SensorPanel = CType(result(0), SensorPanel)
                Dim NextSensor As SensorPanel = sensor.NextSensor
                While Not NextSensor Is Nothing
                    System.Console.WriteLine("Object " + _
NextSensor.ToString() + " is active: " + _
db.Ext().IsActive(NextSensor).ToString())
                    NextSensor = NextSensor.NextSensor
                End While
            Finally
                db.Close()
            End Try
        End Sub
```

This method can be useful in applications with deep object hierarchy if you prefer to use manual activation.

IsStored

ExtObjectContainer#isStored helps you to define if the object is stored in the database. The following example shows how to use it:

```
UtilityExample.cs: CheckStored
public static void CheckStored()
     {
      // create a linked list with length 10
      SensorPanel list = new SensorPanel().CreateList(10);
      File.Delete(Db4oFileName);
      IObjectContainer db = Db4oFactory.OpenFile(Db4oFileName);
      try
       {
        // store all elements with one statement, since all elements are new
        db.Store(list);
        Object sensor = (Object)list.Sensor;
        SensorPanel sp5 = list.Next.Next.Next.Next;
        System.Console.WriteLine("Root element "+list+" isStored: "
+ db.Ext().IsStored(list));
        System.Console.WriteLine("Simple type  "+sensor+" isStored: "
+ db.Ext().IsStored(sensor));
        System.Console.WriteLine("Descend element  "+sp5+" isStored: "
+ db.Ext().IsStored(sp5));
        db.Delete(list);
        System.Console.WriteLine("Root element "+list+" isStored: "
+ db.Ext().IsStored(list));
```

```
      }
      finally
       {
        db.Close();
      }
    }
```

```
UtilityExample.vb: CheckStored
Public Shared Sub CheckStored()
            ' create a linked list with length 10
            Dim list As SensorPanel = New SensorPanel().CreateList(10)
            File.Delete(Db4oFileName)
            Dim db As IObjectContainer = Db4oFactory.OpenFile(Db4oFileName)
            Try
                ' store all elements with one statement,
                ' since all elements are new
                db.Store(list)
                Dim sensor As Object = CType(list.Sensor, Object)
                Dim sp5 As SensorPanel = list.NextSensor.NextSensor. _
NextSensor.NextSensor
                System.Console.WriteLine("Root element " + list.ToString() _
+ " isStored: " + db.Ext().IsStored(list).ToString())
                System.Console.WriteLine("Simple type  " + sensor.ToString() _
+ " isStored: " + db.Ext().IsStored(sensor).ToString())
                System.Console.WriteLine("Descend element  " + sp5.ToString() _
+ " isStored: " + db.Ext().IsStored(sp5).ToString())
                db.Delete(list)
                System.Console.WriteLine("Root element " + list.ToString() _
+ " isStored: " + db.Ext().IsStored(list).ToString())
            Finally
                db.Close()
            End Try
        End Sub
```

Descend

ExtObjectContainer#descend method allows you to navigate from a persistent object to it's members without activating or instantiating intermediate objects.

```
UtilityExample.cs: TestDescend
public static void TestDescend()
     {
      StoreSensorPanel();
            IConfiguration configuration = Db4oFactory.NewConfiguration();
            configuration.ActivationDepth(1);
            IObjectContainer db = Db4oFactory.OpenFile(
configuration, Db4oFileName);
      try
       {
        System.Console.WriteLine("Object container activation depth = 1");
        IObjectSet result = db.QueryByExample(new SensorPanel(1));
        SensorPanel spParent = (SensorPanel)result[0];
        SensorPanel spDescend = (SensorPanel)db.Ext().
Descend((Object)spParent, new String[] {"_next","_next",
"_next","_next","_next"});
        db.Ext().Activate(spDescend, 5);
        System.Console.WriteLine(spDescend);
      }
      finally
```

```
        {
         db.Close();
       }
     }
```

```
UtilityExample.vb: TestDescend
Public Shared Sub TestDescend()
            StoreSensorPanel()
            Dim configuration As IConfiguration = Db4oFactory.NewConfiguration()
            configuration.ActivationDepth(1)
            Dim db As IObjectContainer = Db4oFactory.OpenFile(configuration, _
Db4oFileName)
            Try
                System.Console.WriteLine("Object container activation depth = 1")
                Dim result As IObjectSet = db.QueryByExample(New SensorPanel(1))
                Dim spParent As SensorPanel = CType(result(0), SensorPanel)
                Dim fields() As String = {"_next", "_next", "_next", "_next", "_next"}
                Dim spDescend As SensorPanel = CType(db.Ext().Descend( _
CType(spParent, Object), fields), Object)
                db.Ext().Activate(spDescend, 5)
                System.Console.WriteLine(spDescend)
            Finally
                db.Close()
            End Try
        End Sub
```

Navigating in this way can save you resources on activating only the objects you really need.

Usage Pitfalls

The db4o team tries hard to make the db4o as easy to use as possible. However there are still some pitfalls.

When objects are only partially loaded and you run into null pointer exceptions, then you ran into not activated objects. This is a common issue. See "The Activation Pitfall" on page 336

When updates are not stored then you have issues with the update-depth.See "Update Depth Pitfall" on page 337

db4o tries hard to store every object without any mapping. However in practice the mapping is not perfect. .For example remoting objects shouldn't be stored in db4o. See "Storing MarshalByRef Objects" on page 339

db4o includes the assembly-name in the type names. This also means that you need to be careful when accessing the same database from different applications. See "Accessing Persistent Classes From Different .NET Applications" on page 339

There are a few dangerous practices which you really should avoid. See "Dangerous Practices" on page 340

db4o has database size limit, which is by default 2 GByte. See how you can increase it. See "Working With Large Amounts Of Data" on page 341

The Activation Pitfall

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 two main pitfalls that you must be aware about.

Accessing Not Activated Objects

One common pitfall is to access not activate objects. This usually results in null pointer exceptions or invalid values. This happens when you navigate beyond the activated object-graph area. For example, we have a complex relationships and follow them:

```
Person jodie = QueryForJodie(container);

Person julia = jodie.Mother.Mother.Mother.Mother.Mother;

// This will print null
// Because julia is not activated
// and therefore all fields are not set
Console.WriteLine(julia.Name);
// This will throw a NullPointerException.
// Because julia is not activated
// and therefore all fields are not set
string joannaName = julia.Mother.Name;
```

ActivationDepthPitfall.cs: Run into not activated objects

```
Dim jodie As Person = QueryForJodie(container)

Dim julia As Person = jodie.Mother.Mother.Mother.Mother.Mother

' This will print null
' Because julia is not activated
' and therefore all fields are not set
Console.WriteLine(julia.Name)
' This will throw a NullPointerException.
' Because julia is not activated
' and therefore all fields are not set
Dim joannaName As String = julia.Mother.Name
```

ActivationDepthPitfall.vb: Run into not activated objects

This will result in a exception. Because by default db4o only activates object up the a depth of 5. This means that when you load a object, that object and all object which are reachable via 4 references are activated.

There are multiple solutions to this issue.

- Activate the object explicitly as you dive deeper into the object graph.
- Increase the global activation-depth.
- Increase the activation-depth for certain types.
- Use wisely the cascading activation.
- The most elegant solution is transparent activation. With transparent activation db4o takes care of activating object as you access them.

To High Activation Depth Or Two Many Cascade Activation

Having a high activation-depth makes working with db4o much easier. However activation can take a long time with deeper object graphs and become a serious performance bottleneck. The same applies when using cascade activation on almost all types. To reduce the time spend on activating objects, you need to be more selective about what to activate and what not.

- Activate the object explicitly as you dive deeper into the object graph.
- The most elegant solution is transparent activation. With transparent activation db4o takes care of activating object as you access them.

Update Depth Pitfall

db4o update behavior is regulated by Update Depth. Understanding update depth will help you to improve performance and avoid unnecessary memory usage. A common pitfall is that the update-depth is to small and that the objects are not updated. In such cases you either need to explicitly store the related objects individually or increase the update-depth.

For example in this code we add a new friend and store the object. Since a collection is also handled like a regular object, it is affected by the update-depth. In this example we only store the person-object, but not the friend-collection-object. Therefore with the default-update depth of one the update isn't store. In order to update this properly, you either need to set the update depth to two or store the friend-list explicitly.

```
using (IObjectContainer container = Db4oEmbedded.OpenFile(DatabaseFile))
{
    Person jodie = QueryForJodie(container);
    jodie.Add(new Person("Jamie"));
    // Remember that a collection is also a regular object
    // so with the default-update depth of one, only the changes
    // on the person-object are stored, but not the changes on
    // the friend-list.
    container.Store(jodie);
}
using (IObjectContainer container = Db4oEmbedded.OpenFile(DatabaseFile))
{
    Person jodie = QueryForJodie(container);
    foreach (Person person in jodie.Friends)
    {
        // the added friend is gone, because the update-depth is to low
        Console.WriteLine("Friend=" + person.Name);
    }
}
```

UpdateDepthPitfall.cs: Update doesn't work on collection

```
Using container As IObjectContainer = Db4oEmbedded.OpenFile(DatabaseFile)
    Dim jodie As Person = QueryForJodie(container)
    jodie.Add(New Person("Jamie"))
    ' Remember that a collection is also a regular object
    ' so with the default-update depth of one, only the changes
    ' on the person-object are stored, but not the changes on
    ' the friend-list.
    container.Store(jodie)
End Using
Using container As IObjectContainer = Db4oEmbedded.OpenFile(DatabaseFile)
    Dim jodie As Person = QueryForJodie(container)
    For Each person As Person In jodie.Friends
        ' the added friend is gone, because the update-depth is to low
        Console.WriteLine("Friend=" & person.Name)
    Next
End Using
```

UpdateDepthPitfall.vb: Update doesn't work on collection

So for this example setting the update-depth to two will fix the issue. For lots of operation a update-depth of two is pretty reasonable. This allows you to update collections without storing them explicitly.

