Welcome

db4o is the native Java, .NET and Mono open source object database.

This documentation and tutorial is intended to get you started with db4o and to be a reliable companion while you develop with db4o. Before you start, please make sure that you have downloaded the latest db4o distribution from the db4objects website.

The db4o Mono distribution comes as either a binary or source RPM package, db4o-4.5.noarch.rpm and db4o-4.5.src.rpm respectively. After you install the noarch RPM package you should find the db4o.dll assembly in /usr/lib/db4o (or in the respective path of your Linux distribution).

This tutorial comes in multiple versions. Make sure that you use the right one for the right purpose.

/usr/share/doc/packages/db4o/tutorial/index.html
The tutorial in HTML format. The Java and .NET db4o distributions also provide the HTML documentation with live execution capabilities.

/usr/share/doc/packages/db4o/tutorial/db4o-4.x-tutorial.pdf
The PDF version of the tutorial allows best fulltext search capabilities.

Java, .NET and Mono
db4o is available for Java, for .NET and for Mono. This tutorial was written for Mono. The structure of the other distributions may be considerably different, so please use the tutorial for the version that you plan to experiment with first.

1. First Glance
Before diving straight into the first source code samples let's get you familiar with some basics.

1.1. The db4o engine...
The db4o object database engine consists of one single DLL. This is all that you need to program against. The version supplied with the distribution can be found in /usr/lib/db4o/.☐

1.2. Installation
To use db4o in a development project, you only need to add one of the above db4o.dll files to your project references.

Here is how to do this if you are using MonoDevelop:
- Right-click on "References" in the Solution Tab
- choose "Edit References"

- select the ".NET Assembly" tab and then "Browse"
- select /usr/lib/db4o/db4o.dll
- click "Open"
- click "OK"

1.3. API

The API documentation for db4o is supplied as a set of HTML pages in /usr/share/doc/packages/db4o/api/index.html. While you read through this tutorial, it may be helpful to look into the API documentation occasionaly. For the start, the namespaces com.db4o and com.db4o.query are all that you need to worry about.

Let's take a first brief look at one of the most important interfaces:

```
com.db4o.ObjectContainer
```

This will be your view of a db4o database:
- An ObjectContainer can either be a database in single-user mode or a client to a db4o server.
- Every ObjectContainer owns one transaction. All work is transactional. When you open an ObjectContainer, you are in a transaction, when you commit() or rollback(), the next transaction is started immediately.
- Every ObjectContainer maintains it's own references to stored and instantiated objects. In doing so, it manages object identities.

In case you wonder why you only see very few methods in an ObjectContainer, here is why: The db4o interface is supplied in two steps in two namespaces, com.db4o and com.db4o.ext for the following reasons:
- It's easier to get started, because the important methods are emphasized.
- It will be easier for other products to copy the basic db4o interface.
- We hint how a very-light-version of db4o should look like.

Every com.db4o.ObjectContainer object also always is a com.db4o.ext.ExtObjectContainer. You can cast to ExtObjectContainer or you can call the #ext() method if you want to use advanced features.

2. First Steps

Let us get started as simple as possible. We are going to learn how to store, retrieve, update and delete instances of a single class that only contains primitive and String members. In our example this will be a Formula One (F1) pilot whose attributes are his name and the F1 points he has already gained this season.

First we create a native class such as:

```
namespace com.db4o.f1.chapter1
{
    public class Pilot
    {
        string _name;
        int _points;

        public Pilot(string name, int points)
        {
            _name = name;
            _points = points;
        }

        public string Name
        {
            get
            {
                return _name;
            }
        }

        public int Points
        {
            get
            {
                return _points;
            }
        }

        public void AddPoints(int points)
```

```
        {
            _points += points;
        }

        override public string ToString()
        {
            return _name + "/" + _points;
        }
    }
}
```

Note that this class does not contain any db4o related code.

2.1. Storing objects

To access a db4o database file or create a new one, call Db4o.openFile(), providing the path to your file as the parameter, to obtain an ObjectContainer instance. ObjectContainer will be your primary interface to db4o. Closing the container will release all resources associated with it.

```
[accessDb4o]

ObjectContainer db=Db4o.openFile(Util.YapFileName);
    try
    {
        // do something with db4o
    }
    finally
    {
        db.close();
    }
```

For the following examples we will assume that our environment takes care of opening and closing the ObjectContainer automagically.

To store an object, we simply call set() on our database, passing the object as a parameter.

```
[storeFirstPilot]

Pilot pilot1 = new Pilot("Michael Schumacher", 100);
    db.set(pilot1);
    Console.WriteLine("Stored " + pilot1);
```

We'll need a second pilot, too.

```
[storeSecondPilot]

Pilot pilot2 = new Pilot("Rubens Barrichello", 99);
    db.set(pilot2);
    Console.WriteLine("Stored " + pilot2);
```

2.2. Retrieving objects

To query the database for our pilot, we shall use *Query by Example* (QBE) for now. This means we will create a prototypical object for db4o to use as an example. db4o will retrieve all objects of the given type that contain the same (non-default) field values as the candidate. The result will be handed as an ObjectSet instance. We will use a convenience method 'listResult', inherited by all our main example classes, to display a result's content and reset it for further use:

```
public static void listResult(ObjectSet result)
    {
        Console.WriteLine(result.size());
        while (result.hasNext())
        {
            Console.WriteLine(result.next());
        }
    }
```

To retrieve all pilots from our database, we provide an 'empty' prototype:

```
[retrieveAllPilots]

Pilot proto = new Pilot(null, 0);
    ObjectSet result = db.get(proto);
    listResult(result);
```

Note that our results are not constrained to have 0 points, as 0 is the default value for int fields.

To query for a pilot by name:

```
[retrievePilotByName]

Pilot proto = new Pilot("Michael Schumacher", 0);
    ObjectSet result = db.get(proto);
    listResult(result);
```

Let's retrieve a pilot by exact points:

```
[retrievePilotByExactPoints]

Pilot proto = new Pilot(null, 100);
    ObjectSet result = db.get(proto);
    listResult(result);
```

Of course there's much more to db4o queries. We'll come to that in a moment.

2.3. Updating objects

To update an object already stored in db4o, just call set() again after modifying it.

```
[updatePilot]

ObjectSet result = db.get(new Pilot("Michael Schumacher", 0));
    Pilot found = (Pilot)result.next();
    found.AddPoints(11);
    db.set(found);
    Console.WriteLine("Added 11 points for " + found);
    retrieveAllPilots(db);
```

Note that it is necessary that db4o already 'knows' this pilot, else it will store it as a new object. 'Knowing' an object basically means having it set or retrieved during the current db4o session. We'll explain this later in more detail.

To make sure you've updated the pilot, please return to any of the retrieval examples above and run them again.

2.4. Deleting objects

Objects are removed from the database using the delete() method.

```
[deleteFirstPilotByName]

ObjectSet result = db.get(new Pilot("Michael Schumacher", 0));
    Pilot found = (Pilot)result.next();
    db.delete(found);
    Console.WriteLine("Deleted " + found);
    retrieveAllPilots(db);
```

Let's delete the other one, too.

```
[deleteSecondPilotByName]
```

```
ObjectSet result = db.get(new Pilot("Rubens Barrichello", 0));
    Pilot found = (Pilot)result.next();
    db.delete(found);
    Console.WriteLine("Deleted " + found);
    retrieveAllPilots(db);
```

Please check the deletion with the retrieval examples above.

Again, the object to be deleted has to be known to db4o. It is not sufficient to provide a prototype object with the same field values.

2.5. Conclusion

That was easy, wasn't it? We have stored, retrieved, updated and deleted objects with a few lines of code. But what about complex queries? Let's have a look at the restrictions of QBE and alternative approaches in the next chapter .

2.6. Full source

```
namespace com.db4o.f1.chapter1
{
    using System;
    using System.IO;
    using com.db4o;
    using com.db4o.f1;

    public class FirstStepsExample : Util
    {
        public static void Main(string[] args)
        {
            File.Delete(Util.YapFileName);
            accessDb4o();
            File.Delete(Util.YapFileName);
            ObjectContainer db = Db4o.openFile(Util.YapFileName);
            try
            {
```

```
            storeFirstPilot(db);
            storeSecondPilot(db);
            retrieveAllPilots(db);
            retrievePilotByName(db);
            retrievePilotByExactPoints(db);
            updatePilot(db);
            deleteFirstPilotByName(db);
            deleteSecondPilotByName(db);
        }
        finally
        {
            db.close();
        }
    }

    public static void accessDb4o()
    {
     ObjectContainer db=Db4o.openFile(Util.YapFileName);
     try
     {
         // do something with db4o
     }
     finally
     {
         db.close();
     }
}

    public static void storeFirstPilot(ObjectContainer db)
    {
        Pilot pilot1 = new Pilot("Michael Schumacher", 100);
        db.set(pilot1);
        Console.WriteLine("Stored " + pilot1);
    }

    public static void storeSecondPilot(ObjectContainer db)
    {
        Pilot pilot2 = new Pilot("Rubens Barrichello", 99);
        db.set(pilot2);
        Console.WriteLine("Stored " + pilot2);
    }
```

```
        public static void retrieveAllPilots(ObjectContainer db)
        {
            Pilot proto = new Pilot(null, 0);
            ObjectSet result = db.get(proto);
            listResult(result);
        }

        public static void retrievePilotByName(ObjectContainer db)
        {
            Pilot proto = new Pilot("Michael Schumacher", 0);
            ObjectSet result = db.get(proto);
            listResult(result);
        }

        public static void retrievePilotByExactPoints(ObjectContainer
db)
        {
            Pilot proto = new Pilot(null, 100);
            ObjectSet result = db.get(proto);
            listResult(result);
        }

        public static void updatePilot(ObjectContainer db)
        {
            ObjectSet result = db.get(new Pilot("Michael Schumacher",
0));
            Pilot found = (Pilot)result.next();
            found.AddPoints(11);
            db.set(found);
            Console.WriteLine("Added 11 points for " + found);
            retrieveAllPilots(db);
        }

        public static void deleteFirstPilotByName(ObjectContainer db)
        {
            ObjectSet result = db.get(new Pilot("Michael Schumacher",
0));
            Pilot found = (Pilot)result.next();
            db.delete(found);
            Console.WriteLine("Deleted " + found);
```

PDF by iText, generated by Doctor, courtesy of db4objects

```
            retrieveAllPilots(db);
        }

        public static void deleteSecondPilotByName(ObjectContainer
db)
        {
            ObjectSet result = db.get(new Pilot("Rubens Barrichello",
0));
            Pilot found = (Pilot)result.next();
            db.delete(found);
            Console.WriteLine("Deleted " + found);
            retrieveAllPilots(db);
        }
    }
}
```

PDF by iText, generated by Doctor, courtesy of db4objects

3. Query API

We have already seen how to retrieve objects from db4o via QBE. While this approach is easy and intuitive, there are situations where it is not sufficient.

- There are queries that simply cannot be expressed with QBE: Retrieve all pilots with more than 100 points, for example.
- Creating a prototype object may have unwanted side effects.
- Default values (e.g. null) may not be accepted by the domain class constructor.
- We may want to query for field default values.

db4o provides a dedicated query API that can be used in those cases.

