PART II

Working with db4o

In Part I you were introduced to object orientation with a focus on its role in object-oriented databases. In Part II we formally introduce db4o, showing you how to perform queries and transactions, handle complex objects, use the client/server mode, configure replication, use db4o reflection, extend db4o using pluggable interfaces, and a lot more.

CHAPTER 5

Quick Start

The aim of this chapter is to show you how to get up and running very quickly with db4o, and to introduce some of the basic techniques you need to use it. You should be able to install db4o, and in just a few minutes create a simple standalone application that stores and retrieves some objects. The techniques described in this chapter illustrate the standard database operations: create, read, update, and delete. Once you know how to do these operations, you will want to go on to explore the other features that db4o offers.

C# or Java?

Before you can start using db4o, you need to decide what version to use. Your decision depends on the language and platform you want to use for your application. db4o provides native persistence for .NET and Java objects, so you can use several languages, such as C#, Java, and all .NET managed languages, including VB.NET, ASP.NET, Boo, and Managed C++. There are three distinct versions of db4o available: one for .NET, one for Java, and one for Mono (the Mono project offers cross-platform support for .NET and C#). In this book, we will focus on both C# and Java. db4o supports both of these languages in a very similar manner, which is perhaps not surprising as the languages themselves have many similarities. In fact, the similarities are so broad that you can even download a converter that will translate one language to the other. You can find one of the most popular such converters at `http://msdn.microsoft.com/vstudio/downloads/tools/jlca`, which is available as a plug-in for Microsoft Visual Studio and converts Java to C# quite reliably.

db4o was originally developed when C# hadn't yet been released, so it has a strong Java heritage. However, interest in C# and the .NET environment has grown rapidly, and so db4objects now puts a lot of effort into keeping the two versions as close and feature-equivalent as possible.

In the examples found throughout Part II of this book we try to do the same. C# and Java code listings are provided, except in cases where the code is absolutely identical in the two languages. In some cases the difference is simply the case notation supported by the db4o methods: *PascalCase* in C# and *camelCase* in Java. For example, the method that is named `OpenFile` in C# is named `openFile` in the Java version. These notations follow the common practices used by programmers in the respective languages.

■ **Caution** .NET versions prior to 5.0 do not support PascalCase notation for db4o method names. db4objects recommends that new .NET users adopt PascalCase, while existing users may choose to retain camelCase.

The main language differences that arise in the examples are:

- `import`/`using` to include libraries
- `namespace`/`package` to qualify the name of the class
- Inheritance syntax
- PascalCase/camelCase notation differences
- C# properties/Java getters and setters
- C# delegates (see Chapter 6)
- Differences in collections API classes (see Chapter 7)
- Event handling

Of course, if you don't want or need to know about the differences, you can simply look at the examples in your own langauge.

Installing db4o

In this section you'll learn how to install db4o and tailor the configuration for both C# and Java, and for different IDEs. The installation is very simple and requires only that you run an installer or unzip or double-click an archive, depending on which version you are using.

You can find a link to the db4o Product Test Drive on the db4objects.com home page, `www.db4objects.com`. Clicking this link allows you to download the current release of db4o. You can choose the appropriate version for your development platform. Note that the Java version will run on any Java-enabled platform, including Windows and Linux. The .NET version is for Windows only, and is available in separate forms that support .NET 1.1 and .NET 2.0, respectively. Separate Mono versions are provided for Windows and Linux.

Let's have a look at the db4o folders:

- A `doc` folder that contains the API documentation as HTML or a Microsoft Compressed HTML Help (CHM) file, and also a useful interactive tutorial that allows you to run live examples in an applet. The tutorial, which is also provided as a PDF file, contains a wealth of examples that complement those in this book.
- The `dll` (.NET) or `lib` (Java) folder, which includes the db4o DLLs or JARs. Different libraries are provided to work with different platform versions, for example, .NET Framework or Compact Framework, or Java 1.4 or 5.0.
- The `src` folder, which includes the full db4o source code along with some tools that you might find useful.

If you go to the Download Center on the db4objects website, you can also download the ObjectManager graphical object database browser, which you have already seen in Chapter 4. A guide to getting started with the ObjectManager is given later in this chapter. To get to the Download Center, you need to follow the Free Resources link and register—this is well worth doing, as you will find further useful material there, including community forums and a knowledge base.

■ **Note** All examples in this book are based on version 5.2 of db4o, which was the most recent version at the time of writing.

Installing db4o for C#

Using C# in conjunction with db4o requires that the appropriate .NET Framework be installed. Installing Microsoft Visual Studio should automatically install the .NET Framework and Compact Framework (version 1.1 with VS 2003, version 2.0 with VS 2005). Note that the examples in this book will also run in the free Visual C# 2005 Express Edition.