```
IEmbeddedConfiguration config = Db4oEmbedded.NewConfiguration();
config.Common.UpdateDepth = 2;
using (IObjectContainer container = Db4oEmbedded.OpenFile(DatabaseFile))
{
```

UpdateDepthPitfall.cs: A higher update depth fixes the issue

```
Dim config As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
config.Common.UpdateDepth = 2
Using container As IObjectContainer = Db4oEmbedded.OpenFile(DatabaseFile)
```

UpdateDepthPitfall.vb: A higher update depth fixes the issue

When the 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.

Try transparent persistence, which removes the issue of the update-depth completly.

Storing MarshalByRef Objects

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 storable 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.

2. If using MarshalByRef object is mandatory for you, use the following configuration:

```
configuration.Common()
    .ObjectClass("System.Runtime.Remoting.ServerIdentity, mscorlib")
    .Translate(new TTransient());
configuration.Common()
     .ObjectClass("System.Threading.TimerCallback,mscorlib")
     .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.
Note: in many cases using db4o client-server version can be a better option for a remote persistence.

Accessing Persistent Classes From Different .NET Applications

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.Store(new MyClasses.Pilot("David Barrichello",99))
// internally the class is saved as "MyClasses.Pilot, Application1".
```

Application2.exe ("Application2" assembly):

c#:

```
// create an Alias for the MyClasses.Pilot, Application1:
IEmbeddedConfiguration configuration = Db4oEmbedded.NewConfiguration();
configuration.Common.AddAlias(new TypeAlias("MyClasses.Pilot, Application1", " MyClasses.Pilot,
Application2"));
IObjectContainer container = Db4oEmbedded.OpenFile(configuration, "reference.db4o");
// now you query as usual
IObjectSet result = container.QueryByExample(new Test());
```

VB:

```
// create an Alias for the MyClasses.Pilot, Application1:
Dim configuration As IEmbeddedConfiguration = Db4oEmbedded.NewConfiguration()
configuration.Common.AddAlias(new TypeAlias("MyClasses.Pilot, Application1", " MyClasses.Pilot,
Application2"))
Dim container As IObjectContainer = Db4oEmbedded.OpenFile(configuration, "reference.db4o")
// now you query as usual
Dim result As IObjectSet = container.QueryByExample(New Test())
```

For more information see Class Name Format In .NET and Aliases.

Dangerous Practices

Db4o databases are well protected against corruption. However some specific configurations can make your database file vulnerable.

- Disabling the file-lock. When two db4o instance write to the same database file at the same time, the database will be corrupted. Therefore you should avoid disabling this setting. You can safely disable the file-lock, when you're using the database in a read-only mode.
- Using the non-flushing storage. This storage will disable the flush-operation to the disk. While this improves the performance, it endangers consistent commits in a crash.
- db4o cannot deal with some class-hierarchy-changes. You cannot add a class between two existing classes in the class-hierarchy. Or remove a class from the top of the class-hierarchy. See "Refactoring Class Hierarchy" on page 211
- You cannot change a field from a array to a simple field of the same type and back.This only applies when you change it from a type to the same array-type. So for example from a string to an array of strings. A change from string to an array of integer is fine. See "Field Refactoring Limitation" on page 210

Persistent Hashtables

Hashtable or Hashmap is a data structure that associates keys to values. Hashtable uses a hash function to quickly navigate to a specific value. Hash function returns an integer value based on a specific algorithm whichis based on the contents of the object. Different hash algorithms can be used to produce hash codes for different objects. The general requirements for hash code are the following:

- Hash function must return the same result for the same object during the lifetime of the application.
- Hash function must produce the same results for the objects that are equal according to the equals(object) function
- If 2 objects are unequal according to the equals(object) function it is not required that the hash function produce distinct results

As you can see from the last point there can be more than one distinct key object in a hashtable that have the same hash code. Special methods called collision resolution are used to find the correct value for the specific key. Usually a separate storage - a bucket - is used for all keys with the same hash code. In this case a bucket is located by the hash code and then the right key is searched within the bucket, which allows to get a good enough performance. This works good enough for an in-memory hashtable as the hash values are not changed during application lifetime. However, it gets more difficult with a persistent hashtable.

When a hashtable is stored to a database - the hash values are not stored. As we know from the definition, the hash value of an object is only required to stay the same during the application lifetime, which means that if the hashtable will be loaded into memory from the database in another application or in another session, the hash values of the keys can differ from their initial value. We will still be able to retrieve values by their key objects if equals and hashCode functions are based on the object contents. However the consistency of the hashtable can potentially be broken. This can happen if the key objects from different buckets will obtain the same hash value as the result of re-instantiation from the database.

The simplest way to avoid the inconsistency of the persisted hash table use object content-based hash code functions for your key objects.

Working With 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. The maximum possible size is 254GB.

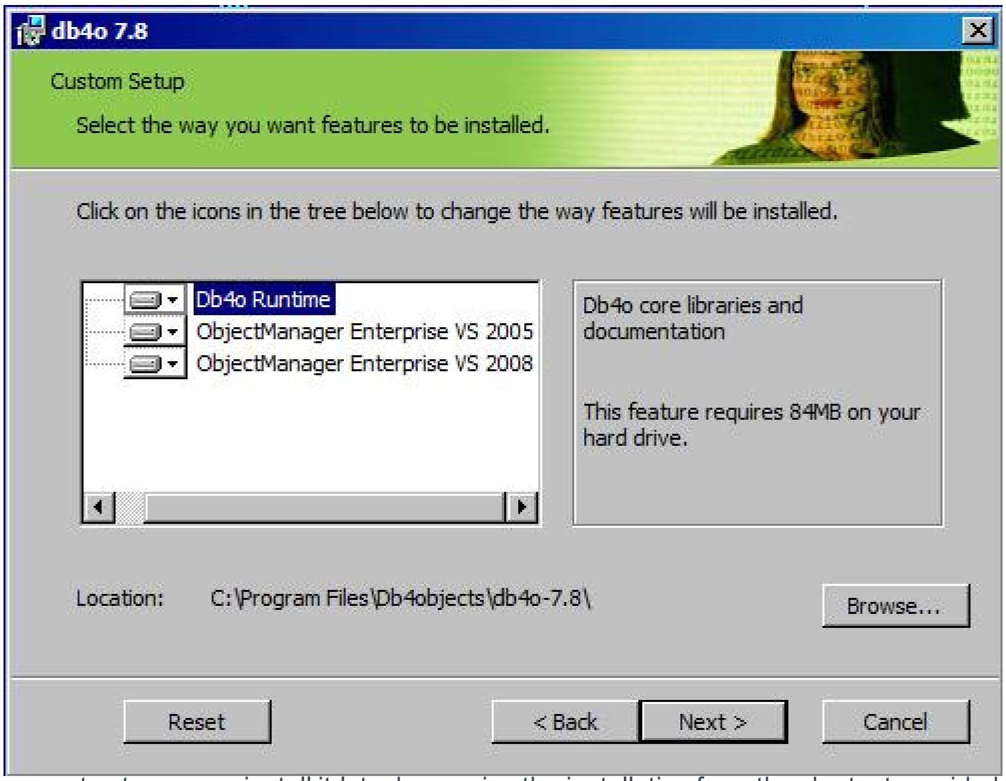
You cannot change this setting for an existing database. In order to change it for an existing database, you need to defragment the database.

db4o Replication System

The db4o Replication System is a separate project. You can download it and its documentation on the official db4o website.

Object Manager Enterprise

Object Manager Enterprise (**OME**[1]) is an object browser for db4o databases. OME provided with this installation comes as Visual Studio plugin. OME can be installed as part of db4o setup.


db4o 7.8

Custom Setup

Select the way you want features to be installed.

Click on the icons in the tree below to change the way features will be installed.

- Db4o Runtime
- ObjectManager Enterprise VS 2005
- ObjectManager Enterprise VS 2008

Db4o core libraries and documentation

This feature requires 84MB on your hard drive.

Location: C:\Program Files\Db4objects\db4o-7.8\

Alternatively, if you opt out, you can install it later by running the installation from the shortcut provided in the db4objects

[1]Object Manager, a tool to view and edit a db4o database

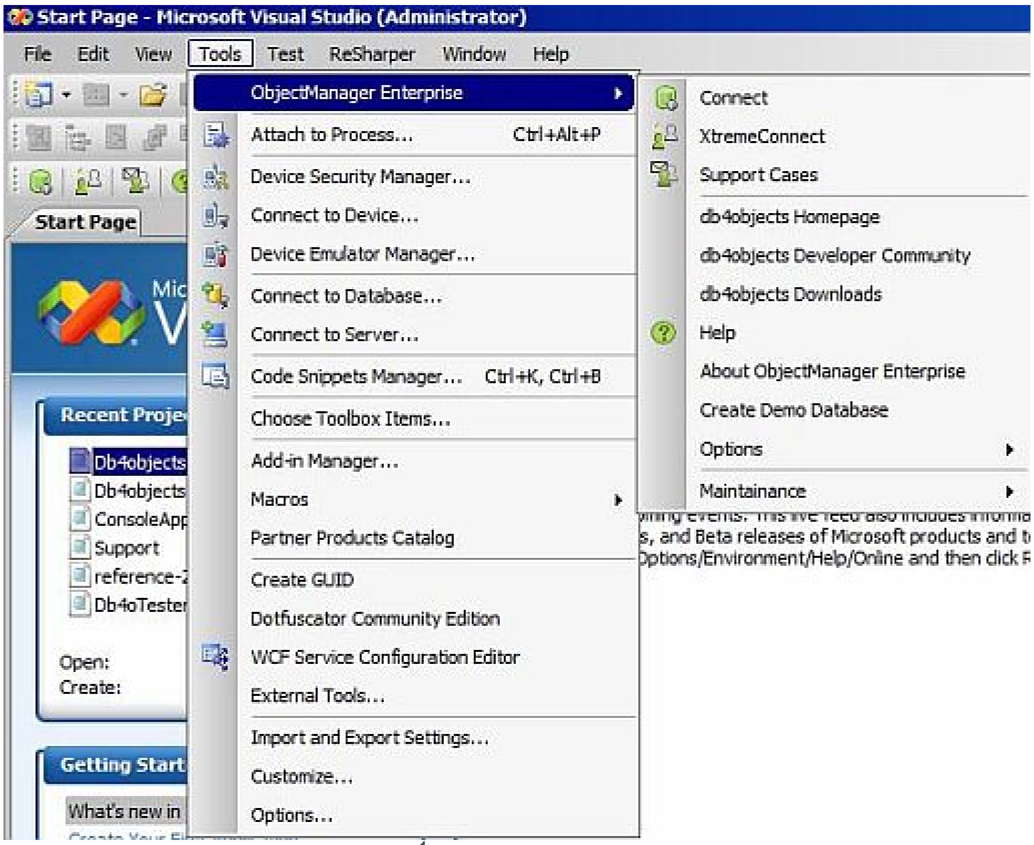
Start menu folder. If you've downloaded .NET distribution as a zip archive, you will find OME installation in the OME-folder of your db4o distribution. The 3.5 distribution includes the OMEfor Visual Studio 2008, the 4.0 distribution the version for Visual Studio 2010.

More Reading:

- OME Interface
- Browsing A Database
- Querying
- Viewing Collections
- Viewing Enums

OME Interface

Once the Object Manager (OM) is installed you can see it in Visual Studio Tools menu:



You should also be able to see **OME**[1] toolbar icons:

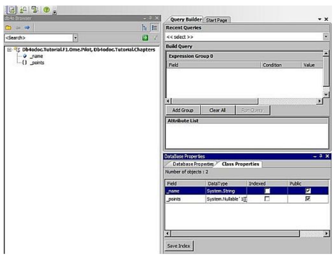
Browsing A Database

Let's look into a simple database containing Pilot and Car objects. If you don't have such a db4o database - please, create one using any of the code in the examples. To open the database select Tools->Object

[1]Object Manager, a tool to view and edit a db4o database

Manager Enterprise->Connect (or use a shortcut button from the toolbar menu) and browse to your database file.

Once you've connected you will see a screen similar to this:

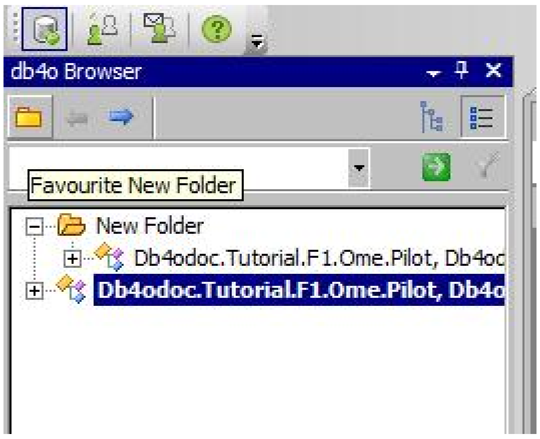


In this view you can see:

- Db4o Browser: window displaying the contents of the open db4o database
- Database Properties: window displaying the properties of the open database or the properties of the selected database class