We need some pilots in our database again to explore it.

```
[storeFirstPilot]

Pilot pilot1 = new Pilot("Michael Schumacher", 100);
    db.set(pilot1);
    Console.WriteLine("Stored " + pilot1);
```

```
[storeSecondPilot]

Pilot pilot2 = new Pilot("Rubens Barrichello", 99);
    db.set(pilot2);
    Console.WriteLine("Stored " + pilot2);
```

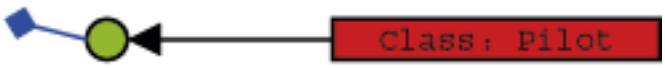
3.1. Simple queries

First, let's see how our familiar QBE queries are expressed within the query API. This is done by retrieving a 'fresh' Query object from the ObjectContainer and adding Constraint instances to it. To find all Pilot instances, we constrain the query with the Pilot class object.

```
[retrieveAllPilots]

Query query = db.query();
    query.constrain(typeof(Pilot));
    ObjectSet result = query.execute();
    listResult(result);
```

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

Our first simple graph looks like this.



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

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

```
[retrievePilotByName]

Query query = db.query();
    query.constrain(typeof(Pilot));
    query.descend("_name").constrain("Michael Schumacher");
    ObjectSet result = query.execute();
    listResult(result);
```

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

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

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

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

```
[retrievePilotByExactPoints]

Query query = db.query();
    query.constrain(typeof(Pilot));
    query.descend("_points").constrain(100);
    ObjectSet result = query.execute();
    listResult(result);
```

3.2. Advanced queries

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

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

```
[retrieveByNegation]

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

```
    query.constrain(typeof(Pilot));
    query.descend("_name").constrain("Michael Schumacher").not();
    ObjectSet result = query.execute();
    listResult(result);
```

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

```
[retrieveByConjunction]

Query query = db.query();
    query.constrain(typeof(Pilot));
    Constraint constr = query.descend("_name")
            .constrain("Michael Schumacher");
    query.descend("_points")
            .constrain(99).and(constr);
    ObjectSet result = query.execute();
    listResult(result);
```

```
[retrieveByDisjunction]

Query query = db.query();
    query.constrain(typeof(Pilot));
    Constraint constr = query.descend("_name")
            .constrain("Michael Schumacher");
    query.descend("_points")
            .constrain(99).or(constr);
    ObjectSet result = query.execute();
    listResult(result);
```

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

```
[retrieveByComparison]

Query query = db.query();
    query.constrain(typeof(Pilot));
    query.descend("_points")
            .constrain(99).greater();
    ObjectSet result = query.execute();
    listResult(result);
```

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

```
[retrieveByDefaultFieldValue]

Pilot somebody = new Pilot("Somebody else", 0);
    db.set(somebody);
    Query query = db.query();
    query.constrain(typeof(Pilot));
    query.descend("_points").constrain(0);
    ObjectSet result = query.execute();
    listResult(result);
    db.delete(somebody);
```

It is also possible to have db4o sort the results.

```
[retrieveSorted]

Query query = db.query();
    query.constrain(typeof(Pilot));
    query.descend("_name").orderAscending();
    ObjectSet result = query.execute();
    listResult(result);
    query.descend("_name").orderDescending();
    result = query.execute();
    listResult(result);
```

All these techniques can be combined arbitrarily, of course. Please try it out.

To prepare for the next chapter, let's clear the database.

```
[clearDatabase]

ObjectSet result = db.get(new Pilot(null, 0));
    while (result.hasNext())
    {
        db.delete(result.next());
    }
```

3.3. Conclusion

Now we know how to build arbitrarily complex queries. But our domain model is not complex at all, consisting of one class only. Let's have a look at the way db4o handles object associations in the next chapter .

3.4. Full source

```
namespace com.db4o.f1.chapter1
{
    using System;
    using com.db4o;
    using com.db4o.query;
    using com.db4o.f1;

    public class QueryExample : Util
    {
        public static void Main(string[] args)
        {
            ObjectContainer db = Db4o.openFile(Util.YapFileName);
            try
            {
                storeFirstPilot(db);
```

```
            storeSecondPilot(db);
            retrieveAllPilots(db);
            retrievePilotByName(db);
            retrievePilotByExactPoints(db);
            retrieveByNegation(db);
            retrieveByConjunction(db);
            retrieveByDisjunction(db);
            retrieveByComparison(db);
            retrieveByDefaultFieldValue(db);
            retrieveSorted(db);
            clearDatabase(db);
        }
        finally
        {
            db.close();
        }
    }

    public static void storeFirstPilot(ObjectContainer db)
    {
        Pilot pilot1 = new Pilot("Michael Schumacher", 100);
        db.set(pilot1);
        Console.WriteLine("Stored " + pilot1);
    }

    public static void storeSecondPilot(ObjectContainer db)
    {
        Pilot pilot2 = new Pilot("Rubens Barrichello", 99);
        db.set(pilot2);
        Console.WriteLine("Stored " + pilot2);
    }

    public static void retrieveAllPilots(ObjectContainer db)
    {
        Query query = db.query();
        query.constrain(typeof(Pilot));
        ObjectSet result = query.execute();
        listResult(result);
    }

    public static void retrievePilotByName(ObjectContainer db)
```

PDF by iText, generated by Doctor, courtesy of db4objects

```
        {
            Query query = db.query();
            query.constrain(typeof(Pilot));
            query.descend("_name").constrain("Michael Schumacher");
            ObjectSet result = query.execute();
            listResult(result);
        }

        public static void retrievePilotByExactPoints(ObjectContainer
db)
        {
            Query query = db.query();
            query.constrain(typeof(Pilot));
            query.descend("_points").constrain(100);
            ObjectSet result = query.execute();
            listResult(result);
        }

        public static void retrieveByNegation(ObjectContainer db)
        {
            Query query = db.query();
            query.constrain(typeof(Pilot));
            query.descend("_name").constrain("Michael
Schumacher").not();
            ObjectSet result = query.execute();
            listResult(result);
        }

        public static void retrieveByConjunction(ObjectContainer db)
        {
            Query query = db.query();
            query.constrain(typeof(Pilot));
            Constraint constr = query.descend("_name")
                    .constrain("Michael Schumacher");
            query.descend("_points")
                    .constrain(99).and(constr);
            ObjectSet result = query.execute();
            listResult(result);
        }

        public static void retrieveByDisjunction(ObjectContainer db)
```

```
        {
            Query query = db.query();
            query.constrain(typeof(Pilot));
            Constraint constr = query.descend("_name")
                    .constrain("Michael Schumacher");
            query.descend("_points")
                    .constrain(99).or(constr);
            ObjectSet result = query.execute();
            listResult(result);
        }

        public static void retrieveByComparison(ObjectContainer db)
        {
            Query query = db.query();
            query.constrain(typeof(Pilot));
            query.descend("_points")
                    .constrain(99).greater();
            ObjectSet result = query.execute();
            listResult(result);
        }

        public static void
retrieveByDefaultFieldValue(ObjectContainer db)
        {
            Pilot somebody = new Pilot("Somebody else", 0);
            db.set(somebody);
            Query query = db.query();
            query.constrain(typeof(Pilot));
            query.descend("_points").constrain(0);
            ObjectSet result = query.execute();
            listResult(result);
            db.delete(somebody);
        }

        public static void retrieveSorted(ObjectContainer db)
        {
            Query query = db.query();
            query.constrain(typeof(Pilot));
            query.descend("_name").orderAscending();
            ObjectSet result = query.execute();
            listResult(result);
```

PDF by iText, generated by Doctor, courtesy of db4objects

```
            query.descend("_name").orderDescending();
            result = query.execute();
            listResult(result);
        }

        public static void clearDatabase(ObjectContainer db)
        {
            ObjectSet result = db.get(new Pilot(null, 0));
            while (result.hasNext())
            {
                db.delete(result.next());
            }
        }
    }
}
```

4. Structured objects

It's time to extend our business domain with another class and see how db4o handles object interrelations. Let's give our pilot a vehicle.

```
namespace com.db4o.f1.chapter2
{
    public class Car
    {
        string _model;
        Pilot _pilot;

        public Car(string model)
        {
            _model = model;
            _pilot = null;
        }

        public Pilot Pilot
        {
            get
            {
                return _pilot;
            }

            set
            {
                _pilot = value;
            }
        }

        public string Model
        {
            get
            {
                return _model;
            }
        }
```

```
        override public string ToString()
        {
            return _model + "[" + _pilot + "]";
        }
    }
}
```

4.1. Storing structured objects

To store a car with its pilot, we just call set() on our top level object, the car. The pilot will be stored implicitly.

```
[storeFirstCar]

Car car1 = new Car("Ferrari");
    Pilot pilot1 = new Pilot("Michael Schumacher", 100);
    car1.Pilot = pilot1;
    db.set(car1);
```

Of course, we need some competition here. This time we explicitly store the pilot before entering the car - this makes no difference.

```
[storeSecondCar]

Pilot pilot2 = new Pilot("Rubens Barrichello", 99);
    db.set(pilot2);
    Car car2 = new Car("BMW");
    car2.Pilot = pilot2;
    db.set(car2);
```

4.2. Retrieving structured objects

4.2.1. QBE

To retrieve all cars, we simply provide a 'blank' prototype.

```
[retrieveAllCarsQBE]

Car proto = new Car(null);
    ObjectSet result = db.get(proto);
    listResult(result);
```

We can also query for all pilots, of course.

```
[retrieveAllPilotsQBE]

Pilot proto = new Pilot(null, 0);
    ObjectSet result = db.get(proto);
    listResult(result);
```

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

```
[retrieveCarByPilotQBE]

Pilot pilotproto = new Pilot("Rubens Barrichello",0);
    Car carproto = new Car(null);
    carproto.Pilot = pilotproto;
    ObjectSet result = db.get(carproto);
    listResult(result);
```

What about retrieving a pilot by car? We simply don't need that - if we already know the car, we can simply ask it for its pilot directly.

4.2.2. Query API

To query for a car given its pilot's name we have to descend one level deeper in our query.

```
[retrieveCarByPilotNameQuery]

Query query = db.query();
    query.constrain(typeof(Car));
    query.descend("_pilot").descend("_name")
        .constrain("Rubens Barrichello");
    ObjectSet result = query.execute();
    listResult(result);
```

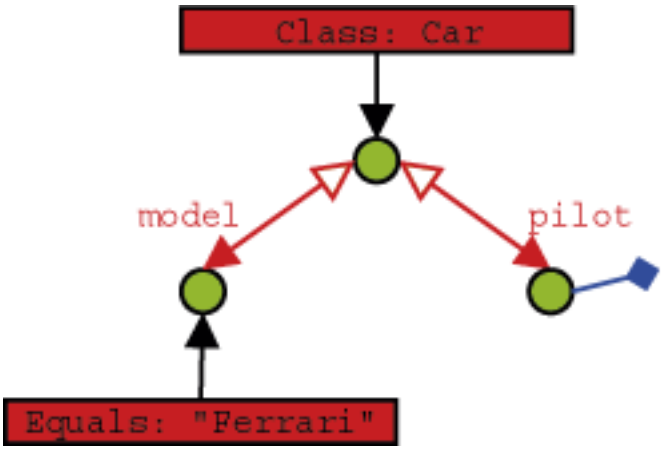
We can also constrain the pilot field with a prototype to achieve the same result.

