Brought to you by... db4objects BY VERSANT

CONTENTS INCLUDE:

- About db4o and Object Databases
- Getting Started
- Basic Database Operations
- Queries
- Dealing with Object Activation
- Hot Tips and more...

Getting Started with db4o:

Persisting .NET Object Data

By Stefan Edlich and Eric Falsken

ABOUT DB4O AND OBJECT DATABASES

db4o is the open source object database that enables developers to store and retrieve any application object with one line of code.

The Object Database (ODB) arrived in the software industry with the advent of object oriented languages. The ODB is primarily used as an application specific database in either extreme scale systems or embedded systems where typical DBA activities are automated.

If you are familiar with object-relational mapping (ORM) tools, then you are already an object database expert because many of the APIs and query languages are comparable. You will even find db4o both familiar and yet simpler to use as ORM tools as there is no requirement for XML mapping, annotations or IDs.

db4o was introduced in 2000 with a focus on performance, compactness, zero administration, simplicity and (the most important of all) native object persistence. Developers were finally able to combine the power of a full database engine with plain undecorated objects.

GETTING STARTED

Note The current version of db4o at the time of this writing is version 7.8.

db4o comes distributed as a few native .NET assemblies. Only `Db4objects.Db4o.dll` is required for basic db4o operation. Use the Visual Studio "Add Reference" command to add the necessary assemblies. Then use the Solution Explorer to locate the new reference, right-click and open the properties window to ensure that "Copy Local" is set to true for each db4o assembly. This will copy the necessary db4o libraries to your application's `bin` folder automatically when compiling your project.

Note Running db4o from the GAC is not supported.

Required Environment

Developing with db4o requires only the .NET SDK version 2.0 or better. (3.5 suggested)

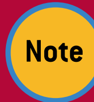
Visual Studio 2008 or better is suggested for the best experience, but any .NET IDE will do. Microsoft Visual Studio 2008 Express editions are available for free download from Microsoft.

Running db4o requires only the .NET Framework to be installed. Some hosting environments, such as shared website hosting providers, do not allow code to run with full trust. When the environment is configured to run with reduced trust, all basic database operations require at least ReflectPermission (`MemberAccess` and `ReflectionEmit`) for the classes and types being persisted.

Required Libraries per Database Feature

Depending on the features your application requires, reference and distribute these assemblies when you distribute your application:

Feature	Assembly
(Required for all Installations)	`Db4objects.Db4o.dll`
Client-Server	`Db4objects.Db4o.CS.dll`
LINQ	`Db4objects.Db4o.Linq.dll`
Native Queries	`Db4objects.Db4o.NativeQueries.dll`
Transparent Activation	`Db4objects.Db4o.Instrumentation.dll`



Run-time Optimization of LINQ, NQ, and TA requires the `Mono.Cecil.dll` and `Cecil.FlowAnalysis.dll` assemblies. Optimization can also be done at build-time using `Db4oTool.exe`. (see db4o Reference Documentation for usage)

ObjectManager for Debugging

Included in the db4o distribution you'll find the installer for ObjectManager Enterprise (OME) which, once installed, will integrate into your Visual Studio Environment and allow you to open and inspect, query, and edit (value types only) object instances stored in your database file.

VERSANT
the Database for Objects
OEM
Jump Start
ISVs and VARs Get a $36,000 OEM Jump Start Package for Free!
www.versant.com

BASIC DATABASE OPERATIONS

A Complete Example

```
IObjectContainer db =
    Db4oFactory.OpenFile([filename]), ([config]);
try{
    // Store a few Person objects
    db.Store(new Person("Petra"));
    db.Store(new Person("Gallad"));
    // Retrieve the Person
    var results = db.Query<Person>(x => x.Name == "Petra");
    Person p = result.First();
    // Update the Person
    p.Name = "Peter";
    db.Store(p);
    // Delete the person
    db.Delete(p);
    // Don't forget to commit!
    db.Commit();
}
catch{
    db.Rollback();
}
finally{
   // Close the db cleanly
   db.Close();
}
```

Create or Open a Database File

```
IObjectContainer container = Db4oFactory.OpenFile([filename]);
```

While a db4o database file is open, it is locked and cannot be accessed by another application at the same time.