- Build Query: windows allowing to build a query using drag&drop functionality
- Query Results: window to browse the results of the query execution

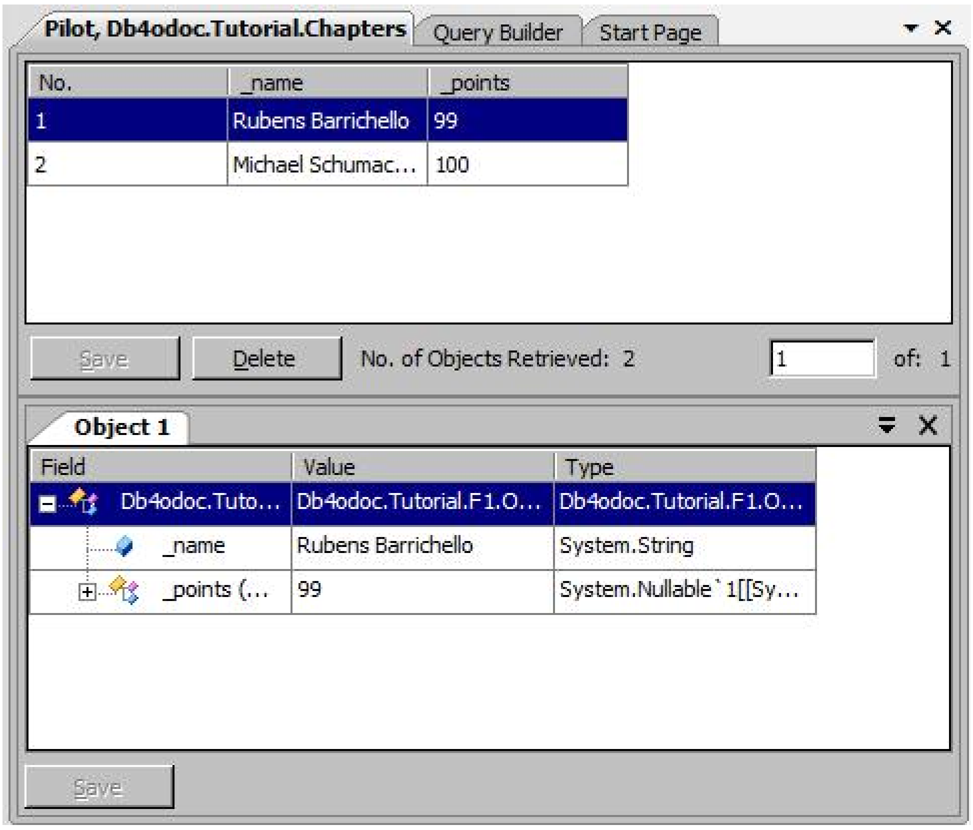
The db4o Browser window shows that there is 1 class in the database (Pilot), which contains 2 fields: name and points. In the Property Viewer you can see more information about the class fields. You can also change "Indexed" field and add the index to the database by pressing "Save Index" button.

The filter panel on the top of the view allows easier navigation through the database with lots of different classes. You can use wildcard searches and benefit from the search history to make the selection faster. To further improve the navigation experience, you can create favourite folders and drag&drop frequently used classes into these folders.


db4o Browser
Favourite New Folder
New Folder
Db4odoc.Tutorial.F1.Ome.Pilot, Db4oc
Db4odoc.Tutorial.F1.Ome.Pilot, Db4o

Querying

It is easy to retrieve all of the Pilot instances from the database: just right-click the Pilot class in db4o Browser and select "Show All Objects". The list of the Pilot objects will be shown in the Query Result view:


Pilot, Db4odoc.Tutorial.Chapters | Query Builder | Start Page

No.	_name	_points
1	Rubens Barrichello	99
2	Michael Schumac...	100

Save | Delete | No. of Objects Retrieved: 2 | 1 | of: 1

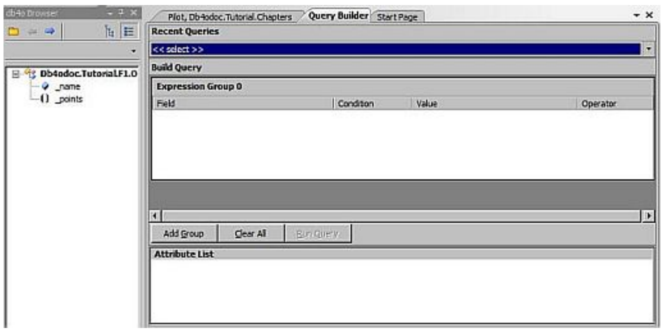
Object 1

Field	Value	Type
Db4odoc.Tuto...	Db4odoc.Tutorial.F1.O...	Db4odoc.Tutorial.F1.O...
_name	Rubens Barrichello	System.String
_points (...	99	System.Nullable`1[[Sy...

Save

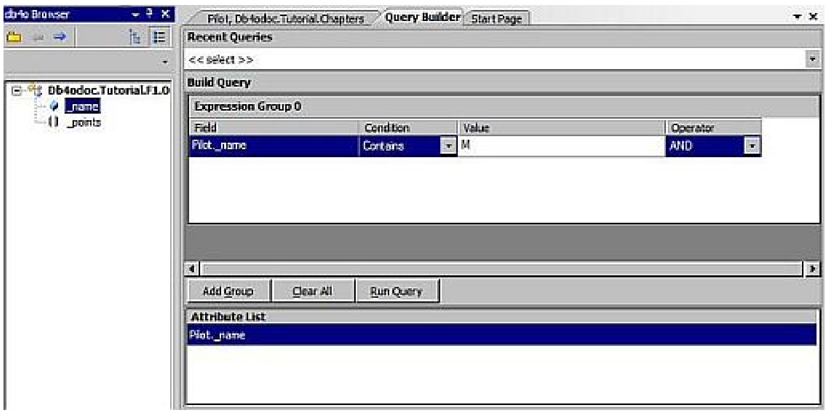
You can see object details in the detailed view below. Try to change any values and use Save button to persist the changes to the database. You can also use Delete button to delete objects from the database. For the objects containing field objects you will be prompted to use cascade on delete.

More complex queries can be done by using Query Builder view:



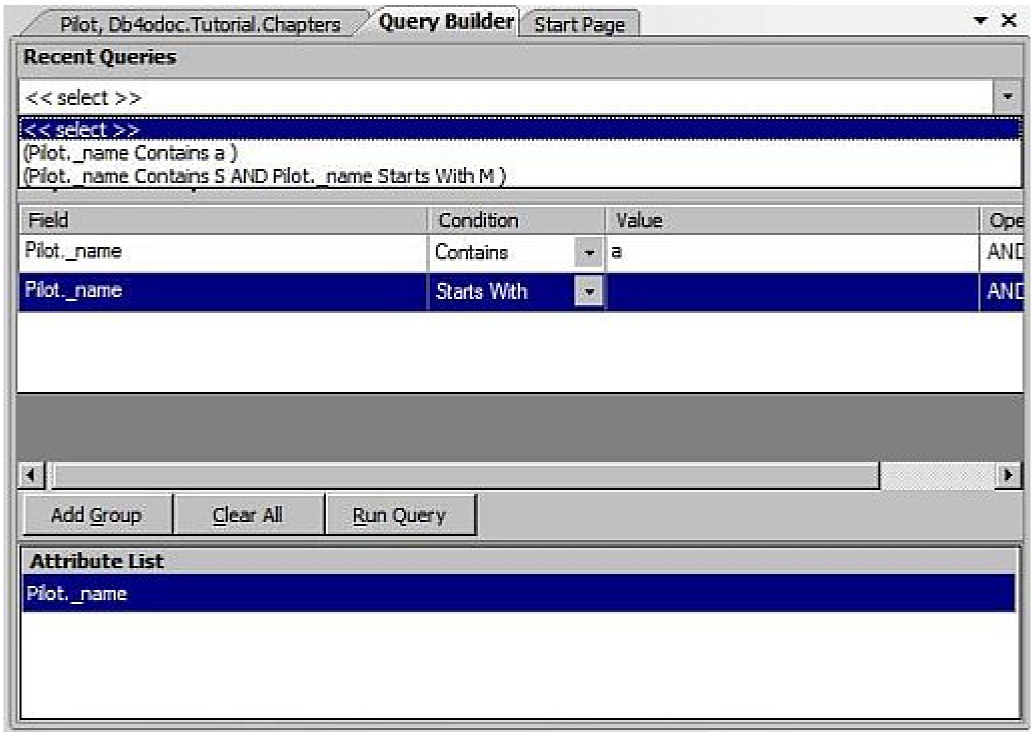
Drag "_name" field from the Db4o Browser view into the Query Builder view, set condition "Contains", put a value "a" and run the query. You can return to the Built Query tab and modify the query later on again. For example: add "AND" operator, drag "_name" field, set Condition to "Starts With" and the value to "M". Re-run the query.

If you want only selected fields to be displayed in the query result, drag and drop fields to be displayed from Db4o Browser into "Attribute List" window.



Field	Condition	Value	Operator
Pilot._name	Contains	M	AND

When the new query is created, the previous query is stored and can be selected from the history drop-down:



Field	Condition	Value	Ope
Pilot._name	Contains	a	AND
Pilot._name	Starts With		AND

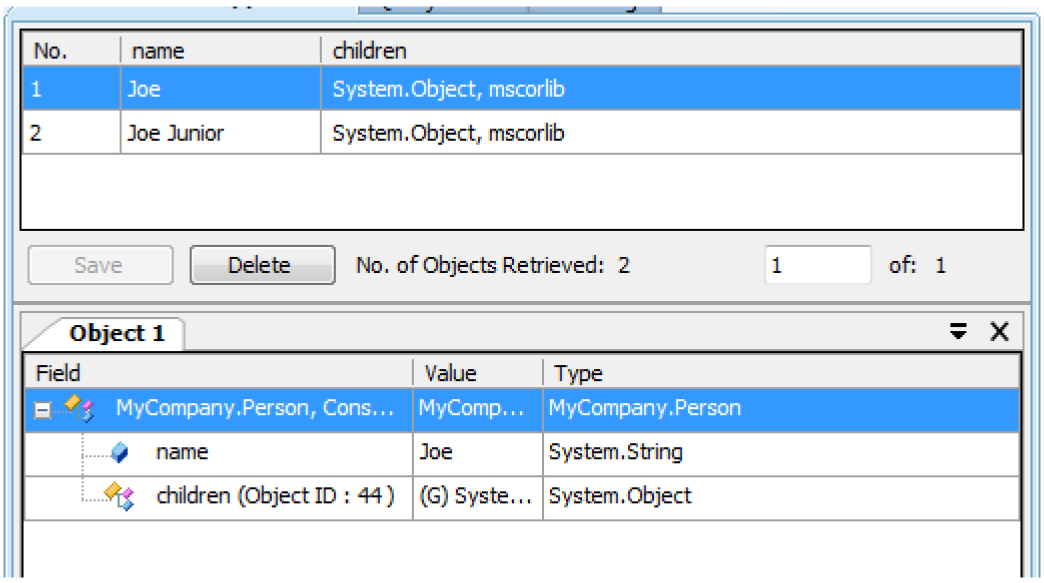
More sophisticated queries can be build by joining grouped constraints using "Add Group" button.

When you are done working with the database in **OME**[1], you can close the connection by using Tools->Object Manager Enterprise-> Disconnect menu command or by using the equivalent button on the toolbar.

Viewing Collections

When you try to view a object which contains a collection in **OME**[2], you cannot descend into the collection-items. Instead you see only the collection-type.

[1]Object Manager, a tool to view and edit a db4o database
[2]Object Manager, a tool to view and edit a db4o database



No.	name	children
1	Joe	System.Object, mscorlib
2	Joe Junior	System.Object, mscorlib

Save Delete No. of Objects Retrieved: 2 1 of: 1

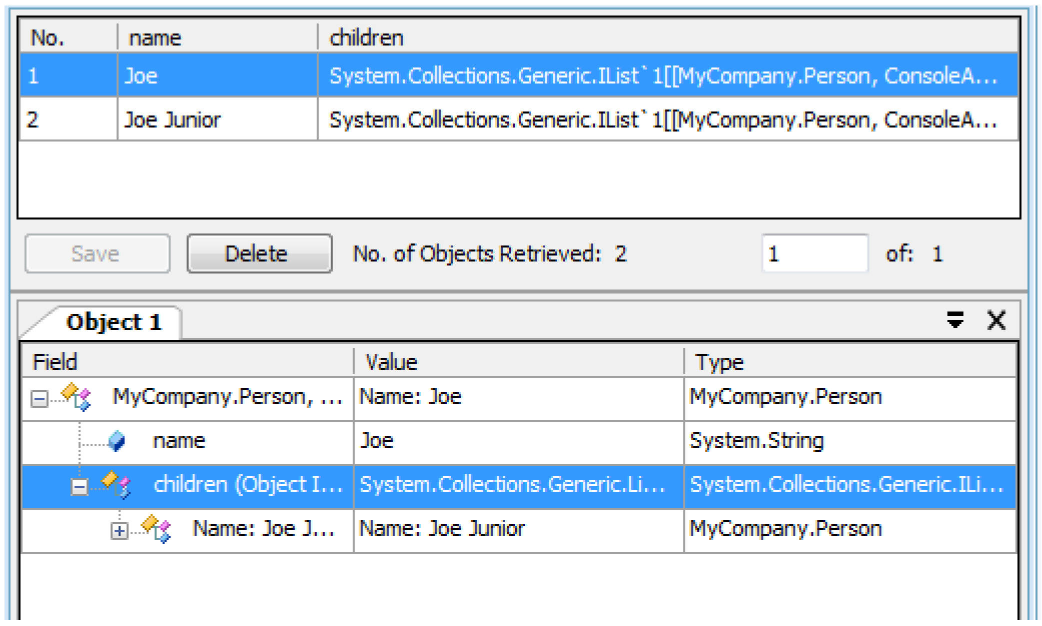
Object 1

Field	Value	Type
MyCompany.Person, Cons...	MyComp...	MyCompany.Person
name	Joe	System.String
children (Object ID : 44)	(G) Syste...	System.Object

To descend into the collection-members, OME needs the original classes. Therefore you need to specify where the assembly for this class is.

Go to "Tools" -> "Object Manager Enterprise" -> "Options" menu and select "Assembly Search Path...". Add the path to the assembly, containing the type definition. Now disconnect and reconnect the database and browse the objects again:

After that, the collections-members a visible.



No.	name	children
1	Joe	System.Collections.Generic.IList`1[[MyCompany.Person, ConsoleA...
2	Joe Junior	System.Collections.Generic.IList`1[[MyCompany.Person, ConsoleA...

Save Delete No. of Objects Retrieved: 2 1 of: 1

Object 1

Field	Value	Type
MyCompany.Person, ...	Name: Joe	MyCompany.Person
name	Joe	System.String
children (Object I...	System.Collections.Generic.Li...	System.Collections.Generic.ILi...
Name: Joe J...	Name: Joe Junior	MyCompany.Person

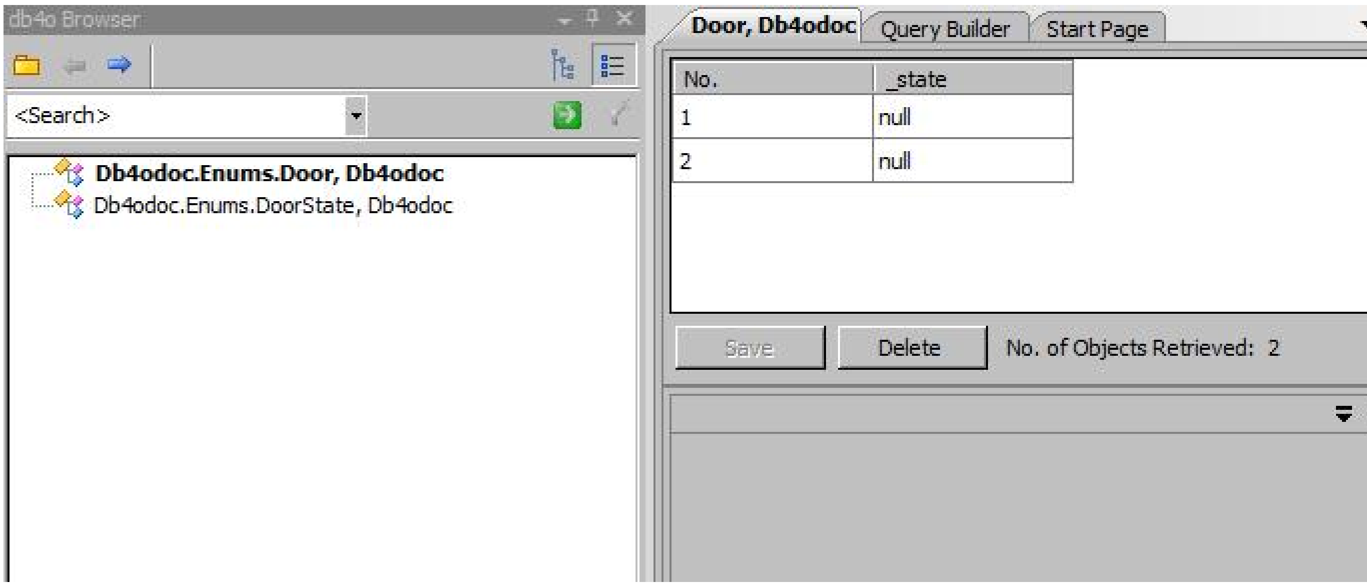
Viewing Enum Values In OME

.NET-enums do not carry any database identity and are stored in their parent object slot. If you open a database with enum values in **OME**[1], you will be able to see the enum class in the db4o Browser view,

[1]Object Manager, a tool to view and edit a db4o database

but if you proceed and select "Show all objects" from the right-click menu you will see that there are no enum objects stored in the database.

But don't worry, an enum is just an integer so all we need are the values of the enum fields in the database. Please, select a class with the enum field in the db4o Browser view. Right-click it and select "Show all objects".


db4o Browser
<Search>
Db4odoc.Enums.Door, Db4odoc
Db4odoc.Enums.DoorState, Db4odoc
Door, Db4odoc
Query Builder
Start Page
Save
Delete
No. of Objects Retrieved: 2

No.	_state
1	null
2	null

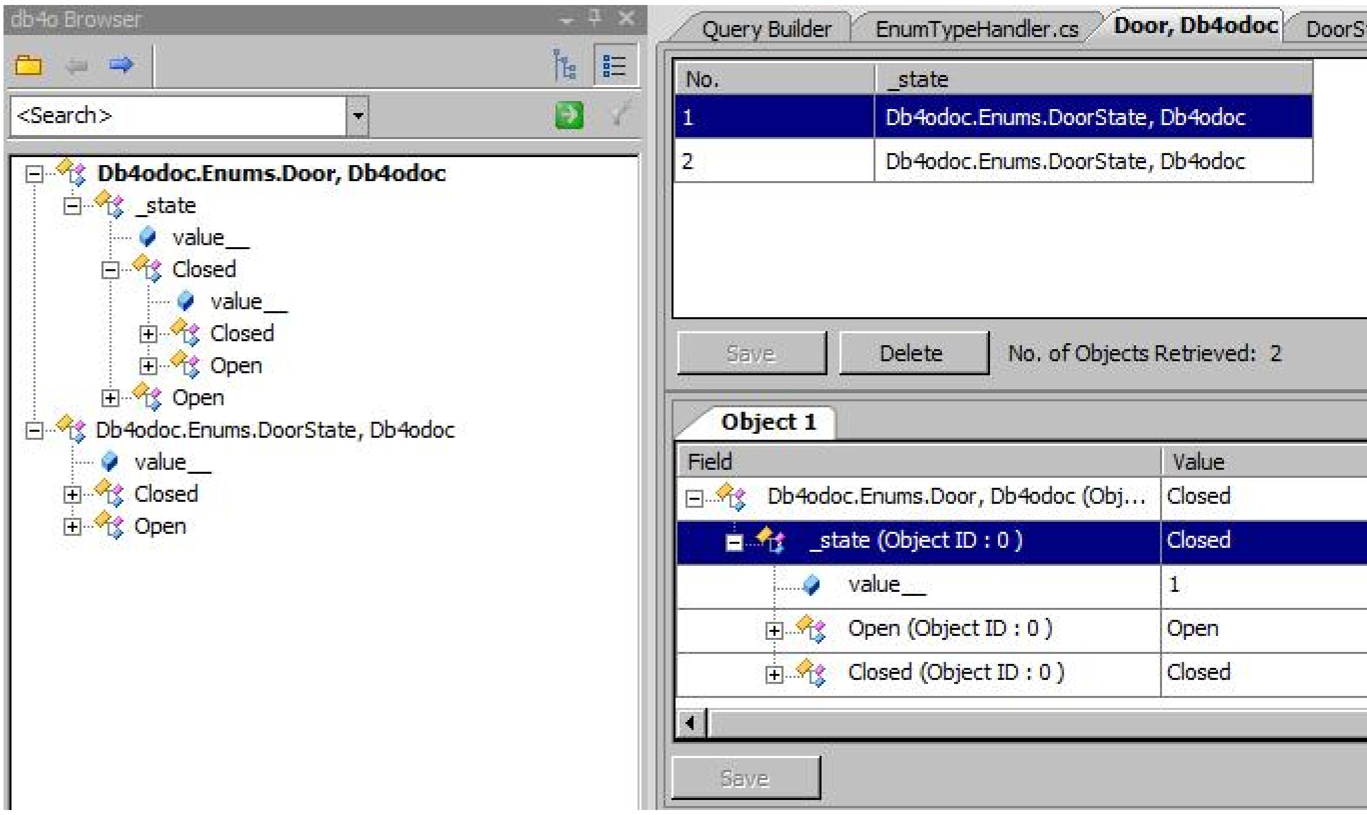
Something went wrong here - there are no Enum values and nulls are stored instead. Is the information lost?

Here you will need to understand how enum values are stored in the database. In order to save space in the database file, db4o stores only the absolutely necessary information for the enums:

- Enum class id
- Enum numeric value

The actual enum representation will be picked up in the runtime using enum class definition. As OME by default runs without class definitions using Generic Reflector, OME can't interpret the Enum field value correctly. In order to fix you need to add the assemblies which contain the enum-type.

Go to "Tools" -> "Object Manager Enterprise" -> "Options" menu and select "Assembly Search Path...". Add the path to the assembly, containing the type definition. Now disconnect and reconnect the database and browse the objects again:



```
Db4odoc.Enums.Door, Db4odoc
    _state
        value__
        Closed
            value__
            Closed
            Open
        Open
Db4odoc.Enums.DoorState, Db4odoc
    value__
    Closed
    Open
```

No.	_state
1	Db4odoc.Enums.DoorState, Db4odoc
2	Db4odoc.Enums.DoorState, Db4odoc

No. of Objects Retrieved: 2

Field	Value
Db4odoc.Enums.Door, Db4odoc (Obj...	Closed
_state (Object ID : 0)	Closed
value__	1
Open (Object ID : 0)	Open
Closed (Object ID : 0)	Closed

Community

Participate in the db4o community and help to improve db4o. The main community website for db4o is http://developer.db4o.com. There you can find the db4o forums, blog posts additional resources and the newest db4o releases.

db4o Forums

The best place to ask questions about db4o, discuss and make suggestions are the db4o forums. You can find the db4o forums here:http://developer.db4o.com/Forums.aspx

Additional Resources on the Web

There's a overview of additional resources here: http://developer.db4o.com/Resources.aspx To stay informed about new features, community activity and other news subscribe to the different db4o blogs: http://developer.db4o.com/Blogs.aspx.

Or follow us on Twitter: http://twitter.com/db4objects

Report Bugs

Every software contains bugs and so does db4o. If you find bugs please report them to us. In case you're not sure if you really found a bug, you maybe want to discuss your issue first on the db4o forum. As soon as you're sure that you found a bug it best to report in our bug-tracking system. The login is the same as in the db4o forums. Read more on tips and tricks for reporting bugs here.

When possible include a small test program which reproduces the issue with the bug report. A test case avoids confusion and effort to reproduce the bug.

Working with the Source Code

See "Working With Source Code" on page 352

Report Bugs

Find a bug in db4o? Here are a few tips and tricks for reporting bugs.

Where Do I Report Bugs?

The best way to report bugs is to directly report the bug in our issue tracking system here. The login for the bug tracker is the same as in the db4o community page and db4o forums. If you don't have a login yet you can register here.

In cases where you are not sure if the issue is an bug or another problem you maybe want to first discuss it in the db4o forums.

What Should The Bug Report Contain?

The bug report should be as specific as possible, so that the room for interpretation is very small. Those items are important:

- Steps to reproduce the bug. In order to fix the bug a step by step instructions on how to reproduce the issue are essential.
- Include the db4o version in your bug report. To see the version you can right click on the db4o assembly, go to 'Properties', then to the tab 'Details' and read the version there in the 'Product Version' section.
- Include information about your platform. Which .NET version and what kind of device (for example a embedded system)
- In case of exceptions, please add also the stack trace.

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.

More Reading:

- Unit Tests for db4o
- Sharpen: Converting Java to C#
- Using The Sources From db4o Distribution
- Using The Repository
- Building Full Distribution
- Building Java Version
- Building .NET Version
- Patch Submission

Sharpen

Sharpen[1] is an tool that allows you to convert your Java db4o project into C#. The difference between sharpen and other java-to-C# converters is the native support for db4o, .NET naming conventions and customization options.

More Reading:

- How To Setup Sharpen
- Doing a Sharpen Conversion
- Sharpen Command-Line Arguments
- Sharpen Annotations

How to Setup Sharpen

You can obtain sharpen source from db4o svn repository at:

https://source.db4o.com/db4o/trunk/sharpen

For the ease of use check-out sharpen projects:

- sharpen.builder
- sharpen.core
- sharpen.ui
- sharpen.ui.tests

Additionally **Sharpen**[2] requires a valid Eclipse installation to run. Install Eclipse on your machine or reuse an existing installation. Check out the four projects into the same workspace as your java project which you want to convert. This is not required is easier to use and maintain.

The first step is to build sharpen. For that we can use Apache Ant:

---

[1]Sharpen is a tool to translate Java source code to C# source code
[2]Sharpen is a tool to translate Java source code to C# source code