```
[retrieveCarByPilotProtoQuery]

Query query = db.query();
    query.constrain(typeof(Car));
    Pilot proto = new Pilot("Rubens Barrichello", 0);
    query.descend("_pilot").constrain(proto);
    ObjectSet result = query.execute();
    listResult(result);
```

We have seen that descending into a query provides us with another query. You may also have noticed that the associations between query nodes look somehow bidirectional in our diagrams. Let's move upstream.

```
[retrievePilotByCarModelQuery]

Query carquery=db.query();
    carquery.constrain(typeof(Car));
```

```
    carquery.descend("_model").constrain("Ferrari");
    Query pilotquery=carquery.descend("_pilot");
    ObjectSet result=pilotquery.execute();
    listResult(result);
```



4.3. Updating structured objects

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

```
[updateCar]

ObjectSet result = db.get(new Car("Ferrari"));
    Car found = (Car)result.next();
    found.Pilot = new Pilot("Somebody else", 0);
    db.set(found);
    result = db.get(new Car("Ferrari"));
    listResult(result);
```

Let's modify the pilot, too.

```
[updatePilotSingleSession]
```

```
ObjectSet result = db.get(new Car("Ferrari"));
    Car found = (Car)result.next();
    found.Pilot.AddPoints(1);
    db.set(found);
    result = db.get(new Car("Ferrari"));
    listResult(result);
```

Nice and easy, isn't it? But wait, there's something evil lurking right behind the corner. Let's see what happens if we split this task in two separate db4o sessions: In the first we modify our pilot and update his car, in the second we query for the car again.

```
[updatePilotSeparateSessionsPart1]

ObjectSet result = db.get(new Car("Ferrari"));
    Car found = (Car)result.next();
    found.Pilot.AddPoints(1);
    db.set(found);
```

```
[updatePilotSeparateSessionsPart2]

ObjectSet result = db.get(new Car("Ferrari"));
    listResult(result);
```

Looks like we're in trouble. What's happening here and what can we do to fix it?

4.3.1. Update depth

Imagine a complex object with many members that have many members themselves. When updating this object, db4o would have to update all its children, grandchildren, etc. This poses a severe performance penalty and will not be necessary in most cases - sometimes, however, it will.

To be able to handle this dilemma as flexible as possible, db4o introduces the concept of update depth

to control how deep an object's member tree will be traversed on update. The default update depth for all objects is 1, meaning that only primitive and String members will be updated, but changes in object members will not be reflected.

db4o provides means to control update depth with very fine granularity. For our current problem we'll advise db4o to update the full graph for Car objects by setting cascadeOnUpdate() for this class accordingly.

```
[updatePilotSeparateSessionsImprovedPart1]

Db4o.configure().objectClass(typeof(Car))
        .cascadeOnUpdate(true);
```

```
[updatePilotSeparateSessionsImprovedPart2]

ObjectSet result = db.get(new Car("Ferrari"));
    Car found = (Car)result.next();
    found.Pilot.AddPoints(1);
    db.set(found);
```

```
[updatePilotSeparateSessionsImprovedPart3]

ObjectSet result = db.get(new Car("Ferrari"));
    listResult(result);
```

This looks much better.

Note that container configuration must be set before the container is opened.

We'll cover update depth as well as other issues with complex object graphs and the respective db4o configuration options in more detail in a later chapter.

4.4. Deleting structured objects

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

```
[deleteFlat]

ObjectSet result = db.get(new Car("Ferrari"));
    Car found = (Car)result.next();
    db.delete(found);
    result = db.get(new Car(null));
    listResult(result);
```

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

```
[retrieveAllPilotsQBE]

Pilot proto = new Pilot(null, 0);
    ObjectSet result = db.get(proto);
    listResult(result);
```

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

4.4.1. Recursive deletion

You may already suspect that the problem of recursive deletion (and perhaps its solution, too) is quite similar to our little update problem, and you're right. Let's configure db4o to delete a car's pilot, too, when the car is deleted.

```
[deleteDeepPart1]

Db4o.configure().objectClass(typeof(Car))
```

```
        .cascadeOnDelete(true);
```

```
[deleteDeepPart2]

ObjectSet result = db.get(new Car("BMW"));
    Car found = (Car)result.next();
    db.delete(found);
    result = db.get(new Car(null));
    listResult(result);
```

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

Let's have a look at our pilots again.

```
[retrieveAllPilotsQBE]

Pilot proto = new Pilot(null, 0);
    ObjectSet result = db.get(proto);
    listResult(result);
```

4.4.2. Recursive deletion revisited

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

```
[deleteDeepRevisited]

ObjectSet result = db.get(new Pilot("Michael Schumacher", 0));
    Pilot pilot = (Pilot)result.next();
    Car car1 = new Car("Ferrari");
    Car car2 = new Car("BMW");
    car1.Pilot = pilot;
    car2.Pilot = pilot;
```

```
    db.set(car1);
    db.set(car2);
    db.delete(car2);
    result = db.get(new Car(null));
    listResult(result);
```

```
[retrieveAllPilotsQBE]

Pilot proto = new Pilot(null, 0);
    ObjectSet result = db.get(proto);
    listResult(result);
```

Houston, we have a problem - and there's no simple solution at hand. Currently db4o does **not** check whether objects to be deleted are referenced anywhere else, so please be very careful when activating this feature.

Let's clear our database for the next chapter.

```
[deleteAll]

ObjectSet cars=db.get(new Car(null));
    while(cars.hasNext()) {
        db.delete(cars.next());
    }
    ObjectSet pilots=db.get(new Pilot(null,0));
    while(pilots.hasNext()) {
        db.delete(pilots.next());
    }
```

4.5. Conclusion

So much for object associations: We can hook into a root object and climb down its reference graph to

specify queries. But what about multi-valued objects like arrays and collections? We will cover this in the next chapter .

4.6. Full source

```
namespace com.db4o.f1.chapter2
{
    using System;
    using System.IO;
    using com.db4o;
    using com.db4o.f1;
    using com.db4o.query;

    public class StructuredExample : Util
    {
        public static void Main(String[] args)
        {
            File.Delete(Util.YapFileName);

            ObjectContainer db = Db4o.openFile(Util.YapFileName);
            try
            {
                storeFirstCar(db);
                storeSecondCar(db);
                retrieveAllCarsQBE(db);
                retrieveAllPilotsQBE(db);
                retrieveCarByPilotQBE(db);
                retrieveCarByPilotNameQuery(db);
                retrieveCarByPilotProtoQuery(db);
             retrievePilotByCarModelQuery(db);
                updateCar(db);
                updatePilotSingleSession(db);
                updatePilotSeparateSessionsPart1(db);
                db.close();
                db=Db4o.openFile(Util.YapFileName);
                updatePilotSeparateSessionsPart2(db);
                db.close();
                updatePilotSeparateSessionsImprovedPart1(db);
                db=Db4o.openFile(Util.YapFileName);
```

```
            updatePilotSeparateSessionsImprovedPart2(db);
            db.close();
            db=Db4o.openFile(Util.YapFileName);
            updatePilotSeparateSessionsImprovedPart3(db);
            deleteFlat(db);
            db.close();
            deleteDeepPart1(db);
            db=Db4o.openFile(Util.YapFileName);
            deleteDeepPart2(db);
            deleteDeepRevisited(db);
        }
        finally
        {
            db.close();
        }
    }

    public static void storeFirstCar(ObjectContainer db)
    {
        Car car1 = new Car("Ferrari");
        Pilot pilot1 = new Pilot("Michael Schumacher", 100);
        car1.Pilot = pilot1;
        db.set(car1);
    }

    public static void storeSecondCar(ObjectContainer db)
    {
        Pilot pilot2 = new Pilot("Rubens Barrichello", 99);
        db.set(pilot2);
        Car car2 = new Car("BMW");
        car2.Pilot = pilot2;
        db.set(car2);
    }

    public static void retrieveAllCarsQBE(ObjectContainer db)
    {
        Car proto = new Car(null);
        ObjectSet result = db.get(proto);
        listResult(result);
    }
```

PDF by iText, generated by Doctor, courtesy of db4objects

```
        public static void retrieveAllPilotsQBE(ObjectContainer db)
        {
            Pilot proto = new Pilot(null, 0);
            ObjectSet result = db.get(proto);
            listResult(result);
        }

        public static void retrieveCarByPilotQBE(ObjectContainer db)
        {
            Pilot pilotproto = new Pilot("Rubens Barrichello",0);
            Car carproto = new Car(null);
            carproto.Pilot = pilotproto;
            ObjectSet result = db.get(carproto);
            listResult(result);
        }

        public static void
retrieveCarByPilotNameQuery(ObjectContainer db)
        {
            Query query = db.query();
            query.constrain(typeof(Car));
            query.descend("_pilot").descend("_name")
                .constrain("Rubens Barrichello");
            ObjectSet result = query.execute();
            listResult(result);
        }

        public static void
retrieveCarByPilotProtoQuery(ObjectContainer db)
        {
            Query query = db.query();
            query.constrain(typeof(Car));
            Pilot proto = new Pilot("Rubens Barrichello", 0);
            query.descend("_pilot").constrain(proto);
            ObjectSet result = query.execute();
            listResult(result);
        }

        public static void
retrievePilotByCarModelQuery(ObjectContainer db)
        {
```

```
         Query carquery=db.query();
         carquery.constrain(typeof(Car));
         carquery.descend("_model").constrain("Ferrari");
         Query pilotquery=carquery.descend("_pilot");
         ObjectSet result=pilotquery.execute();
         listResult(result);
        }

        public static void updateCar(ObjectContainer db)
        {
            ObjectSet result = db.get(new Car("Ferrari"));
            Car found = (Car)result.next();
            found.Pilot = new Pilot("Somebody else", 0);
            db.set(found);
            result = db.get(new Car("Ferrari"));
            listResult(result);
        }

        public static void updatePilotSingleSession(ObjectContainer
db)
        {
            ObjectSet result = db.get(new Car("Ferrari"));
            Car found = (Car)result.next();
            found.Pilot.AddPoints(1);
            db.set(found);
            result = db.get(new Car("Ferrari"));
            listResult(result);
        }

        public static void
updatePilotSeparateSessionsPart1(ObjectContainer db)
        {
            ObjectSet result = db.get(new Car("Ferrari"));
            Car found = (Car)result.next();
            found.Pilot.AddPoints(1);
            db.set(found);
        }

        public static void
updatePilotSeparateSessionsPart2(ObjectContainer db)
        {
```

PDF by iText, generated by Doctor, courtesy of db4objects

```
            ObjectSet result = db.get(new Car("Ferrari"));
            listResult(result);
        }

        public static void
updatePilotSeparateSessionsImprovedPart1(ObjectContainer db)
        {
            Db4o.configure().objectClass(typeof(Car))
                .cascadeOnUpdate(true);
        }

        public static void
updatePilotSeparateSessionsImprovedPart2(ObjectContainer db)
        {
            ObjectSet result = db.get(new Car("Ferrari"));
            Car found = (Car)result.next();
            found.Pilot.AddPoints(1);
            db.set(found);
        }

        public static void
updatePilotSeparateSessionsImprovedPart3(ObjectContainer db)
        {
            ObjectSet result = db.get(new Car("Ferrari"));
            listResult(result);
        }

        public static void deleteFlat(ObjectContainer db)
        {
            ObjectSet result = db.get(new Car("Ferrari"));
            Car found = (Car)result.next();
            db.delete(found);
            result = db.get(new Car(null));
            listResult(result);
        }

        public static void deleteDeepPart1(ObjectContainer db)
        {
            Db4o.configure().objectClass(typeof(Car))
                .cascadeOnDelete(true);
        }
```

PDF by iText, generated by Doctor, courtesy of db4objects