Hot Tip It's important to know that db4o works best if you open the database file when you start working with data, and close the db when all posible operations are completed.

In traditional relational databases, it's common to open a connection, get/update data, close connection, and then perform your operation. Because db4o uses the native object (or referential) identity, it's better to open the database or connection when your application begins, do all your work, then close the database when your program is shutting down. You'll see why when we get to updating an object with our changes.

Starting a db4o Server

By default, the db4o server runs in-process within your application. To start a db4o server, place this code into your application:

```
IObjectServer server =
    Db4oFactory.OpenServer([filename], [port]);
server.GrantAccess([user], [password]);
```

To shut down the server:

```
server.Close();
```

The port parameter specifies the network port number. Acceptable values are any number above 1024 which are not already in use.

Hot Tip Using Port 0 for your server creates an "Embedded" server which will not be available remotely. This is useful for multi-threaded operations or web-server style environments where you wish to handle parallel operations in a single process.

The `GrantAccess()` method must be called for each username/password combination. It is not required at all for embedded servers.

Connecting to a db4o Server

After starting a db4o server instance, use either of the commands below to open a db4o client connection:

```
// In-Process mode (embedded server)
IObjectContainer client = server.OpenClient();

// Client/Server mode (remote server)
IObjectContainer client =
    Db4oFactory.OpenClient([serverAddress], [port], [user],
[password]);
```

To close the client connection to the server:

```
client.Close();
```

Storing an Object

```
db.Store(anObject);
db.Commit()
```

Just one line of code is all it takes. All of an object's properties and child objects will be stored.

Updating an Object

```
db.Store(anObject);
db.Commit();
```

Looks familiar? You can use the same `Store(object)` command to update an object. One difference, however, is that db4o will (for performance reasons) not automatically check child objects for changes.

Hot Tip By default, db4o will NOT descend into child objects. Store() must be called for each modified object unless you change the default UpdateDepth (see the UpdateDepth parameter in the Configuration section, below) or configure cascading update for the persisted class.

There is one more thing to be aware of: db4o uses an object's native identity (referential identity) to identify an object and map it to the stored instance of the object. This means that there is only ever 1 instance of an object in memory for each stored instance of the object. (per `ObjectContainer` or connection) This is important when dealing with class instances that may come from ASP.NET ViewState, Web Services, Interop, or any other serialized source of object data.

Hot Tip Avoid Confusion: Always make sure that the object you are trying to update or delete was previously stored or retrieved in the database. Calling Store() with 2 User objects both with an ID of "jack" will result in 2 separate instances. However, if you retrieve the user, and modify the first instance, then store it again, you will have only 1 updated instance in the database.

Deleting an Object

```
db.Delete(anObject);
db.Commit();
```

You didn't think it was any harder than that, did you? Like updates, db4o will not automatically delete child objects unless you configure cascading deletes for your object will remain in memory until the objects are refreshed or garbage collected.

Database Transactions

Whenever you start making changes to your database (using the `Store()` and `Delete()` commands) you are automatically in

an open transaction. To close the transaction, use the `Commit()` method:

```
db.Commit();
```

Your changes will be permanently saved. If you wish to cancel or roll back any uncommitted changes, use the `Rollback()` method:

```
db.Rollback();
```

Hot Tip Useful for beginners: Rollback only undoes uncommitted changes in the database. It will not undo changes to any currently loaded objects. So, when you call `Rollback()` you will not see any difference to your objects. If concerned about consistency, use the `Refresh(object)` command to cause the objects to be refreshed with stored database values.

Closing a database cleanly will automatically call `Commit()` for you, so any uncommitted transactions are committed automatically.

If the database is not closed cleanly, or if the application crashes at any time and uncommitted (or incomplete) transactions are discarded.

QUERIES

Query by Example (QBE)

QBE lets you pass db4o an example object. db4o will search and return all objects which match the object you specify. To do this, db4o will reflect all of the properties of the object and assemble all non-default property values into a single query expression.

Hot Tip Useful for beginners: QBE queries are not able to use advanced boolean constraints (AND, OR, NOT) and cannot constrain on default values (zero, empty strings, null). QBE queries also cannot query for value ranges (greater than, less than) or string-based expressions (contains, starts with). So QBE can be used only to retrieve exact-value matches.