```
<target name="build-sharpen">
    <property name="sharpen.core.dir" location="${dir.workspace}/sharpen.core"/>
    <reset-dir dir="${dir.dist.classes.sharp}"/>

    <javac fork="true"
           debug="true"
           target="1.5"
           source="1.5"
           destdir="${dir.dist.classes.sharp}"
           srcdir="${sharpen.core.dir}/src"
           encoding="UTF-8">
        <classpath>
            <fileset dir="${eclipse.home}/plugins">
                <include name="org.eclipse.osgi_*.jar"/>
                <include name="org.eclipse.core.resources_*.jar"/>
                <include name="org.eclipse.core.runtime_*.jar"/>
                <include name="org.eclipse.jdt.core_*.jar"/>
                <include name="org.eclipse.jdt.launching_*.jar"/>
                <include name="org.eclipse.equinox.*.jar"/>
                <include name="org.eclipse.core.jobs_*.jar"/>
            </fileset>
        </classpath>
    </javac>
    <jar destfile="${dir.dist.classes.sharp}/sharpen.core_1.0.0.jar" basedir="${dir.dist.classes.sharp}">
        <fileset dir="${sharpen.core.dir}">
            <include name="plugin.xml"/>
        </fileset>
    </jar>
</target>
```

sharpen-install.xml: Build Sharpen

To run Sharpen you should install it to an Eclipse instance:

```
<target name="install-sharpen-plugin" depends="build-sharpen">
    <copyfile src="${dir.dist.classes.sharp}/sharpen.core_1.0.0.jar" dest="${plugins.home}/sharpen.core_1.0.0.jar
</target>
```

sharpen-install.xml: Install Sharpen to Eclipse

Put the paths for the build in a property file, so that you can easily change them. Here's a example of the property file. You have to configure the JDK-path, the Eclipse path and the path to the Sharpen source.

```
#The workspace where the sharpen projects are
dir.workspace=C:/Users/Gamlor/Develop/db4o/sharpenProject
# Java executable
jdk.home=${env.JAVA_HOME}
jdk.home.java=${jdk.home}/bin/java.exe
# Eclipse home directory
eclipse.home=C:/progs/eclipse
# Sandcastle can be used to convert javadoc to .NET xml comments
# dir.lib.sandcastle=e:/sandcastle/
# sharpen compile directory
dir.dist.classes.sharp=${dir.workspace}/dist/
# Eclipse plugins home
plugins.home=${eclipse.home}/plugins
```

sharpen.properties: The configuration for building sharpen

After that Sharpen is set up to run.

Doing a Sharpen Conversion

Ensure that you've installed sharpen to an existing eclipse installation as explained here.

Use Ant scripts to run **Sharpen**[1] and translate your Java code to C#. The best way for this is to define an Ant macro which you then can reuse. This task takes two arguments. The first argument is the path to a valid Eclipse workspace which contains the project to translate. The second parameter is the project in the workspace which you want to translate.

```
<macrodef name="sharpen">
    <attribute name="workspace"/>
    <attribute name="resource"/>

    <element name="args" optional="yes"/>

    <sequential>
        <java taskname="sharpen"
              fork="true"
              classname="org.eclipse.core.launcher.Main"
              failonerror="true" timeout="1800000">

            <classpath>
                <fileset dir="${eclipse.home}/plugins">
                    <include name="org.eclipse.equinox.launcher_*.jar"/>
                </fileset>
            </classpath>

            <arg value="-clean"/>
            <arg value="-data"/>
            <arg file="@{workspace}"/>
            <arg value="-application"/>
            <arg value="sharpen.core.application"/>
            <arg value="@{resource}"/>
            <args/>
        </java>
    </sequential>
</macrodef>
```

sharpen-install.xml: The sharpen task

Now you can use this task to sharpen your project. First ensure that your project is in a valid Eclipse workspace. Then you specify the workspace and the sources of the project:

```
<target name="sharpen">
    <sharpen
        workspace="C:\temp\sharpenExamples\"
        resource="example/src">
        <args>
            <arg value="@sharpen-config"/>
        </args>
        </sharpen>
</target>
```

sharpen-example.xml: Sharpen a example project

Additionally you can pass the sharpen configuration as a file-name. When you add a '@' in front of the file-name sharpen will read that file and use all configuration flags of that. For example:

[1] Sharpen is a tool to translate Java source code to C# source code

```
-pascalCase+
-nativeTypeSystem
-nativeInterfaces
```

You can find a list of all Sharpen configuration flags here and a list of all Sharpen annotations here.

Sharpen Command-Line Arguments

Sharpen[1] command-line arguments can be defined in an options file

```
<sharpen workspace="${target.dir}" resource="sharpened_examples/src">
   <args>
      <!-- Sharpen options are defined in a separate file -->
      <arg value="@sharpen-all-options" />
    </args>
 </sharpen>
```

Here sharpen-all-options file contains all command-line options needed to convert current project. For an example of command-line options file see the previous topic.

Command-line arguments can also be specified directly in an ant script:

```
<sharpen workspace="${dir.sharpen}" resource="db4oj/core/src">
    <args>
        <arg value="-xmldoc"/>
        <arg file="config/sharpen/ApiOverlay.xml" />
        <arg value="@sharpen-all-options" />
     </args>
 </sharpen>
```

The following table shows available command-line options, their meaning and example usage:

Argument	Usage
-pascalCase	Convert Java identifiers to Pascal case
-pascalCase+	Convert Java indentifiers and package names (namespaces) to Pascal case
-cp	Adds a new entry to classpath: <arg value="-cp" /> <arg path="lib/db4o-7.2.37.10417-java5.jar" />
-srcFolder	Adds a new source folder for sharpening
-nativeTypeSystem	Map java classes to .NET classes with a similar functionality. For example: java.lang.Class - System.Type
-nativeInterfaces	Adds an "I" in front of the interface name
-organizeUsings	Adds "using" for the types used
-fullyQualify	Converts to a fully-qualified name: -fullyQualify File
-namespaceMapping	Maps a java package name to a .NET namespace. For example: -namespaceMapping com.db4o Db4objects.Db4o
-methodMapping	Maps a java method name to a .NET method (can be method in another class). For example: - methodMapping java.util.Date.getTime Sharpen.Runtime.ToJavaMilliseconds
-typeMapping	Maps a java class to .NET type:

[1]Sharpen is a tool to translate Java source code to C# source code

	-typeMapping com.db4o.Db4o Db4objects.Db4o.Db4oFactory
-propertyMapping	Maps a java method to .NET property: -propertyMapping com.db4odoc.structured.Car.getPilot Pilot
-runtimeTypeName	Name of the runtime class. The runtime class provides implementation for methods that don't have a direct mapping or that are simpler to map at the language level than at the sharpen level. For instance: String.substring, String.valueOf, Exception.printStackTrace, etc. For a complete list of all the method that can be mapped to the runtime class see Configuration#runtimeMethod call hierarchy.
-header	Header comment to be added to all converted files. -header config/copyright_comment.txt
-xmldoc	Specifies an xml-overlay file, which overrides javadoc documentation for specific classes: -xmldoc config/sharpen/ApiOverlay.xml
-eventMapping	Converts the methods to an event.
-eventAddMapping	Marks the method as an event subscription method. Invocations to the method in the form <target>.method(<argument>) will be replaced by the c# event subscription idiom: <target> += <argument>
-conditionalCompilation	Add a condition when to translate the Java code - - conditionalCompilation com.db4o.db4ounit.common.cs !SILVERLIGHT
-configurationClass	Change the configuration class. The default is 'sharpen.core.DefaultConfiguration'

Sharpen Annotations

Sharpen[1] annotations decorate java source code and are used to notify sharpener about how the code should be processed and converted. Annotations can be used to specify how a code element should be converted (for example class to enum), to skip conversion of some code elements, to rename classes, to change visibility etc.

The following table shows existing annotations, their meaning and examples.

Annotation	Meaning
@sharpen.enum	Mark java class to be processed as a .NET enum
@sharpen.rename	Specifies a different name for the converted type, takes a single name argument. For example: @sharpen.rename Db4oFactory

@sharpen.private	Specifies that the element must be declared private in the converted file, though it can be not private in the java source:
	`/*`
	`* @sharpen.private`
	`*/`
	public List4 _first;

@sharpen.internal	Specifies that the element must be declared internal in the con-

[1]Sharpen is a tool to translate Java source code to C# source code

	verted file: `/**` `* @sharpen.internal` `*/` public abstract int size();
@sharpen.protected	Specifies that the element must be declared protected in the converted file: `/**` `* @sharpen.protected` `*/` public abstract int size();
@sharpen.new	Adds the C#-'new' modifier to the translated code.
@sharpen.event	Links an event to its arguments. For example: Java: `/**` `* @sharpen.event com.db4o.events.QueryEventArgs` `*/` `public Event4 queryStarted();` is converted to: `public delegate void QueryEventHandler(` `object sender,` `Db4objects.Db4o.Events.QueryEventArgs args);` `.......` `event Db4objects.Db4o.Events.QueryEventHandler QueryStarted;`
@sharpen.event.add	Marks the method as an event subscription method. Invocations to the method in the form <target>.method(<argument>) will be replaced by the c# event subscription idiom: <target> += <argument>
@sharpen.event.onAdd	Valid for event declaration only (SHARPEN_EVENT). Configures the method to be invoked whenever a new event handler is subscribed to the event.
@sharpen.if	Add #if <expression>#endif declaration: @sharpen.if <expression>
@sharpen.property	Convert a java method as a property: `/**` `* @sharpen.property` `*/` public abstract int size();
@sharpen.indexer	Marks an element as an indexer property
@sharpen.ignore	Skip the element while converting
@sharpen.ignore.extends	Ignore the extends clause in Java class definition
@sharpen.ignore.implements	Ignore the implements clause in Java class definition
@sharpen.extends	Adds an extends clause to the converted class definition. For example: Java: `/**` `* @sharpen.extends System.Collections.IList` `*/` `public interface ObjectSet {...`

	converts to `public interface IObjectSet : System.Collections.IList`
@sharpen.partial	Marks the converted class as partial
@sharpen.remove	Marks a method invocation that should be removed

@sharpen.remove.first	Removes the first line of the method/constructor when converting to C#:
	`/**`
	`* @sharpen.remove.first`
	`*/`
	`public void doSomething(){`
	`    System.out.println("Java");`
	`    NextMethod();`
	`}`
	converts to:
	`public void DoSomething(){`
	`    NextMethod();`
	`}`

@sharpen.struct	Marks class to be converted as c# struct

@sharpen.unwrap	When a method is marked with this annotation all method calls are removed. This is useful for removing conversion methods when their aren't required in C#.
	`/*`
	`* @sharpen.unwrap`
	`*/`
	`public Iterable toIterable(Object[] array){`
	`   return Arrays.asList(array);`
	`}`
	`public void doSomething(Object[] objs){`
	`  Iterable iterable = toIterable(objs);`
	`  // do something with the iterable`
	`}`
	Is converted to:
	`public IEnumerable ToIterable(object[] array){`
	`    return Arrays.AsList(array);`
	`}`
	`public void doSomething(object[] objs){`
	`   Iterable iterable = objs;`
	`   // do something with the iterable`
	`}`

@sharpen.attribute	Adds an attribute to the converted code:
	`/*`
	`* @sharpen.attribute TheAttribute`
	`*/`
	`public void doSomething(){}`
	Will be converted to:
	`[TheAttribute]`
	`public void DoSomething(){}`

@sharpen.macro	Add a replace-pattern macro to your code.

Db4o Testing Framework

db4ounit is a minimal xUnit (jUnit, NUnit) style testing framework. The 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-X.XX-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:

- Creating A Sample Test
- Db4ounit Methods

Creating A Sample Test

Let's create the a extremely simple test case. The first step is to setup db4ounit. The best way is to use db4ounit directly from the source. db4ounit is in the db4o source code folder of the distribution.

Open the db4o source-code of the db4o test-projects in Visual Studio. The db4ounit projects are called 'Db4oUnit' and 'Db4oUnit.Extensions'. Then create a new project which references the db4ounit projects.

After that we're ready to write our first db4ounit test. Create a new class which inherits from AbstractDb4oTestCase. Then add a test-method. Any method which starts with the prefix 'test' is a test method. Then add a main method which starts the test.