```
        public static void deleteDeepPart2(ObjectContainer db)
        {
            ObjectSet result = db.get(new Car("BMW"));
            Car found = (Car)result.next();
            db.delete(found);
            result = db.get(new Car(null));
            listResult(result);
        }

        public static void deleteDeepRevisited(ObjectContainer db)
        {
            ObjectSet result = db.get(new Pilot("Michael Schumacher",
0));
            Pilot pilot = (Pilot)result.next();
            Car car1 = new Car("Ferrari");
            Car car2 = new Car("BMW");
            car1.Pilot = pilot;
            car2.Pilot = pilot;
            db.set(car1);
            db.set(car2);
            db.delete(car2);
            result = db.get(new Car(null));
            listResult(result);
        }

        public static void deleteAll(ObjectContainer db) {
         ObjectSet cars=db.get(new Car(null));
         while(cars.hasNext()) {
             db.delete(cars.next());
         }
         ObjectSet pilots=db.get(new Pilot(null,0));
         while(pilots.hasNext()) {
             db.delete(pilots.next());
         }
     }
    }
}
```

PDF by iText, generated by Doctor, courtesy of db4objects

PDF by iText, generated by Doctor, courtesy of db4objects

5. Collections and Arrays

We will slowly move towards real-time data processing now by installing sensors to our car and collecting their output.

```
namespace com.db4o.f1.chapter3
{
    using System;
    using System.Text;

    public class SensorReadout
    {
        double[] _values;
        DateTime _time;
        Car _car;

        public SensorReadout(double[] values, DateTime time, Car car)
        {
            _values = values;
            _time = time;
            _car = car;
        }

        public Car Car
        {
            get
            {
                return _car;
            }
        }

        public DateTime Time
        {
            get
            {
                return _time;
            }
        }
```

```
        public int NumValues
        {
            get
            {
                return _values.Length;
            }
        }

        public double GetValue(int idx)
        {
            return _values[idx];
        }

        override public string ToString()
        {
            StringBuilder builder = new StringBuilder();
            builder.Append(_car);
            builder.Append(" : ");
            builder.Append(_time.TimeOfDay);
            builder.Append(" : ");
            for (int i=0; i<_values.Length; ++i)
            {
                if (i > 0)
                {
                    builder.Append(", ");
                }
                builder.Append(_values[i]);
            }
            return builder.ToString();
        }
    }
}
```

A car may produce its current sensor readout when requested and keep a list of readouts collected during a race.

```
namespace com.db4o.f1.chapter3
{
    using System;
    using System.Collections;

    public class Car
    {
        string _model;
        Pilot _pilot;
        IList _history;

        public Car(string model) : this(model, new ArrayList())
        {
        }

        public Car(string model, IList history)
        {
            _model = model;
            _pilot = null;
            _history = history;
        }

        public Pilot Pilot
        {
            get
            {
                return _pilot;
            }

            set
            {
                _pilot = value;
            }
        }

        public string Model
        {
            get
            {
                return _model;
```

PDF by iText, generated by Doctor, courtesy of db4objects

```
            }
        }

        public SensorReadout[] GetHistory()
        {
            SensorReadout[] history = new
SensorReadout[_history.Count];
            _history.CopyTo(history, 0);
            return history;
        }

        public void Snapshot()
        {
            _history.Add(new SensorReadout(Poll(), DateTime.Now,
this));
        }

        protected double[]  Poll()
        {
            int factor = _history.Count + 1;
            return new double[] { 0.1d*factor, 0.2d*factor,
0.3d*factor };
        }

        override public string ToString()
        {
            return _model + "[" + _pilot + "]/" + _history.Count;
        }
    }
}
```

We will constrain ourselves to rather static data at the moment and add flexibility during the next chapters.

5.1. Storing

This should be familiar by now.

```
[storeFirstCar]

Car car1 = new Car("Ferrari");
    Pilot pilot1 = new Pilot("Michael Schumacher", 100);
    car1.Pilot = pilot1;
    db.set(car1);
```

The second car will take two snapshots immediately at startup.

```
[storeSecondCar]

Pilot pilot2 = new Pilot("Rubens Barrichello", 99);
    Car car2 = new Car("BMW");
    car2.Pilot = pilot2;
    car2.Snapshot();
    car2.Snapshot();
    db.set(car2);
```

5.2. Retrieving

5.2.1. QBE

First let us verify that we indeed have taken snapshots.

```
[retrieveAllSensorReadouts]

SensorReadout proto = new SensorReadout(null, DateTime.MinValue,
null);
    ObjectSet result = db.get(proto);
    listResult(result);
```

As a prototype for an array, we provide an array of the same type, containing only the values we expect the result to contain.

```
[retrieveSensorReadoutQBE]

SensorReadout proto = new SensorReadout(new double[] { 0.3, 0.1 },
DateTime.MinValue, null);
    ObjectSet result = db.get(proto);
    listResult(result);
```

Note that the actual position of the given elements in the prototype array is irrelevant.

To retrieve a car by its stored sensor readouts, we install a history containing the sought-after values.

```
[retrieveCarQBE]

SensorReadout protoreadout = new SensorReadout(new double[] { 0.6,
0.2 }, DateTime.MinValue, null);
    IList protohistory = new ArrayList();
    protohistory.Add(protoreadout);
    Car protocar = new Car(null, protohistory);
    ObjectSet result = db.get(protocar);
    listResult(result);
```

We can also query for the collections themselves, since they are first class objects.

```
[retrieveCollections]

ObjectSet result = db.get(new ArrayList());
    listResult(result);
```

This doesn't work with arrays, though.

```
[retrieveArrays]

ObjectSet result = db.get(new double[] { 0.6, 0.4 });
    listResult(result);
```

5.2.2. Query API

Handling of arrays and collections is analogous to the previous example.

```
[retrieveSensorReadoutQuery]

Query query = db.query();
    query.constrain(typeof(SensorReadout));
    Query valuequery = query.descend("_values");
    valuequery.constrain(0.3);
    valuequery.constrain(0.1);
    ObjectSet result = query.execute();
    listResult(result);
```

```
[retrieveCarQuery]

Query query = db.query();
    query.constrain(typeof(Car));
    Query historyquery = query.descend("_history");
    historyquery.constrain(typeof(SensorReadout));
    Query valuequery = historyquery.descend("_values");
    valuequery.constrain(0.3);
    valuequery.constrain(0.1);
    ObjectSet result = query.execute();
    listResult(result);
```

5.3. Updating and deleting

This should be familiar, we just have to remember to take care of the update depth.

```
[updateCarPart1]

Db4o.configure().objectClass(typeof(Car)).cascadeOnUpdate(true);
```

```
[updateCarPart2]

ObjectSet result = db.get(new Car("BMW", null));
    Car car = (Car)result.next();
    car.Snapshot();
    db.set(car);
    retrieveAllSensorReadouts(db);
```

There's nothing special about deleting arrays and collections, too.

Deleting an object from a collection is an update, too, of course.

```
[updateCollection]

Query query = db.query();
    query.constrain(typeof(Car));
    ObjectSet result = query.descend("_history").execute();
    IList coll = (IList)result.next();
    coll.RemoveAt(0);
    db.set(coll);
    Car proto = new Car(null, null);
    result = db.get(proto);
    while (result.hasNext())
    {
```

```
        Car car = (Car)result.next();
        foreach (object readout in car.GetHistory())
        {
            Console.WriteLine(readout);
        }
    }
```

(This example also shows that with db4o it is quite easy to access object internals we were never meant to see. Please keep this always in mind and be careful.)

We will delete all cars from the database again to prepare for the next chapter.

```
[deleteAllPart1]

Db4o.configure().objectClass(typeof(Car)).cascadeOnDelete(true);
```

```
[deleteAllPart2]

ObjectSet result = db.get(new Car(null, null));
    while (result.hasNext())
    {
        db.delete(result.next());
    }
    ObjectSet readouts = db.get(new SensorReadout(null,
DateTime.MinValue, null));
    while(readouts.hasNext())
    {
        db.delete(readouts.next());
    }
```

5.4. db4o custom collections

db4o also provides customized collection implementations, tweaked for use with db4o. We will get to that in a later chapter when we have finished our first walkthrough.

5.5. Conclusion

Ok, collections are just objects. But why did we have to specify the concrete ArrayList type all the way? Was that necessary? How does db4o handle inheritance? We will cover that in the next chapter .

5.6. Full source

```
namespace com.db4o.f1.chapter3
{
    using System;
    using System.Collections;
    using System.IO;
    using com.db4o;
    using com.db4o.query;

    public class CollectionsExample : Util
    {
        public static void Main(string[] args)
        {
            File.Delete(Util.YapFileName);
            ObjectContainer db = Db4o.openFile(Util.YapFileName);
            try
            {
                storeFirstCar(db);
                storeSecondCar(db);
                retrieveAllSensorReadouts(db);
                retrieveSensorReadoutQBE(db);
                retrieveCarQBE(db);
                retrieveCollections(db);
                retrieveArrays(db);
                retrieveSensorReadoutQuery(db);
                retrieveCarQuery(db);
                db.close();
                updateCarPart1();
                db = Db4o.openFile(Util.YapFileName);
                updateCarPart2(db);
```

```
            updateCollection(db);
            db.close();
            deleteAllPart1();
            db=Db4o.openFile(Util.YapFileName);
            deleteAllPart2(db);
            retrieveAllSensorReadouts(db);
        }
        finally
        {
            db.close();
        }
    }

    public static void storeFirstCar(ObjectContainer db)
    {
        Car car1 = new Car("Ferrari");
        Pilot pilot1 = new Pilot("Michael Schumacher", 100);
        car1.Pilot = pilot1;
        db.set(car1);
    }

    public static void storeSecondCar(ObjectContainer db)
    {
        Pilot pilot2 = new Pilot("Rubens Barrichello", 99);
        Car car2 = new Car("BMW");
        car2.Pilot = pilot2;
        car2.Snapshot();
        car2.Snapshot();
        db.set(car2);
    }

    public static void retrieveAllSensorReadouts(ObjectContainer
db)
    {
        SensorReadout proto = new SensorReadout(null,
DateTime.MinValue, null);
        ObjectSet result = db.get(proto);
        listResult(result);
    }

    public static void retrieveSensorReadoutQBE(ObjectContainer
```

PDF by iText, generated by Doctor, courtesy of db4objects