Here's an example QBE query that will contain all `Customer` objects with "Lee" as their Surname:

```
Customer proto = new Customer ()
      {Surname = "Lee"};
IObjectSet result = db.Get(proto);
```

Native Queries (NQ)

Like any query language, Native Queries are capable of expressing complex parameterized constraints, however NQ also have the benefit of being 100% compiler checked and can be refactored using common code refactoring tools. NQ can do all this because they are expressed as native .NET code rather than as strings (like SQL statements)

Hot Tip Developers are encouraged to use the Native Query interfaces when working with db4o.

```
// NQ Lambda Expression (.NET 3.5 syntax)
var result = db.Query<type>(
    o => [boolean expression]);

// NQ Anonymous Method (.NET 2.0 syntax)
IList<type> result = db.Query<type>(
    delegate(type o){
        return [boolean expression];
    });
```

Native Query Examples

```
// Query all instances of a type.
IList<User> result = db.Query<User>();

// Query Users by Name.
IList<User> result = db.Query<User>(u => u.Name == "Joe");

// Query Users with at least 10 orders
IList<User> result = db.Query<user>(u => u.Orders.Count >= 10);

// Complex Query
IList<User> result = db.Query<User>(
    u => u.Name.StartsWith("Bob")
      && (u.Country == "Canada" ||
          u.Country == "USA"    );
```

Sorting Native Queries

Native Query results can be sorted by using an `IComparer` or a comparison delegate. Here is the query syntax:

```
IList<User> result = db.Query<User>([predicate],[comparer]);
```

And here's an example:

```
// All users with "Smith" in their name
// sorted by name
IList<User> result = db.Query<User>(
    u => u.Name.Contains("Smith"),
    (u1, u2) => u1.Name.CompareTo(u2.Name)
    );
```

LINQ Queries

LINQ was introduced by Microsoft .NET Framework version 3.5 (also called C# 3.0) in November 2007. To enable the use of LINQ queries, you'll need to add a reference to the `Db4objects.Db4o.LINQ.dll` assembly and import the `Db4objects.Db4o.LINQ` namespace. (with a using statement)

A full description of LINQ syntax will not fit in this document. You can find the URL to Microsoft's LINQ reference in the Resources section at the end.

LINQ Queries have all the benefit of compiler checking and automated code refactorability that Native Queries have, but are expressed in syntax more familiar to SQL developers. Here's one quick and easy example that gets all of the Customers with "Smith" in their name, and sorts the results by name.

```
var results =
    from Customer c in db
    where c.Name.Contains("Smith")
    orderby c.Name descending
    select c;
```

With LINQ queries you can:

- Use `ORDERBY` to sort the results.
- Use `JOIN` expressions to filter one dataset based on the contents of another.
- Use Aggregate Expressions to get the sum, min, max, or average values and `GROUPBY` to group the results.
- Use Anonymous Types to get back only the fields you wish.
- Use an expression to process, filter, or format the data as it is returned from the database.
- Use LINQ extensions like `First()`, `Any()`, `All()`,

Single(), and ElementAt() to access or constrain the result set.

SODA Queries

SODA query expressions are a standard that was present in all but the earliest versions of db4o. Using combinations of SODA query and constraint keywords, you can build up what is called a query "graph". A graph is a network of objects which represent a segment of data.

Before using SODA Queries, you must import the Db4objects.Db4o.Query Namespace. (with a using statement)

In this example, we're querying for all of the customers named "Smith".

```
IQuery query = db.Query();
query.Constrain(typeof(Customer));
query.Descend("Name").Constrain("Smith");
IObjectSet results = query.Execute();
```

And in this more complicated example, we'll use a compound constraint:

```
IQuery query = db.Query();
query.Constrain(typof(Customer));
IConstraint c1 = query
    .Descend("Name")
    .Constrain("Smith");
IConstraint c2 = query
    .Descend("Country")
    .Constrain("USA");
c1.And(c2);
IObjectSet results = query.Execute();
```

Notice how each of the calls to Constrain() will return an IConstraint? You can keep references to those constraints and then use constraint keywords like And(), Or(), and Not() to relate the constraints together, as we did at the end of that example.