```
public class ExampleTestCase : AbstractDb4oTestCase
{
    public static void Main(string[] args)
    {
        new ExampleTestCase().RunSolo();
    }

    public void TestStoresElement()
    {
        Db().Store(new TestItem());
        IList<TestItem> result = Db().Query<TestItem>();
        Assert.AreEqual(1, result.Count);
    }


    private class TestItem
    {
    }
}
```

ExampleTestCase.cs: Basic test case

```
Public Class ExampleTestCase
    Inherits AbstractDb4oTestCase
    Public Shared Sub Main(ByVal args As String())
        Dim testCase = New ExampleTestCase()
        testCase.RunSolo()
    End Sub

    Public Sub TestStoresElement()
        Db().Store(New TestItem())
        Dim result As IList(Of TestItem) = Db().Query(Of TestItem)()
        Assert.AreEqual(1, result.Count)
    End Sub


    Private Class TestItem

    End Class
End Class
```

ExampleTestCase.vb: Basic test case

Db4ounit Methods

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. It will setup a db4o instance for you which you can use in your tests. Additionally it provides different utility methods for configuring, querying and modifying the database.

Additionally it contains methods to run the test. You can create new instance of a class which extends AbstractDb4oTestCase and run the test in different environments. For example the method 'runSolo' will run the test with an embedded local container.

ITestCase

This interface provides the basic for a unit test in the db4ounit framework. When you implement this interface the class is a valid unit test. All methods starting with 'test' will be executed as test.

For most tests it is more continent to use the AbstractDb4oTestCase.

ITestLifeCycle

This interface provides a test case with additional setup and tear down methods. Those will be called before and after each test method

ConsoleTestRunner

A test-runner which runs the tests as console application.

Usually it's more convenient to extend the AbstractDb4oTestCase class and use the provided run methods instead of the console test runner.

Db4ounit.Assert

This class provides a variety of methods for asserting certain conditions

Using The Sources From db4o Distribution

The sources for db4o are in the 'src' subdirectory of the db4o distribution. In that directory is a selection of solution-files appropriate for different versions of db4o. Open the appropriate solution file to view, edit and compile the sources.

Note that big parts of the .NET source-code are generated from the Java source code. The translated code is harder to read and not intended to be changed.

Following projects are shipped with db4o:

- Db4objects.Db4o: The db4o core.
- Db4objects.Db4o.CFCompatibilityTests: Special tests for the compact framework.
- Db4objects.Db4o.CS: The db4o client server mode.
- Db4objects.Db4o.CS.Optional: Optional features for the client server mode.
- Db4objects.Db4o.Data.Services: db4o integration into the WCF data services.
- Db4objects.Db4o.Data.Services.Tests: Test suite for the WCF data services integration.
- Db4objects.Db4o.Instrumentation: The instrumentation code used for code enhancer and native queries.
- Db4objects.Db4o.Linq: The db4o LINQ provider.
- Db4objects.Db4o.Linq.Tests, Db4objects.Db4o.Linq.VB.Tests, Db4objects.Db4o.Linq.Instrumentation.Tests: The test suite for the LINQ provider.
- Db4objects.Db4o.NativeQueries: The native query implementation.
- Db4objects.Db4o.Optional: Optional features for db4o.
- Db4objects.Db4o.Silverlight.TestStart, Db4objects.Db4o.SilverlightTestHost: Part of the Silverlight test suite.
- Db4objects.Db4o.Tests: The db4o test suite.
- Db4oTool: The db4o tool source code.
- Db4oTutorial: The db4o tutorial.
- Db4oUnit: The db4o unit test framework.
- Db4oUnit.Extensions: Utilities and extensions for the db4o unit framework.

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.

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.

- bloat: The byte code manipulation library db4o uses.
- db4o.cs: The client server implementation for db4o.
- db4o.cs.optional: Optional client server features for db4o.
- db4o.instrumentation: The instrumentation basic functionality for db4o.

- db4o.net: The .NET version of db4o.
- db4obuild: The build scripts for db4o.
- db4onqopt: The db4o native query optimizer.
- db4otaj: The db4o transparent activation/persistence enhancer.
- db4otools: The db4o tools like Ant scripts.
- db4ounit: The db4o test project.
- db4ounit.extensions: Additional unit test functionality.
- decaf: The Java 1.5 to Java 1.4 converter. You need to checkout all sub-projects on the same level as other db4o projects.
- drs: The db4o replication system.
- sharpen: The Java to C# converter. You need to checkout all sub-projects on the same level as other db4o projects.

Building db4o

Building full distribution will allow you to get the same db4o packages as you can get from db4o download center. However, the flexibility of the build project also allows you to get only parts of it, like only java distro, only documentation, only tests etc.

The following documentation explains how to build a full distribution using Eclipse version 3.4 Ganymede. It is assumed that you have ant and one of Eclipse SVN clients (Subclipse or Subversive) installed.

Projects Required

In order to build db4o you will need to check out the following projects.

- bloat: The byte code manipulation library db4o uses.
- db4o.cs: The client server implementation for db4o.
- db4o.cs.optional: Optional client server features for db4o.
- db4o.instrumentation: The instrumentation basic functionality for db4o.
- db4o.net: The .NET version of db4o.
- db4obuild: The build scripts for db4o.
- db4onqopt: The db4o native query optimizer.
- db4otaj: The db4o transparent activation/persistence enhancer.
- db4otools: The db4o tools like Ant scripts.
- db4ounit: The db4o test project.
- db4ounit.extensions: Additional unit test functionality.
- decaf: The Java 1.5 to Java 1.4 converter. You need to checkout all sub-projects on the same level as other db4o projects.
- drs: The db4o replication system.
- sharpen: The Java to C# converter. You need to checkout all sub-projects on the same level as other db4o projects.
- doctor: A tool for building the tutorial.
- tutorial: The db4o tutorial

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.

Read the instructions at top of the build.xml to find out what options are available. Here's an example:

```
file.compiler.jdk1.3=%JAVA_HOME%/bin/javac.exe
file.compiler.jdk1.3.args.optional=-source 1.3
file.jvm.jdk1.5=%JAVA_HOME%/bin/java.exe
dir.workspace=C:/Users/Gamlor/Develop/db4o/db4o-src/
eclipse.home=C:/progs/eclipse
msbuild.executable="C:/Windows/Microsoft.NET/Framework/v4.0.30319/MSBuild.exe"
```

Build Preparation

First you will need to run some preparation scripts. This is done only once per workspace and should not be repeated in the future.

Run build-db4obuild.xml, this will compile some of the tools used in the build process.

You will need to generate a key to sign the tutorial applet. Use the following commands:

keytool -genkey -alias db4objects -keyalg rsa

keytool -export -alias db4objects -file [path]/db4obuild/config/db4objects.crt

Use "kistoa" (without quotes) as your keypass and storepass.

Replace [path] with the path to db4obuild project on your system and make sure that db4objects.crt file is created in db4obuild/config folder.

If you've already generated db4objects key pair before, you will need to delete it before re-generating:

keytool -delete -alias db4objects

You will need to add ant-contrib.jar to your eclipse ant. You can download ant-contib.jar at:

http://sourceforge.net/projects/ant-contrib

- Add ant-contib jar to ant folder in eclipse/plugins.
- After this is done go to Window->Preferences menu in Eclipse.
- Select Ant->Runtime in the list.
- Then select "Ant Home Entries".
- Press "Add External Jar" and select ant-contib.jar location in the plugins folder.

Running The Build

Now everything is ready to run db4o build. Right-click build.xml file and select "Run As/Ant Build". You will need to run "buildall" target to generate java and .NET distribution.

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:

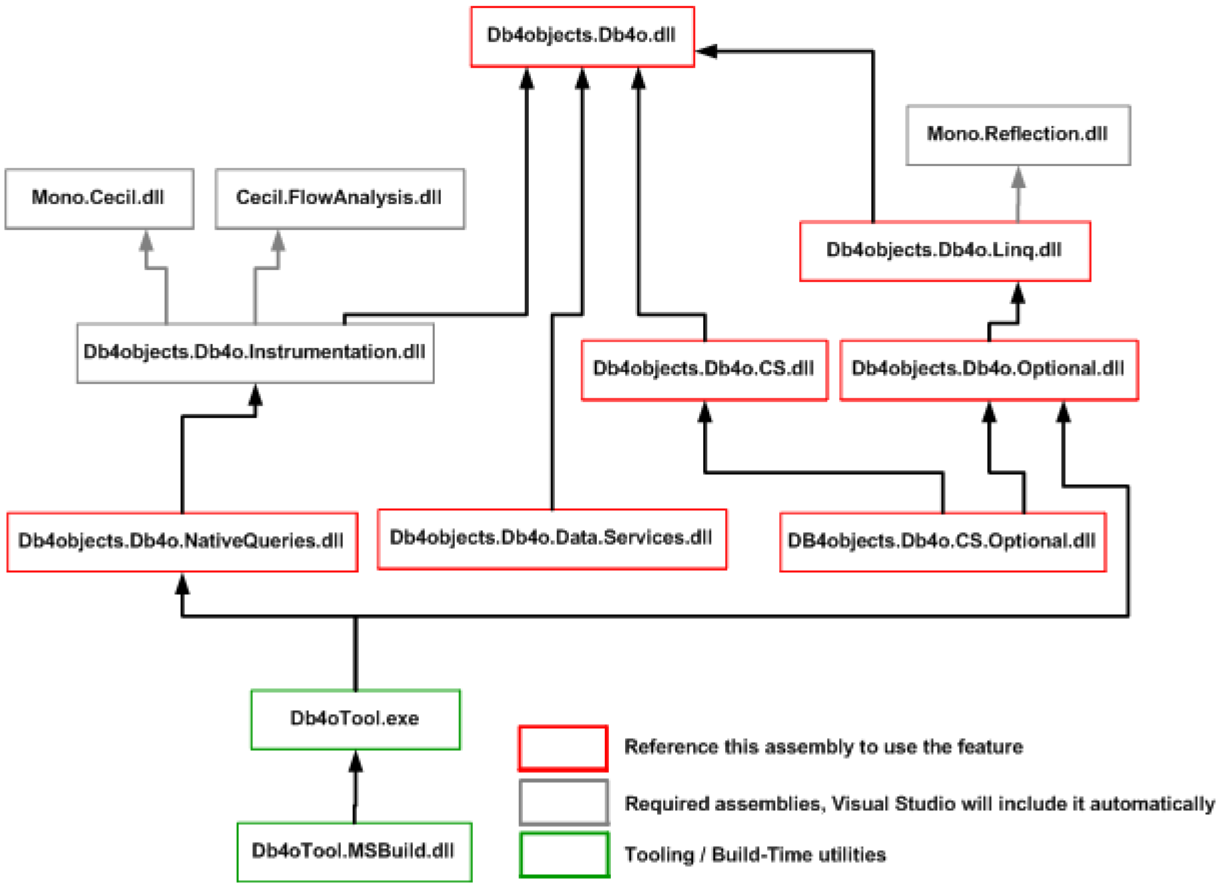
- select the project in Package Explorer;
- right-click and select Team/Create Patch;

- 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.

Dependency Overview

The different functionality of db4o is implemented in multiple assemblys. You need to include only the assembly which are required for you're application. Here's an overview:



Db4objects.Db4o.dll
Mono.Reflection.dll
Mono.Cecil.dll
Cecil.FlowAnalysis.dll
Db4objects.Db4o.Linq.dll
Db4objects.Db4o.Instrumentation.dll
Db4objects.Db4o.CS.dll
Db4objects.Db4o.Optional.dll
Db4objects.Db4o.NativeQueries.dll
Db4objects.Db4o.Data.Services.dll
DB4objects.Db4o.CS.Optional.dll
Db4oTool.exe
Db4oTool.MSBuild.dll
Reference this assembly to use the feature
Required assemblies, Visual Studio will include it automatically
Tooling / Build-Time utilities

Db4objects.Db4o.dll

This assembly contains the core functionality of db4o. It doesn't have any dependencies to other assemblies

Db4objects.Db4o.Linq.dll

This assembly contains the LINQ-support for db4o. It has a dependency to Db4object.Db4o.dll and Mono.Reflection.dll.

In Visual Studio, you don't need to add the Mono.Reflection.dll as a reference, Visual Studio will include this dependency automatically.

On the .NET compact framework, this assembly depends on Cecil.FlowAnalysis.dll and Mono.Cecil.dll instead of Mono.Reflection.dll.

Db4objects.Db4o.CS.dll

This assembly contains the client-server support for db4o. It depends on the Db4object.Db4o.dll and nothing else.

Db4objects.Db4o.Data.Services.dll

This assembly contains the ADO.NET Data Services implementation for db4o. It depends on the Db4object.Db4o.dll assembly. While this assembly doesn't depend on the Db4object.Db4o.Linq.dll, you still need that assembly for the ADO.NET Data Services LINQ query support.

Db4objects.Db4o.Optional.dll

This assembly contains some additional features and functionality for db4o, like monitoring capabilities and Java support.

Db4objects.Db4o.CS.Optional.dll

This assembly contains some additional features and functionality for the db4o client server support. Like SSL and monitoring capabilities. It depends on the Db4objects.Db4o.dll, Db4objects.Db4o.CS.dll, Db4objects.Db4o.Optional.dll and the Db4objects.Db4o.LINQ.dll assembly.

Db4objects.Db4o.NativeQueries.dll

This assembly contains the native query optimization. It depends on Db4object.Db4o.dll, Db4object.Db4o.Instrumentation.dll, Cecil.FlowAnalysis.dll and Mono.Cecil.dll.

In Visual Studio, you don't need to add Db4object.Db4o.Instrumentation.dll, Cecil.FlowAnalysis.dll and Mono.Cecil.dll as a reference. Visual Studio will include this dependency automatically.

Note that native queries work also without this assembly. However the queries cannot be optimized without this assembly and will run a lot slower.

Db4oTool.exe

This is a command line tool for doing various enhancement to your classes. It depends on Db4object.Db4o.dll, Db4object.Db4o.Linq.dll, Db4object.Db4o.Optional.dll, Db4object.Db4o.NativeQueries.dll, Db4object.Db4o.Instrumentation.dll, Cecil.FlowAnalysis.dll and Mono.Cecil.dll

Db4oTool.MSBuild.dll

This is the MSBuild fronted for built time enhancement. It has the same dependencies as Db4oTool.exe and Db4oTool.exe itself.

License

Licensing the db4o Engine

Versant Inc. offers three difference license options for the db4o object database engine db4o:

General Public License (GPL) Version 3

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
- Versant's 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 .NET versions

- http://mono-project.com/Cecil Mono.Cecil (MIT/X11)

  Used inside Db4objects.Db4o.Instrumentation.dll and Db4oTool.exe

  Cecil is used to read the CIL code during native queries optimization.
- http://mono-project.com/Cecil .FlowAnalysis (MIT/X11)

  Used inside db4o support libraries and Db4oTool.exe

  Cecil.FlowAnalysis is used to analyse the CIL code during native queries optimization.
- http://mono-project.com/ Mono.GetOptions (MIT/X11)

  Used inside Db4oTool.exe

  Mono.GetOptions is used to parse the command line of the Db4oTool.exe tool.

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.

Contacts

Join The db4o Community

Join the db4o community for help, tips and tricks. Ask for help in the db4o forums at any time. And take a look at additional resources on the community website. If you want to stay informed, subscribe to db4o blogs.

Address and Contact Information

Versant Corporation

255 Shoreline Drive, Suite 450

Redwood City, CA 94065

USA

Phone

+1 (650) 232-2436

Fax

+1 (650) 232-2401

Sales

Fill out our sales contact form on the db4o website

or mail to sales@db4o.com

Careers

career@db4o.com

Partnering

partner@db4o.com