```
db)
        {
            SensorReadout proto = new SensorReadout(new double[] {
0.3, 0.1 }, DateTime.MinValue, null);
            ObjectSet result = db.get(proto);
            listResult(result);
        }

        public static void retrieveCarQBE(ObjectContainer db)
        {
            SensorReadout protoreadout = new SensorReadout(new
double[] { 0.6, 0.2 }, DateTime.MinValue, null);
            IList protohistory = new ArrayList();
            protohistory.Add(protoreadout);
            Car protocar = new Car(null, protohistory);
            ObjectSet result = db.get(protocar);
            listResult(result);
        }

        public static void retrieveCollections(ObjectContainer db)
        {
            ObjectSet result = db.get(new ArrayList());
            listResult(result);
        }

        public static void retrieveArrays(ObjectContainer db)
        {
            ObjectSet result = db.get(new double[] { 0.6, 0.4 });
            listResult(result);
        }

        public static void retrieveSensorReadoutQuery(ObjectContainer
db)
        {
            Query query = db.query();
            query.constrain(typeof(SensorReadout));
            Query valuequery = query.descend("_values");
            valuequery.constrain(0.3);
            valuequery.constrain(0.1);
            ObjectSet result = query.execute();
            listResult(result);
```

PDF by iText, generated by Doctor, courtesy of db4objects

```
        }

        public static void retrieveCarQuery(ObjectContainer db)
        {
            Query query = db.query();
            query.constrain(typeof(Car));
            Query historyquery = query.descend("_history");
            historyquery.constrain(typeof(SensorReadout));
            Query valuequery = historyquery.descend("_values");
            valuequery.constrain(0.3);
            valuequery.constrain(0.1);
            ObjectSet result = query.execute();
            listResult(result);
        }

        public static void updateCarPart1()
        {
Db4o.configure().objectClass(typeof(Car)).cascadeOnUpdate(true);
   }

        public static void updateCarPart2(ObjectContainer db)
        {
            ObjectSet result = db.get(new Car("BMW", null));
            Car car = (Car)result.next();
            car.Snapshot();
            db.set(car);
            retrieveAllSensorReadouts(db);
        }

        public static void updateCollection(ObjectContainer db)
        {
            Query query = db.query();
            query.constrain(typeof(Car));
            ObjectSet result = query.descend("_history").execute();
            IList coll = (IList)result.next();
            coll.RemoveAt(0);
            db.set(coll);
            Car proto = new Car(null, null);
            result = db.get(proto);
            while (result.hasNext())
            {
```

PDF by iText, generated by Doctor, courtesy of db4objects

```
                Car car = (Car)result.next();
                foreach (object readout in car.GetHistory())
                {
                    Console.WriteLine(readout);
                }
            }
        }

        public static void deleteAllPart1()
        {
Db4o.configure().objectClass(typeof(Car)).cascadeOnDelete(true);
   }

        public static void deleteAllPart2(ObjectContainer db)
        {
            ObjectSet result = db.get(new Car(null, null));
            while (result.hasNext())
            {
                db.delete(result.next());
            }
            ObjectSet readouts = db.get(new SensorReadout(null,
DateTime.MinValue, null));
            while(readouts.hasNext())
            {
                db.delete(readouts.next());
            }
        }
    }
}
```

PDF by iText, generated by Doctor, courtesy of db4objects

6. Inheritance

So far we have always been working with the concrete (i.e. most specific type of an object. What about subclassing and interfaces?

To explore this, we will differentiate between different kinds of sensors.

```
namespace com.db4o.f1.chapter4
{
    using System;

    public class SensorReadout
    {
        DateTime _time;
        Car _car;
        string _description;

        public SensorReadout(DateTime time, Car car, string
description)
        {
            _time = time;
            _car = car;
            _description = description;
        }

        public Car Car
        {
            get
            {
                return _car;
            }
        }

        public DateTime Time
        {
            get
            {
                return _time;
```

```
            }
        }

        public string Description
        {
            get
            {
                return _description;
            }
        }

        override public string ToString()
        {
            return _car + ":" + _time + " : " + _description;
        }
    }
}
```

```
namespace com.db4o.f1.chapter4
{
    using System;

    public class TemperatureSensorReadout : SensorReadout
    {
        double _temperature;

        public TemperatureSensorReadout(DateTime time, Car car,
string description, double temperature)
            : base(time, car, description)
        {
            _temperature = temperature;
        }

        public double Temperature
        {
            get
            {
```

```
                return _temperature;
            }
        }

        override public string ToString()
        {
            return base.ToString() + " temp: " + _temperature;
        }
    }
}
```

```
namespace com.db4o.f1.chapter4
{
    using System;

    public class PressureSensorReadout : SensorReadout
    {
        double _pressure;

        public PressureSensorReadout(DateTime time, Car car, string
description, double pressure)
            : base(time, car, description)
        {
            _pressure = pressure;
        }

        public double Pressure
        {
            get
            {
                return _pressure;
            }
        }

        override public string ToString()
        {
            return base.ToString() + " pressure : " + _pressure;
```

```
        }
    }
}
```

Our car's snapshot mechanism is changed accordingly.

```
namespace com.db4o.f1.chapter4
{
    using System;
    using System.Collections;

    public class Car
    {
        string _model;
        Pilot _pilot;
        IList _history;

        public Car(string model)
        {
            _model = model;
            _pilot = null;
            _history = new ArrayList();
        }

        public Pilot Pilot
        {
            get
            {
                return _pilot;
            }

            set
            {
                _pilot = value;
            }
        }
```

PDF by iText, generated by Doctor, courtesy of db4objects

```
        public string Model
        {
            get
            {
                return _model;
            }
        }

        public SensorReadout[] GetHistory()
        {
            SensorReadout[] history = new
SensorReadout[_history.Count];
            _history.CopyTo(history, 0);
            return history;
        }

        public void Snapshot()
        {
            _history.Add(new TemperatureSensorReadout(DateTime.Now,
this, "oil", PollOilTemperature()));
            _history.Add(new TemperatureSensorReadout(DateTime.Now,
this, "water", PollWaterTemperature()));
            _history.Add(new PressureSensorReadout(DateTime.Now,
this, "oil", PollOilPressure()));
        }

        protected double PollOilTemperature()
        {
            return 0.1*_history.Count;
        }

        protected double PollWaterTemperature()
        {
            return 0.2*_history.Count;
        }

        protected double PollOilPressure()
        {
            return 0.3*_history.Count;
        }
```

PDF by iText, generated by Doctor, courtesy of db4objects

```
        override public string ToString()
        {
            return _model + "[" + _pilot + "]/" + _history.Count;
        }
    }
}
```

6.1. Storing

Our setup code has not changed at all, just the internal workings of a snapshot.

```
[storeFirstCar]

Car car1 = new Car("Ferrari");
    Pilot pilot1 = new Pilot("Michael Schumacher", 100);
    car1.Pilot = pilot1;
    db.set(car1);
```

```
[storeSecondCar]

Pilot pilot2 = new Pilot("Rubens Barrichello", 99);
    Car car2 = new Car("BMW");
    car2.Pilot = pilot2;
    car2.Snapshot();
    car2.Snapshot();
    db.set(car2);
```

6.2. Retrieving

db4o will provide us with all objects of the given type. To collect all instances of a given class, no matter whether they are subclass members or direct instances, we just provide a corresponding prototype.

```
[retrieveTemperatureReadoutsQBE]

SensorReadout proto = new TemperatureSensorReadout(DateTime.MinValue,
null, null, 0.0);
    ObjectSet result = db.get(proto);
    listResult(result);
```

```
[retrieveAllSensorReadoutsQBE]

SensorReadout proto = new SensorReadout(DateTime.MinValue, null,
null);
    ObjectSet result = db.get(proto);
    listResult(result);
```

This is one more situation where QBE might not be applicable: What if the given type is an interface or an abstract class? Well, there's a little DWIM trick to the rescue: Class objects receive special handling with QBE.

```
[retrieveAllSensorReadoutsQBEAlternative]

ObjectSet result = db.get(typeof(SensorReadout));
    listResult(result);
```

And of course there's our query API to the rescue.

```
[retrieveAllSensorReadoutsQuery]

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

```
    query.constrain(typeof(SensorReadout));
    ObjectSet result = query.execute();
    listResult(result);
```

This procedure applies to all first class objects. We can simply query for all objects present in the database, for example.

```
[retrieveAllObjects]

ObjectSet result = db.get(new object());
    listResult(result);
```

6.3. Updating and deleting

is just the same for all objects, no matter where they are situated in the inheritance tree.

Just like we retrieved all objects from the database above, we can delete all stored objects to prepare for the next chapter.

```
[deleteAllObjects]

ObjectSet result=db.get(new object());
    while (result.hasNext())
    {
        db.delete(result.next());
    }
```

6.4. Conclusion

Now we have covered all basic OO features and the way they are handled by db4o. We will complete the first part of our db4o walkthrough in the next chapter by looking at deep object graphs, including recursive structures.

6.5. Full source

```
namespace com.db4o.f1.chapter4
{
    using System;
    using System.IO;
    using com.db4o;
    using com.db4o.f1;
    using com.db4o.query;

    public class InheritanceExample : Util
    {
        public static void main(string[] args)
        {
            File.Delete(Util.YapFileName);
            ObjectContainer db = Db4o.openFile(Util.YapFileName);
            try
            {
                storeFirstCar(db);
                storeSecondCar(db);
                retrieveTemperatureReadoutsQBE(db);
                retrieveAllSensorReadoutsQBE(db);
                retrieveAllSensorReadoutsQBEAlternative(db);
                retrieveAllSensorReadoutsQuery(db);
                retrieveAllObjects(db);
                deleteAllObjects(db);
            }
            finally
            {
                db.close();
            }
        }

        public static void storeFirstCar(ObjectContainer db)
        {
            Car car1 = new Car("Ferrari");
            Pilot pilot1 = new Pilot("Michael Schumacher", 100);
            car1.Pilot = pilot1;
```

```
            db.set(car1);
        }

        public static void storeSecondCar(ObjectContainer db)
        {
            Pilot pilot2 = new Pilot("Rubens Barrichello", 99);
            Car car2 = new Car("BMW");
            car2.Pilot = pilot2;
            car2.Snapshot();
            car2.Snapshot();
            db.set(car2);
        }

        public static void
retrieveAllSensorReadoutsQBE(ObjectContainer db)
        {
            SensorReadout proto = new
SensorReadout(DateTime.MinValue, null, null);
            ObjectSet result = db.get(proto);
            listResult(result);
        }

        public static void
retrieveTemperatureReadoutsQBE(ObjectContainer db)
        {
            SensorReadout proto = new
TemperatureSensorReadout(DateTime.MinValue, null, null, 0.0);
            ObjectSet result = db.get(proto);
            listResult(result);
        }

        public static void
retrieveAllSensorReadoutsQBEAlternative(ObjectContainer db)
        {
            ObjectSet result = db.get(typeof(SensorReadout));
            listResult(result);
        }

        public static void
retrieveAllSensorReadoutsQuery(ObjectContainer db)
        {
```

```
            Query query = db.query();
            query.constrain(typeof(SensorReadout));
            ObjectSet result = query.execute();
            listResult(result);
        }

        public static void retrieveAllObjects(ObjectContainer db)
        {
            ObjectSet result = db.get(new object());
            listResult(result);
        }

        public static void deleteAllObjects(ObjectContainer db)
        {
            ObjectSet result=db.get(new object());
            while (result.hasNext())
            {
                db.delete(result.next());
            }
        }
    }
}
```