Note that the Descend() method returns an IQuery too. So you could Descend() into an object, and then Execute() at a deeper level to return only the matching child objects, like in this example:

```
IQuery q1 = db.Query();
q1.Constrain(typeof(Customer)); // start with Customers
q1.Descend("Name")
  .Constrain("Smith");
IQuery q2 = q1.Descend("Orders"); // constrain Orders
// Order totals greater than $100.
q2.Descend("Total")
  .Constrain(100)
  .Greater();
// Since we want to return Orders, execute q2 instead of q1
IObjectSet results = q2.Execute();
```

SODA Query Interfaces	
IQuery	Provides the location for constraining or selecting.
IConstraint	Constrains the query results with the current IQuery node

SODA Query Keywords	
Descend	Move from one node to another.
OrderAscending	Order the result ascending according to the current node.
OrderDescending	Order the result descending according to the current node.
Execute	Execute the query graph and return the objects at the current node.

SODA Constraint Keywords	
And(IConstraint)	Performs an AND (&&) comparison between 2 constraints.
Contains()	For collection nodes, matches will contain the specified value. For string values, behaves as Like().
EndsWith(bool)	For strings, matches will end with the supplied value. Optionally case sensitive.
Equal()	Combine with Smaller and Greater to include the specified value. (e.g. >= or <=)
Greater()	Matching values will be greater than or larger than the supplied value.
Identity()	Matching values will be the same object instance as the supplied value. (referential equality).
Like()	For strings, matching values will contain the supplied value anywhere within the match.
Not()	Performs a negation comparison. Matching values will NOT equal the supplied value. Added to any other constraint keyword, this will reverse the result.
Or(IConstraint)	Performs an OR (||) comparison between 2 constraints.
Smaller()	Matching values will be smaller or less than the specified value.
StartsWith(bool)	For strings, matches will start with the supplied value. Optionally case sensitive.

Query Performance

- If you are experiencing poor NQ performance, then you probably forgot to either enable run-time optimization of Native Queries (by including a reference to the assemblies listed in the "Required Libraries" section) or running Db4oTool.exe on your compiled assembly.
- You can change the Query EvaluationMode configuration to control how and when a query should be evaluated. (See below in the Configuration section)
- You can index fields to aid query evaluation. Indexing fields causes db4o to store the values of the field in a separate index lookup table in the db. As a result, when evaluating the query, db4o does not have to seek through all of the object data to resolve the query results. (See the Class-Specific Configuration Options below)

Indexing fields is a great way to increase query performance, but each index table is one more place where a field's value is stored. Too many indexed fields can cause poor insert performance. The application developer should tune the number of indexes with the desired Query and Insert performance.

DEALING WITH OBJECT ACTIVATION

When dealing with objects that may have relations to other objects quite deep (think of the path of data from Customer to Order to OrderItem to Product with relations to Address objects for billing and shipping and then PO and payment transactions) it would be quite expensive to have to pull all of that data into memory from the DB if all you wanted was the Customer object. Modern object databases use the idea of activation to control the depth to which objects are instantiated and populated when retrieved from the database.

The default ActivationDepth in db4o is 5.

A properly tuned activation depth is the best way to optimize retrieval of data from a db4o database. (See ActivationDepth in the Configuration section for more ideas)

With an ActivationDepth of 5, objects will be populated up to 5 levels deep. Properties of the 5th descendant object will have their values left as default or null.

If you encounter an object that is not yet activated, you can pass it to db4o for manual (late) activation:

```
db.Activate([unactivatedObject], [depth]);
```

Objects can also be manually de-activated:

```
db.Deactivate([activatedObject], [depth]);
```

Fine-grained activation depth can be configured per class. (see `ActivationDepth` in the Configuration section below) Activation can also be managed transparently using Transparent Activation. (see below)

CONFIGURATION

For all but the simplest db4o use cases, you'll probably want to specify one or more configuration settings when opening your db4o database:

```
IConfiguration config =
Db4oFactory.NewConfiguration();
// Set configuration properties here
IObjectContainer db= Db4oFactory.OpenFile([config], [filename]);
```

The `IConfiguration` object must be passed in the call to open the db4o file, server, or client connection.