Alternatively, you can use the popular and free #develop IDE, which can be downloaded from `www.icsharpcode.net/OpenSource/SD`. In this case, you need to install the .NET Framework 1.1 or later, which can be downloaded from `www.microsoft.com`, before installing the IDE.

If you are using Mono, you must install the Mono libraries, and you will probably want to use the MonoDevelop IDE.

The .NET/Windows distribution of db4o provides an `.msi` archive. This can be installed by either double-clicking the archive or using the command prompt to extract to a location you specify, as shown here:

```
C:\DOWNLOAD\db4oNET>msiexec /a "C:\DOWNLOAD\db4oNET\db4o-5.2-net.msi" /qn ➥
TARGETDIR="C:\DOWNLOAD\INSTALL"
```

If you double-click on the `.msi` icon to start the installation, then you are also able to set the installation directory folder for db4o.

The Mono distribution for Windows gives you a `.zip` archive, which you simply extract to a location of your choice. Mono for Linux distributions are available as RPMs or as a `.tar.gz` archive, which you should extract to your chosen installation directory.

Using db4o in a C# Project

To start using db4o, you simply include a reference to `db4o.dll` in your project. The .NET distribution contains two DLL files with the same name, one in a folder called `net` inside the `dll` folder, the other in a folder called `compact`. The former is the library for use with the standard .NET Framework, while the latter is the Compact Framework library.

The screenshots in this section show a reference being added in #develop, but the steps required are very similar in the other .NET IDEs. If you know how to add a reference in a project in your IDE, then you know how to include db4o.

To add the reference in #develop, right-click on the References node in your project and select the Add Reference menu item (see Figure 5-1). You will then see the Add Reference dialog box, which allows you to add the `db4o.dll` to the project (see Figure 5-2).