PDF by iText, generated by Doctor, courtesy of db4objects

7. Deep graphs

We have already seen how db4o handles object associations, but our running example is still quite flat and simple, compared to real-world domain models. In particular we haven't seen how db4o behaves in the presence of recursive structures. We will emulate such a structure by replacing our history list with a linked list implicitely provided by the SensorReadout class.

```
namespace com.db4o.f1.chapter5
{
    using System;

    public abstract class SensorReadout
    {
        DateTime _time;
        Car _car;
        string _description;
        SensorReadout _next;

        protected SensorReadout(DateTime time, Car car, string
description)
        {
            _time = time;
            _car = car;
            _description = description;
            _next = null;
        }

        public Car Car
        {
            get
            {
                return _car;
            }
        }

        public DateTime Time
        {
            get
```

```
            {
                return _time;
            }
        }

        public SensorReadout Next
        {
            get
            {
                return _next;
            }
        }

        public void Append(SensorReadout sensorReadout)
        {
            if (_next == null)
            {
                _next = sensorReadout;
            }
            else
            {
                _next.Append(sensorReadout);
            }
        }

        public int CountElements()
        {
            return (_next == null ? 1 : _next.CountElements() + 1);
        }

        override public string ToString()
        {
            return _car + " : " + _time + " : " + _description;
        }
    }
}
```

Our car only maintains an association to a 'head' sensor readout now.

```
namespace com.db4o.f1.chapter5
{
    using System;

    public class Car
    {
        string _model;
        Pilot _pilot;
        SensorReadout _history;

        public Car(string model)
        {
            _model = model;
            _pilot = null;
            _history = null;
        }

        public Pilot Pilot
        {
            get
            {
                return _pilot;
            }

            set
            {
                _pilot = value;
            }
        }

        public string Model
        {
            get
            {
                return _model;
            }
        }

        public SensorReadout GetHistory()
```

PDF by iText, generated by Doctor, courtesy of db4objects

```
        {
            return _history;
        }

        public void Snapshot()
        {
            AppendToHistory(new TemperatureSensorReadout(
                DateTime.Now, this, "oil", PollOilTemperature()));
            AppendToHistory(new TemperatureSensorReadout(
                DateTime.Now, this, "water",
PollWaterTemperature()));
            AppendToHistory(new PressureSensorReadout(
                DateTime.Now, this, "oil", PollOilPressure()));
        }

        protected double PollOilTemperature()
        {
            return 0.1*CountHistoryElements();
        }

        protected double PollWaterTemperature()
        {
            return 0.2*CountHistoryElements();
        }

        protected double PollOilPressure()
        {
            return 0.3*CountHistoryElements();
        }

        override public string ToString()
        {
            return _model + "[" + _pilot + "]/" +
CountHistoryElements();
        }

        private int CountHistoryElements()
        {
            return (_history == null ? 0 : _history.CountElements());
        }
```

PDF by iText, generated by Doctor, courtesy of db4objects

```
        private void AppendToHistory(SensorReadout readout)
        {
            if (_history == null)
            {
                _history = readout;
            }
            else
            {
                _history.Append(readout);
            }
        }
    }
}
```

7.1. Storing and updating

No surprises here.

```
[storeCar]

Pilot pilot = new Pilot("Rubens Barrichello", 99);
    Car car = new Car("BMW");
    car.Pilot = pilot;
    db.set(car);
```

Now we would like to build a sensor readout chain. We already know about the update depth trap, so we configure this first.

```
[setCascadeOnUpdate]

Db4o.configure().objectClass(typeof(Car)).cascadeOnUpdate(true);
```

Let's collect a few sensor readouts.

```
[takeManySnapshots]

ObjectSet result = db.get(new Car(null));
    Car car = (Car)result.next();
    for(int i=0; i<5; i++)
    {
        car.Snapshot();
    }
    db.set(car);
```

7.2. Retrieving

Now that we have a sufficiently deep structure, we'll retrieve it from the database and traverse it.

First let's verify that we indeed have taken lots of snapshots.

```
[retrieveAllSnapshots]

ObjectSet result = db.get(typeof(SensorReadout));
    while (result.hasNext())
    {
        Console.WriteLine(result.next());
    }
```

All these readouts belong to one linked list, so we should be able to access them all by just traversing our list structure.

```
[retrieveSnapshotsSequentially]

ObjectSet result = db.get(new Car(null));
    Car car = (Car)result.next();
```

```
    SensorReadout readout = car.GetHistory();
    while (readout != null)
    {
        Console.WriteLine(readout);
        readout = readout.Next;
    }
```

Ouch! What's happening here?

7.2.1. Activation depth

Deja vu - this is just the other side of the update depth issue.

db4o cannot track when you are traversing references from objects retrieved from the database. So it would always have to return 'complete' object graphs on retrieval - in the worst case this would boil down to pulling the whole database content into memory for a single query.

This is absolutely undesirable in most situations, so db4o provides a mechanism to give the client fine-grained control over how much he wants to pull out of the database when asking for an object. This mechanism is called *activation depth*and works quite similar to our familiar update depth.

The default activation depth for any object is 5, so our example above runs into nulls after traversing 5 references.

We can dynamically ask objects to activate their member references. This allows us to retrieve each single sensor readout in the list from the database just as needed.

```
[retrieveSnapshotsSequentiallyImproved]

ObjectSet result = db.get(new Car(null));
    Car car = (Car)result.next();
    SensorReadout readout = car.GetHistory();
    while (readout != null)
    {
        db.activate(readout, 1);
        Console.WriteLine(readout);
        readout = readout.Next;
```

```
    }
```

Note that 'cut' references may also influence the behavior of your objects: In this case the length of the list is calculated dynamically, and therefor constrained by activation depth.

Instead of dynamically activating subgraph elements, you can configure activation depth statically, too. We can tell our SensorReadout class objects to cascade activation automatically, for example.

```
[setActivationDepth]

Db4o.configure().objectClass(typeof(TemperatureSensorReadout))
        .cascadeOnActivate(true);
```

```
[retrieveSnapshotsSequentially]

ObjectSet result = db.get(new Car(null));
    Car car = (Car)result.next();
    SensorReadout readout = car.GetHistory();
    while (readout != null)
    {
        Console.WriteLine(readout);
        readout = readout.Next;
    }
```

You have to be very careful, though. Activation issues are tricky. Db4o provides a wide range of configuration features to control activation depth at a very fine-grained level. You'll find those triggers in com.db4o.config.Configuration and the associated ObjectClass and ObjectField classes.

Don't forget to clean up the database.

```
[deleteAllObjects]
```

```
ObjectSet result = db.get(new object());
    while (result.hasNext())
    {
        db.delete(result.next());
    }
```

7.3. Conclusion

Now we should have the tools at hand to work with arbitrarily complex object graphs. But so far we have only been working forward, hoping that the changes we apply to our precious data pool are correct. What if we have to roll back to a previous state due to some failure? In the next chapter we will introduce the db4o transaction concept.

7.4. Full source

```
namespace com.db4o.f1.chapter5
{
    using System;
    using System.IO;
    using com.db4o;

    public class DeepExample : Util
    {
        public static void Main(string[] args)
        {
            File.Delete(Util.YapFileName);
            ObjectContainer db = Db4o.openFile(Util.YapFileName);
            try
            {
                storeCar(db);
                db.close();
                setCascadeOnUpdate();
                db = Db4o.openFile(Util.YapFileName);
                takeManySnapshots(db);
                db.close();
                db = Db4o.openFile(Util.YapFileName);
```

```
                retrieveAllSnapshots(db);
                db.close();
                db = Db4o.openFile(Util.YapFileName);
                retrieveSnapshotsSequentially(db);
                retrieveSnapshotsSequentiallyImproved(db);
                db.close();
                setActivationDepth();
                db = Db4o.openFile(Util.YapFileName);
                retrieveSnapshotsSequentially(db);
                deleteAllObjects(db);
            }
            finally
            {
                db.close();
            }
        }

        public static void storeCar(ObjectContainer db)
        {
            Pilot pilot = new Pilot("Rubens Barrichello", 99);
            Car car = new Car("BMW");
            car.Pilot = pilot;
            db.set(car);
        }

        public static void setCascadeOnUpdate()
        {
Db4o.configure().objectClass(typeof(Car)).cascadeOnUpdate(true);
   }

        public static void takeManySnapshots(ObjectContainer db)
        {
            ObjectSet result = db.get(new Car(null));
            Car car = (Car)result.next();
            for(int i=0; i<5; i++)
            {
                car.Snapshot();
            }
            db.set(car);
        }
```

PDF by iText, generated by Doctor, courtesy of db4objects

```
        public static void retrieveAllSnapshots(ObjectContainer db)
        {
            ObjectSet result = db.get(typeof(SensorReadout));
            while (result.hasNext())
            {
                Console.WriteLine(result.next());
            }
        }

        public static void
retrieveSnapshotsSequentially(ObjectContainer db)
        {
            ObjectSet result = db.get(new Car(null));
            Car car = (Car)result.next();
            SensorReadout readout = car.GetHistory();
            while (readout != null)
            {
                Console.WriteLine(readout);
                readout = readout.Next;
            }
        }

        public static void
retrieveSnapshotsSequentiallyImproved(ObjectContainer db)
        {
            ObjectSet result = db.get(new Car(null));
            Car car = (Car)result.next();
            SensorReadout readout = car.GetHistory();
            while (readout != null)
            {
                db.activate(readout, 1);
                Console.WriteLine(readout);
                readout = readout.Next;
            }
        }

        public static void setActivationDepth()
        {
Db4o.configure().objectClass(typeof(TemperatureSensorReadout))
         .cascadeOnActivate(true);
        }
```

PDF by iText, generated by Doctor, courtesy of db4objects

```
        public static void deleteAllObjects(ObjectContainer db)
        {
            ObjectSet result = db.get(new object());
            while (result.hasNext())
            {
                db.delete(result.next());
            }
        }
    }
}
```

8. Transactions

Probably you have already wondered how db4o handles concurrent access to a single database. Just as any other DBMS, db4o provides a transaction mechanism. Before we take a look at multiple, perhaps even remote, clients accessing a db4o instance in parallel, we will introduce db4o transaction concepts in isolation.

8.1. Commit and rollback

You may not have noticed it, but we have already been working with transactions from the first chapter on. By definition, you are always working inside a transaction when interacting with db4o. A transaction is implicitly started when you open a container, and the current transaction is implicitly committed when you close it again. So the following code snippet to store a car is semantically identical to the ones we have seen before; it just makes the commit explicit.

```
[storeCarCommit]

Pilot pilot = new Pilot("Rubens Barrichello", 99);
    Car car = new Car("BMW");
    car.Pilot = pilot;
    db.set(car);
    db.commit();
```

```
[listAllCars]

ObjectSet result = db.get(new Car(null));
    listResult(result);
```

However, we can also rollback the current transaction, resetting the state of our database to the last commit point.

```
[storeCarRollback]
```

```
Pilot pilot = new Pilot("Michael Schumacher", 100);
    Car car = new Car("Ferrari");
    car.Pilot = pilot;
    db.set(car);
    db.rollback();
```

```
[listAllCars]

ObjectSet result = db.get(new Car(null));
    listResult(result);
```

8.2. Refresh live objects

There's one problem, though: We can roll back our database, but this cannot automagically trigger a rollback for our live objects.

```
[carSnapshotRollback]

ObjectSet result = db.get(new Car("BMW"));
    Car car = (Car)result.next();
    car.Snapshot();
    db.set(car);
    db.rollback();
    Console.WriteLine(car);
```

We will have to explicitly refresh our live objects when we suspect they may have participated in a rollback transaction.

```
[carSnapshotRollbackRefresh]
```

```
ObjectSet result=db.get(new Car("BMW"));
    Car car=(Car)result.next();
    car.Snapshot();
    db.set(car);
    db.rollback();
    db.ext().refresh(car, int.MaxValue);
    Console.WriteLine(car);
```

What is this ExtObjectContainer construct good for? Well, it provides some functionality that is in itself stable, but the API may still be subject to change. As soon as we are confident that no more changes will occur, *ext* functionality will be transferred to the common ObjectContainer API. We will cover extended functionality in more detail in a later chapter.