Query EvaluationMode

This property controls when and how much of a query is executed.

```
IConfiguration config =
    Db4oFactory.NewConfiguration();
config.Queries().EvaluationMode([mode]);
```

Query EvaluationMode Values	
`Immediate`	(Best when queries must be deterministic or execute as quickly as possible.) A list of object ID matches is generated completely when the query is executed and held in memory
`Lazy`	(Best for limited resource environments.) An iterator is created against the best index found. Accessing the results will simply iterate through the index until no further matches are found. The result set can be influenced by subsequent transactions in the current context or by other clients causing possible concurrency errors. Almost no memory is needed to hold the result set. Accessing the Count or Length properties will cause full evaluation.
`Snapshot`	(Best for servers and concurrent environments.) Same as Lazy, however the iterator is created against a snapshot of the index. This avoids possible concurrency issues of Lazy evaluation. Since the index snapshot is held in memory, the memory required varies greatly depending on what is being queried.
`(All Values)`	Object data is activated as the users accesses each object in the result set. The currently stored state is used when activating field data, not the object state at the time of query execution. The above values affect only the state of field indexes and when the evaluation is performed. Regardless of when it is run, query constraints against non-indexed data are always performed on the currently stored object state.

Global UpdateDepth

We said earlier that when calling `Store()` to update an object graph, that db4o will not (by default) descend into child objects to detect changes to the graph. If you know that you'll be often changing child properties, or when changing a parent object often results in changes to child objects, then you may want to change the `UpdateDepth`.

```
IConfiguration config =
Db4oFactory.NewConfiguration();
config.UpdateDepth([depth]);
```

The default value 1 means db4o will not descend into child objects when updating stored object instances.

Setting the `UpdateDepth` to `int.MaxValue` will cause db4o to descend as deeply as possible to look for changes.

Setting the `UpdateDepth` to `0` will prevent any changes from being saved to the database.

Setting the UpdateDepth too aggressively can cause poor db4o update performance. Higher values should be used to debug UpdateDepth-related issues only.

Global ActivationDepth

As explained in the section on Dealing with Object Activation, the `ActivationDepth` controls how much data is loaded when an object is retrieved from the database.

```
Iconfiguration config =
    Db4oFactory.NewConfiguration();
config.ActivationDepth([depth]);
```

The default value of 5 is a good balance for most applications, but developers should balance this against the weight of their classes and their access patterns.

Setting the `ActivationDepth` to `int.MaxValue` will cause all related objects to be instantiated as deeply as possible, restoring the entire object graph to memory.

Setting the `ActivationDepth` to `0` will cause nothing to be activated. The object returned will have none of its values loaded. You can then call the `Activate(object, depth)` method to manually activate the object as described above.

Setting the ActivationDepth too aggressively can cause poor db4o query performance and high memory usage. Higher values should be used to debug ActivationDepth-related issues only.

Transparent Activation (TA)

The `Db4oTool.exe` tool found in the distribution `bin` folder can be used to instrument your classes to perform activation on-demand transparently as you navigate between object references. When accessing a child object, db4o can automatically activate the child object from the database if it is not yet already activated.

Transparent Activation must be enabled in the configuration when opening a db4o database and your compiled assembly must be instrumented by `Db4oTool.exe` as part of the build (as an MSBuild task) or post-build by running the tool manually.

Using `Db4oTool.exe` is not complicated, but is beyond the scope of this DZone Refcard. You can find complete instructions in the db4o reference documentation.

CASCADING OPERATIONS, CALLBACKS AND CLASS-SPECIFIC CONFIGURATION

The global `UpdateDepth` and `ActivationDepth` configurations are good for general testing. But oftentimes, you will want to configure specific behaviors per-class-type, or per-field. (e.g. field indexing)

```
IConfiguration config =
Db4oFactory.NewConfiguration();
//Get the class-specific canfiguration
IObjectClass objectClassConfig =
    config.ObjectClass([typeName]);
//Get a field-specific configuration
IObjectField objectFieldConfig =
    objectClassConfig.ObjectField([fieldName]);

// Usually shortened to one line. e.g. config.
objectClass(typeof(Customer)).ObjectField ("Name").Indexed(true);
```

A full description of Class-specific and field-specific configuration settings can be found in the db4o reference documentation.

db4o also allows you to specify event handlers when an object is retrieved, activated, deleted, etc. These events can be handled globally using the `EventRegistryFactory` or individually within your classes using any of the `IObjectCallbacks` methods like `objectOnActivate`, `objectOnDelete`, and `objectOnNew`. These handlers are great for selectively activating, refreshing, or cascading db operations.