Finally, we clean up again.

```
[deleteAllObjects]

ObjectSet result = db.get(new object());
    while (result.hasNext())
    {
        db.delete(result.next());
    }
```

8.3. Conclusion

We have seen how transactions work for a single client. In the next chapter we will see how the transaction concept extends to multiple clients, whether they are located within the same VM or on a remote machine.

8.4. Full source

```
namespace com.db4o.f1.chapter5
{
    using System;
```

```
using System.IO;
using com.db4o;
using com.db4o.f1;

public class TransactionExample : Util
{
    public static void Main(string[] args)
    {
        File.Delete(Util.YapFileName);
        ObjectContainer db=Db4o.openFile(Util.YapFileName);
        try
        {
            storeCarCommit(db);
            db.close();
            db = Db4o.openFile(Util.YapFileName);
            listAllCars(db);
            storeCarRollback(db);
            db.close();
            db = Db4o.openFile(Util.YapFileName);
            listAllCars(db);
            carSnapshotRollback(db);
            carSnapshotRollbackRefresh(db);
            deleteAllObjects(db);
        }
        finally
        {
            db.close();
        }
    }

    public static void storeCarCommit(ObjectContainer db)
    {
        Pilot pilot = new Pilot("Rubens Barrichello", 99);
        Car car = new Car("BMW");
        car.Pilot = pilot;
        db.set(car);
        db.commit();
    }

    public static void listAllCars(ObjectContainer db)
    {
```

PDF by iText, generated by Doctor, courtesy of db4objects

```
            ObjectSet result = db.get(new Car(null));
            listResult(result);
        }

        public static void storeCarRollback(ObjectContainer db)
        {
            Pilot pilot = new Pilot("Michael Schumacher", 100);
            Car car = new Car("Ferrari");
            car.Pilot = pilot;
            db.set(car);
            db.rollback();
        }

        public static void deleteAllObjects(ObjectContainer db)
        {
            ObjectSet result = db.get(new object());
            while (result.hasNext())
            {
                db.delete(result.next());
            }
        }

        public static void carSnapshotRollback(ObjectContainer db)
        {
            ObjectSet result = db.get(new Car("BMW"));
            Car car = (Car)result.next();
            car.Snapshot();
            db.set(car);
            db.rollback();
            Console.WriteLine(car);
        }

        public static void carSnapshotRollbackRefresh(ObjectContainer
db)
        {
            ObjectSet result=db.get(new Car("BMW"));
            Car car=(Car)result.next();
            car.Snapshot();
            db.set(car);
            db.rollback();
            db.ext().refresh(car, int.MaxValue);
```

```
            Console.WriteLine(car);
        }
    }
}
```

9. Client/Server

Now that we have seen how transactions work in db4o conceptually, we are prepared to tackle concurrently executing transactions.

We start by preparing our database in the familiar way.

```
[setFirstCar]

Pilot pilot = new Pilot("Rubens Barrichello", 99);
    Car car = new Car("BMW");
    car.Pilot = pilot;
    db.set(car);
```

```
[setSecondCar]

Pilot pilot = new Pilot("Michael Schumacher", 100);
    Car car = new Car("Ferrari");
    car.Pilot = pilot;
    db.set(car);
```

9.1. Embedded server

From the API side, there's no real difference between transactions executing concurrently within the same VM and transactions executed against a remote server. To use concurrent transactions within a single VM, we just open a db4o server on our database file, directing it to run on port 0, thereby declaring that no networking will take place.

```
[accessLocalServer]

ObjectServer server = Db4o.openServer(Util.YapFileName, 0);
    try
```

```
    {
        ObjectContainer client = server.openClient();
        // Do something with this client, or open more clients
        client.close();
    }
    finally
    {
        server.close();
    }
```

Again, we will delegate opening and closing the server to our environment to focus on client interactions.

```
[queryLocalServer]

ObjectContainer client = server.openClient();
    listResult(client.get(new Car(null)));
    client.close();
```

The transaction level in db4o is *read committed* . However, each client container maintains its own weak reference cache of already known objects. To make all changes committed by other clients immediately, we have to explicitly refresh known objects from the server. We will delegate this task to a specialized version of our listResult() method.

```
public static void listRefreshedResult(ObjectContainer container,
ObjectSet items, int depth)
    {
        Console.WriteLine(items.size());
        while (items.hasNext())
        {
            object item = items.next();
            container.ext().refresh(item, depth);
            Console.WriteLine(item);
        }
```

```
    }
```

```
[demonstrateLocalReadCommitted]

ObjectContainer client1 =server.openClient();
    ObjectContainer client2 =server.openClient();
    Pilot pilot = new Pilot("David Coulthard", 98);
    ObjectSet result = client1.get(new Car("BMW"));
    Car car = (Car)result.next();
    car.Pilot = pilot;
    client1.set(car);
    listResult(client1.get(new Car(null)));
    listResult(client2.get(new Car(null)));
    client1.commit();
    listResult(client1.get(typeof(Car)));
    listRefreshedResult(client2, client2.get(typeof(Car)), 2);
    client1.close();
    client2.close();
```

Simple rollbacks just work as you might expect now.

```
[demonstrateLocalRollback]

ObjectContainer client1 = server.openClient();
    ObjectContainer client2 = server.openClient();
    ObjectSet result = client1.get(new Car("BMW"));
    Car car = (Car)result.next();
    car.Pilot = new Pilot("Someone else", 0);
    client1.set(car);
    listResult(client1.get(new Car(null)));
    listResult(client2.get(new Car(null)));
    client1.rollback();
    client1.ext().refresh(car, 2);
    listResult(client1.get(new Car(null)));
```

```
    listResult(client2.get(new Car(null)));
    client1.close();
    client2.close();
```

9.2. Networking

From here it's only a small step towards operating db4o over a TCP/IP network. We just specify a port number greater than zero and set up one or more accounts for our client(s).

```
[accessRemoteServer]

ObjectServer server = Db4o.openServer(Util.YapFileName, ServerPort);
    server.grantAccess(ServerUser, ServerPassword);
    try
    {
        ObjectContainer client = Db4o.openClient("localhost",
ServerPort, ServerUser, ServerPassword);
        // Do something with this client, or open more clients
        client.close();
    }
    finally
    {
        server.close();
    }
```

The client connects providing host, port, user name and password.

```
[queryRemoteServer]

ObjectContainer client = Db4o.openClient("localhost", port, user,
password);
    listResult(client.get(new Car(null)));
    client.close();
```

Everything else is absolutely identical to the local server examples above.

```
[demonstrateRemoteReadCommitted]

ObjectContainer client1 = Db4o.openClient("localhost", port, user,
password);
    ObjectContainer client2 = Db4o.openClient("localhost", port,
user, password);
    Pilot pilot = new Pilot("Jenson Button", 97);
    ObjectSet result = client1.get(new Car(null));
    Car car = (Car)result.next();
    car.Pilot = pilot;
    client1.set(car);
    listResult(client1.get(new Car(null)));
    listResult(client2.get(new Car(null)));
    client1.commit();
    listResult(client1.get(new Car(null)));
    listResult(client2.get(new Car(null)));
    client1.close();
    client2.close();
```

```
[demonstrateRemoteRollback]

ObjectContainer client1 = Db4o.openClient("localhost", port, user,
password);
    ObjectContainer client2 = Db4o.openClient("localhost", port,
user, password);
    ObjectSet result = client1.get(new Car(null));
    Car car = (Car)result.next();
    car.Pilot = new Pilot("Someone else", 0);
    client1.set(car);
    listResult(client1.get(new Car(null)));
    listResult(client2.get(new Car(null)));
    client1.rollback();
    client1.ext().refresh(car,2);
```

```
    listResult(client1.get(new Car(null)));
    listResult(client2.get(new Car(null)));
    client1.close();
    client2.close();
```

9.3. Conclusion

That's it, folks. No, of course it isn't. There's much more to db4o we haven't covered yet: schema evolution, custom persistence for your classes, writing your own query objects, etc.

This tutorial is work in progress. We will successively add chapters and incorporate feedback from the community into the existing chapters.

We hope that this tutorial has helped to get you started with db4o. How should you continue now?

-*(Interactive version only)*While this tutorial is basically sequential in nature, try to switch back and forth between the chapters and execute the sample snippets in arbitrary order. You will be working with the same database throughout; sometimes you may just get stuck or even induce exceptions, but you can always reset the database via the console window.

- The examples we've worked through are included in your db4o distribution in full source code. Feel free to experiment with it.

- I you're stuck, see if the FAQ can solve your problem, browse the information on our web site, check if your problem is submitted to Bugzilla or join our newsgroup at news://news.db4odev.com/db4o.users.

9.4. Full source

```
namespace com.db4o.f1.chapter5
{
    using System;
    using System.IO;
    using com.db4o;
    using com.db4o.f1;
```

```
    public class ClientServerExample : Util
    {
        public static void Main(string[] args)
        {
            File.Delete(Util.YapFileName);
            accessLocalServer();
            File.Delete(Util.YapFileName);
            ObjectContainer db = Db4o.openFile(Util.YapFileName);
            try
            {
                setFirstCar(db);
                setSecondCar(db);
            }
            finally
            {
                db.close();
            }

            configureDb4o();
            ObjectServer server = Db4o.openServer(Util.YapFileName,
0);
            try
            {
                queryLocalServer(server);
                demonstrateLocalReadCommitted(server);
                demonstrateLocalRollback(server);
            }
            finally
            {
                server.close();
            }

            accessRemoteServer();
            server = Db4o.openServer(Util.YapFileName, ServerPort);
            server.grantAccess(ServerUser, ServerPassword);
            try
            {
                queryRemoteServer(ServerPort, ServerUser,
ServerPassword);
                demonstrateRemoteReadCommitted(ServerPort,
ServerUser, ServerPassword);
```

```
                demonstrateRemoteRollback(ServerPort, ServerUser,
ServerPassword);
            }
            finally
            {
                server.close();
            }
        }

        public static void setFirstCar(ObjectContainer db)
        {
            Pilot pilot = new Pilot("Rubens Barrichello", 99);
            Car car = new Car("BMW");
            car.Pilot = pilot;
            db.set(car);
        }

        public static void setSecondCar(ObjectContainer db)
        {
            Pilot pilot = new Pilot("Michael Schumacher", 100);
            Car car = new Car("Ferrari");
            car.Pilot = pilot;
            db.set(car);
        }

        public static void accessLocalServer()
        {
            ObjectServer server = Db4o.openServer(Util.YapFileName,
0);
            try
            {
                ObjectContainer client = server.openClient();
                // Do something with this client, or open more
clients
                client.close();
            }
            finally
            {
                server.close();
            }
        }
```