Lastly, setting the `Transient` attribute on your class's field members will prevent db4o from storing the values of those fields. When read from the DB, these members will always be left at their default (null) values, useful for remote connections, non- native data (interop) and temporary state.

DB4O RESOURCES

db4o Homepage	http://www.db4o.com
db4o Community Forum	http://developer.db4o.com
db4o Downloads	http://download.db4o.com
db4o Bug Tracker	http://tracker.db4o.com
db4o Source Code Repository	https://source.db4o.com/db4o/ (https only)
db4o Reference Documentation	http://docs.db4o.com
db4o Community Projects	http://projects.db4o.com
LINQ General Programming Guide	http://msdn.microsoft.com/en-us/library/bb397912.aspx
Versant Corporation	http://www.versant.com
Versant Corporation supports the db4o team and open source community. Versant is a public company (NASDAQ:VSNT) and develops the Versant Object Database technology for users of .NET, C++ and Java who require a database capable of supporting extreme scale systems.	

ABOUT THE AUTHOR

Prof. Dr. Stefan Edlich is a senior lecturer at Beuth University of Technology Berlin (App.Sc.) with a focus on Object Databases, Software-Engineering and E-Learning. He sold his first commercial software in 1986 and has a 26 year software development experience. Furthermore he is the author of ten IT books he wrote for Apress, OReilly, Spektrum / Elsevier, and other publishers. In 2008 he set up the the worlds First International Conference on Object Databases (ICOODB.org) which is continued 2009 at ETH-Zürich.

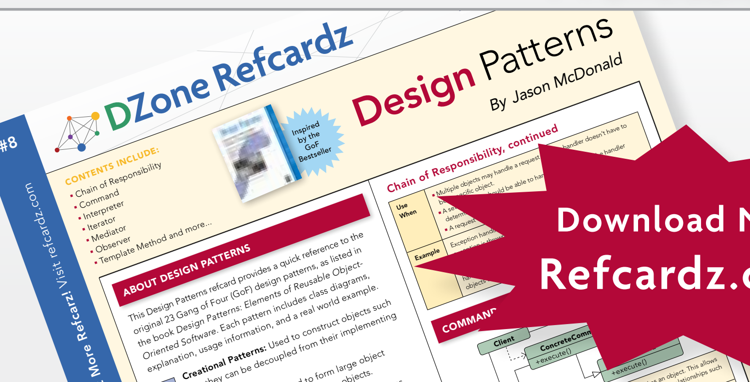
Eric Falsken is a longtime web and embedded software developer. He wrote his first of many websites in 1995, and went on to e-commerce, enterprise and internet media before shifting his focus to embeded device software, where he found his love of object databases and db4o. Eric has been a member of the db4o team since 2006, and enjoys travel adventuring, meeting new people, and looking at beautiful code.

RECOMMENDED BOOK

The Definitive Guide to db4o is the first book to comprehensively cover this project in detail. You will learn about all relevant topics, including installing and configuring db4o, querying and managing objects, performing transactions, and replicating data. To aid newcomers to the topic, early chapters cover object database fundamentals, as well as technical considerations and migration strategies. The book is complete with numerous C# and Java examples, so you'll be able to follow along regardless of your chosen language.

BUY NOW
books.dzone.com/books/definitive-db4o

Professional Cheat Sheets You Can Trust



"Exactly what busy developers need: simple, short, and to the point."

James Ward, Adobe Systems

Upcoming Titles
JavaFX
JSF 2.0
Maven
Drupal
Java Performance Tuning
Eclipse RCP
ASP.NET MVC Framework

Most Popular
Spring Configuration
jQuery Selectors
Windows Powershell
Dependency Injection with EJB 3
Netbeans IDE JavaEditor
Getting Started with Eclipse
Very First Steps in Flex

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheatsheets, blogs, feature articles, source code and more. **"DZone is a developer's dream,"** says PC Magazine.

DZone, Inc.
2222 Sedwick Rd Suite 101
Durham, NC 27713
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

Copyright © 2009 DZone, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher. Reference: The Definitive Guide to db4o, Stefan Edlich, Henrik Horning, Reidar Horning, Jim Paterson, APress, June 2006

Version 1.0