PDF by iText, generated by Doctor, courtesy of db4objects

```
        public static void queryLocalServer(ObjectServer server)
        {
            ObjectContainer client = server.openClient();
            listResult(client.get(new Car(null)));
            client.close();
        }

        public static void configureDb4o()
        {
         Db4o.configure().objectClass(typeof(Car)).updateDepth(3);
        }

        public static void demonstrateLocalReadCommitted(ObjectServer
server)
        {
            ObjectContainer client1 =server.openClient();
            ObjectContainer client2 =server.openClient();
            Pilot pilot = new Pilot("David Coulthard", 98);
            ObjectSet result = client1.get(new Car("BMW"));
            Car car = (Car)result.next();
            car.Pilot = pilot;
            client1.set(car);
            listResult(client1.get(new Car(null)));
            listResult(client2.get(new Car(null)));
            client1.commit();
            listResult(client1.get(typeof(Car)));
            listRefreshedResult(client2, client2.get(typeof(Car)),
2);
            client1.close();
            client2.close();
        }

        public static void demonstrateLocalRollback(ObjectServer
server)
        {
            ObjectContainer client1 = server.openClient();
            ObjectContainer client2 = server.openClient();
            ObjectSet result = client1.get(new Car("BMW"));
            Car car = (Car)result.next();
            car.Pilot = new Pilot("Someone else", 0);
```

```
            client1.set(car);
            listResult(client1.get(new Car(null)));
            listResult(client2.get(new Car(null)));
            client1.rollback();
            client1.ext().refresh(car, 2);
            listResult(client1.get(new Car(null)));
            listResult(client2.get(new Car(null)));
            client1.close();
            client2.close();
        }

        public static void accessRemoteServer()
        {
            ObjectServer server = Db4o.openServer(Util.YapFileName,
ServerPort);
            server.grantAccess(ServerUser, ServerPassword);
            try
            {
                ObjectContainer client = Db4o.openClient("localhost",
ServerPort, ServerUser, ServerPassword);
                // Do something with this client, or open more
clients
                client.close();
            }
            finally
            {
                server.close();
            }
        }

        public static void queryRemoteServer(int port, string user,
string password)
        {
            ObjectContainer client = Db4o.openClient("localhost",
port, user, password);
            listResult(client.get(new Car(null)));
            client.close();
        }

        public static void demonstrateRemoteReadCommitted(int port,
string user, string password)
```

```
        {
            ObjectContainer client1 = Db4o.openClient("localhost",
port, user, password);
            ObjectContainer client2 = Db4o.openClient("localhost",
port, user, password);
            Pilot pilot = new Pilot("Jenson Button", 97);
            ObjectSet result = client1.get(new Car(null));
            Car car = (Car)result.next();
            car.Pilot = pilot;
            client1.set(car);
            listResult(client1.get(new Car(null)));
            listResult(client2.get(new Car(null)));
            client1.commit();
            listResult(client1.get(new Car(null)));
            listResult(client2.get(new Car(null)));
            client1.close();
            client2.close();
        }

        public static void demonstrateRemoteRollback(int port, string
user, string password)
        {
            ObjectContainer client1 = Db4o.openClient("localhost",
port, user, password);
            ObjectContainer client2 = Db4o.openClient("localhost",
port, user, password);
            ObjectSet result = client1.get(new Car(null));
            Car car = (Car)result.next();
            car.Pilot = new Pilot("Someone else", 0);
            client1.set(car);
            listResult(client1.get(new Car(null)));
            listResult(client2.get(new Car(null)));
            client1.rollback();
            client1.ext().refresh(car,2);
            listResult(client1.get(new Car(null)));
            listResult(client2.get(new Car(null)));
            client1.close();
            client2.close();
        }
    }
}
```

10. Encryption

db4o provides built-in encryption functionality.

In order to use it, the following two methods have to be called, before a database file is created:

```
Db4o.configure().encrypt(true);
Db4o.configure().password("yourEncryptionPasswordHere");
```

The security standard of the built-in encryption functionality is not very high, not much more advanced than "substract 5 from every byte".

There are 2 reasons for not providing more advanced encryption functionality:
(1) The db4o library is designed to stay small and portable.
(2) The db4o team is determined to avoid problems with U.S. security regulations and export restrictions.

db4o still provides a solution for high-security encryption by allowing any user to choose his own encryption mechanism that he thinks he needs:
The db4o file IO mechanism is pluggable and any fixed-length encryption mechanism can be added. All that needs to be done is to write an IoAdapter plugin for db4o file IO.

This is a lot easier than it sounds. Simply:
- take the sources of com.db4o.io.RandomAccessFileAdapter
- modify the #read() and #write() methods to encrypt and decrypt when bytes are being exchanged with the file
- plug your adapter into db4o with the following method:

```
Db4o.configure().io(new MyEncryptionAdapter());
```

11. Tuning

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.

Measure

```
Db4o.configure().discardFreeSpace(byteCount);
```

Recommended settings for byteCount:
- Integer.MAX_VALUE will turn freespace management off
- Moderate range: 10 to 50
- Default built-in setting: 0

Advantage
will reduce the RAM memory overhead and the speed loss from maintaining the freespace lists.

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:

```
com.db4o.tools.Defragment
```

(supplied as source code insrc/com/db4o/tools)
If defragment can be frequently run, it will also reclaim lost space and decrease the database file to the minimum size. Therefore #discardFreeSpace() may be a good tuning mechanism for setups with frequent defragment runs.

Measure

```
Db4o.configure().weakReferences(false);
```

Advantage

will configure db4o to use hard direct references instead of weak references to control instantiated and stored objects.

Effect
A db4o database keeps a reference to all persistent objects that are currently held in RAM, whether they were stored to the database in this session or instantiated from the database in this session. This is how db4o can "know" than an object is to be updated: Any "known" object must be an update, any "unknown" object will be stored as "new". (Note that the reference system will only be in place as long as an ObjectContainer is open. Closing and reopening an ObjectContainer will clean the references system of the ObjectContainer and all objects in RAM will be treated as "new" afterwards.) In the default configuration db4o uses weak references and a dedicated thread to clean them up after objects have been garbage collected by the VM. Weak references need extra ressources and the cleanup thread will have a considerable impact on performance since it has to be synchronized with the normal operations within the ObjectContainer. Turning off weak references will improve speed.
The downside: To prevent memory consumption from growing consistantly, the application has to take care of removing unused objects from the db4o reference system by itself. This can be done by calling

```
ExtObjectContainer.purge(object);
```

Alternate Strategies


```
ExtObjectContainer.purge(object);
```


can also be called in normal weak reference operation mode to remove an object from the reference cache. This will help to keep the reference tree as small as possible. After calling #purge(object) an object will be unknown to the ObjectContainer so this feature is also suitable for batch inserts.

Measure

```
Db4o.configure().automaticShutDown(false);
```

Advantage
can prevent the creation of a shutdown thread on some platforms.

Effect

On some platforms db4o uses a ShutDownHook to cleanly close all database files upon system termination. If a system is terminated without calling ObjectContainer#close() for all open ObjectContainers, these ObjectContainers will still be usable but they will not be able to write back their freespace management system back to the database file. Accordingly database files will be observed to grow.

Alternate Strategies
Database files can be reduced to their minimal size with

```
com.db4o.tools.Defragment
```

(supplied as source code in /src/com/db4o/tools)

Measure

```
Db4o.configure().callbacks(false);
```

Advantage
will prevent db4o from looking for callback methods in all persistent classes on system startup.

Effect
Upon system startup, db4o will scan all persistent classes for methods with the same signature as the methods defined in com.db4o.ext.ObjectCallbacks, even if the interface is not implemented. db4o uses reflection to do so and on constrained environments this can consume quite a bit of time. If callback methods are not used by the application, callbacks can be turned off safely.

Alternate Strategies
Class configuration features are a good alternative to callbacks. The most recommended mechanism to cascade updates is:

```
Db4o.configure().objectClass("yourPackage.yourClass").cascadeOnUpdate
(true);
```

Measure

```
Db4o.configure().detectSchemaChanges(false);
```

Advantage
will prevent db4o from analysing the class structure upon opening a database file.

Effect
Upon system startup, db4o will use reflection to scan the structure of all persistent classes. This process can take some time, if a large number of classes are present in the database file. For the best possible startup performance on "warm" database files (all classes present), this feature can be turned off.

Alternate Strategies
Instead of using one database file to store a huge and complex class structure, a system may be more flexible and faster, if multiple database files are used. In a client/server setup, database files can also be switched from the client side with

```
((ExtClient)objectContainer).switchToFile(databaseFile);
```

Measure

```
Db4o.configure().lockDatabaseFile(false);
```

Advantage
will prevent the creation of a lock file thread on Java platforms without NIO (< JDK 1.4.1).

Effect
If file locking is not available on the system, db4o will regularily write a timestamp lock information to the database file, to prevent other VM sessions from accessing the database file at the same time. Uncontrolled concurrent access would inevitably lead to corruption of the database file. If the application ensures that it can not be started multiple times against the database file, db4o file locking may not be necessary.

Alternate Strategies
Database files can safely be opened from multiple sessions in readonly

mode. Use:

```
Db4o.configure().readOnly(true)
```

Measure

```
Db4o.configure().testConstructors(false);
```

Advantage
will prevent db4o from creating a test instance of persistent classes upon opening a database file.

Effect
Upon system startup, db4o attempts to create a test instance of all persistent classes, to ensure that a public zero-parameter constructor is present. This process can take some time, if a large number of classes are present in the database file. For the best possible startup performance this feature can be turned off.

Alternate Strategies
In any case it's always good practice to create a zero-parameter constructor. If this is not possible because a class from a third party is used, it may be a good idea to write a translator that translates the third party class to one's own class. The dowload comes with the source code of the preconfigured translators in
src/com/db4o/samples/translators.
The default configuration can be found in the above folder in the file
Default.java/Default.cs
Take a look at the way the built-in translators work to get an idea how to write a translator. It just requires implementing 3 (4 for ObjectConstructors) methods and configuring db4o to use a translator on startup with

```
Db4o.configure().objectClass("yourPackage.yourClass").translate()
```

12. License

db4objects Inc. supplies the object database engine db4o under a dual licensing regime:

12.1. General Public License (GPL)

db4o is free to be used:

- for development,
- in-house as long as no deployment to third parties takes place,
- together with works that are placed under the GPL themselves.

You should have received a copy of the GPL in the file GPL.txt together with the db4o distribution.

12.2. 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 purchasing area on the db4o website or contact db4o sales for licensing terms and pricing.

13. Contacting db4objects Inc.

db4objects Inc.
1900 South Norfolk Street
Suite 350
San Mateo, CA, 94403
USA

Phone
+1 (650) 577-2340

Fax
+1 (650) 577-2341

General Enquiries
info@db4o.com

Sales
fill out our sales contact form on the db4o website
or
sales@db4o.com

Careers

career@db4o.com

Partnering

partner@db4o.com

Support

support@db4o.com

or post to our newsgroup

news://news.db4odev.com/db4o.users