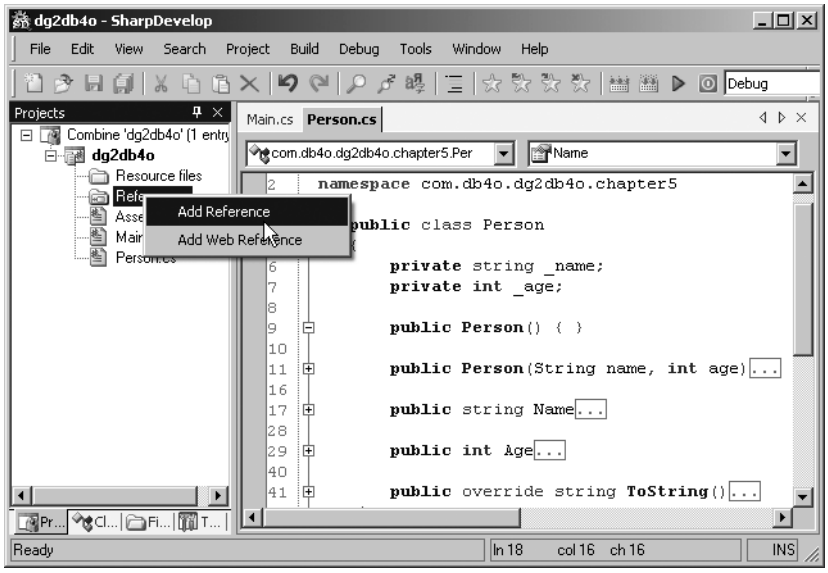

Figure 5-1. *Adding a reference to your C# project in #develop*

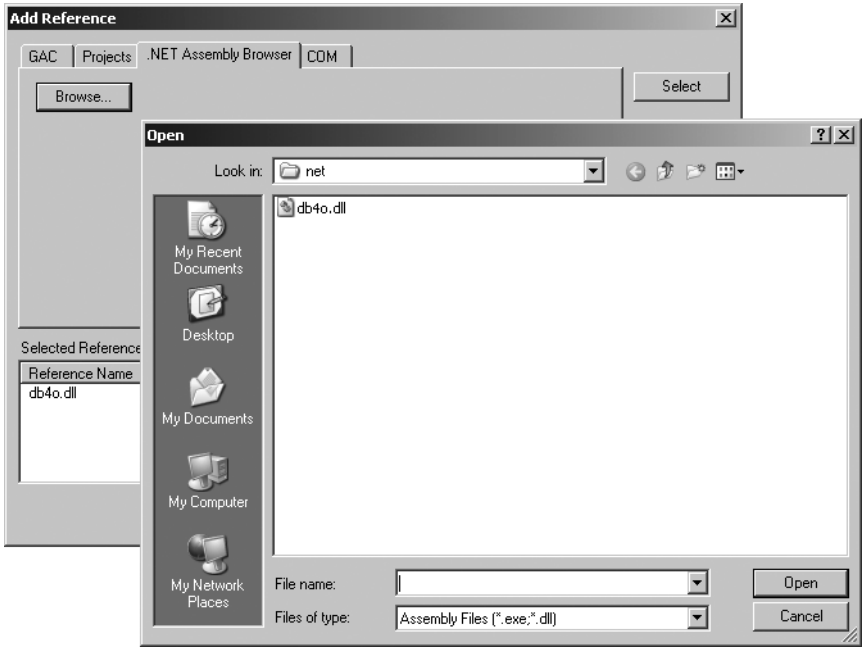

Figure 5-2. *Selecting the db4o.dll in #develop*

Installing db4o for Java

You need to make sure you have Java Runtime Environment installed. We recommend that you use Java 5.0 if possible. Many of the examples in this book use language features, such as generics and the enhanced `for` loop, which were introduced in Java 5.0. However, db4o provides good support for Java versions from 1.2 on.

The cross-platform Java distribution of db4o gives you a `.zip` archive, which you simply extract to a location of your choice.

Using db4o in a Java Project

To begin using db4o, you simply include add the appropriate JAR file to your classpath, or add it as a library to a project in an IDE. You can use db4o with any Java IDE. The procedure described that follows shows this for Eclipse, but the steps required are very similar in the other Java IDEs. If you are using NetBeans or IDEA, for example, you will know how to add libraries in your own IDE.

If you are using Eclipse, you'll have to complete the following steps, as shown in Figure 5-3:

1. Open the project Properties dialog box by right-clicking on your project in the Package Explorer pane and selecting Properties.
2. Select Java Build Path on the left and click the Libraries tab.
3. Finally, add the JAR by clicking the Add External JARs button and browsing to find the db4o JAR in the directory where you installed it.

Be careful to select the right JAR file for the version of Java you are using. For example, `db4o-5.2-java5.jar` supports Java 5.0.

As soon as the classpath is set to your JAR location, you are ready to create your first db4o database—by simply storing an object.

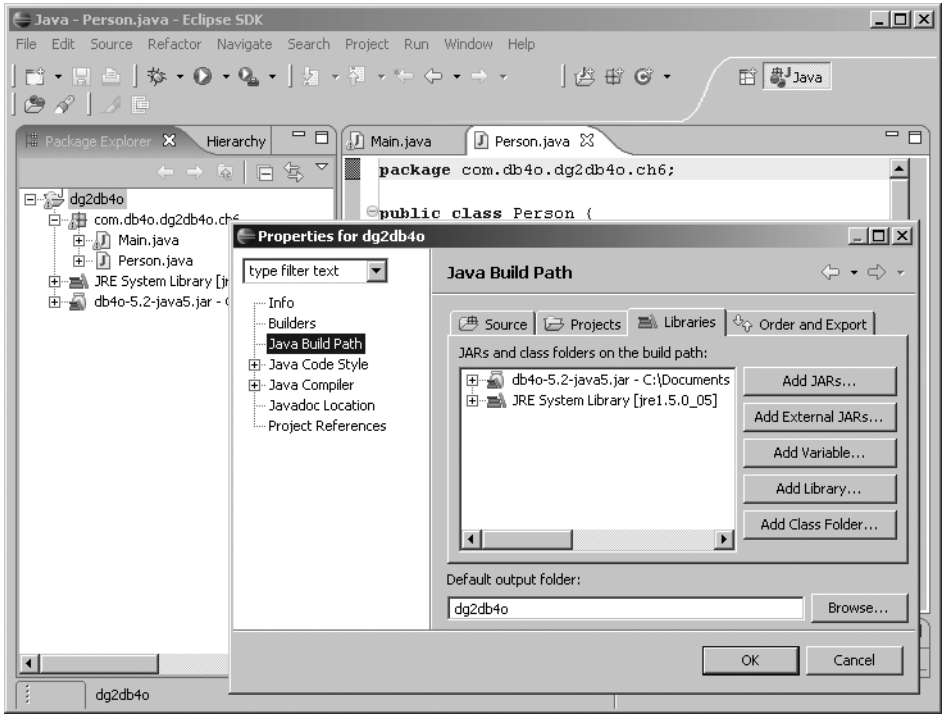

Figure 5-3. *Adding the db4o JAR to a Java project in Eclipse*

Running db4o

If you are coming to db4o with previous experience of typical RDBMSs like MySQL or Oracle, your first thoughts at this point are probably something like:

- How do I create the database?
- How do I start the database?

A database management system usually runs as a daemon or service and provides tools to allow the administrator to define and maintain database schemata, and to manage users, roles, and views. Applications usually interact with the database in a client/server model.

With db4o, the simple answer to both questions is: *you don't*. To understand the reasons for this, think about the most common use case for this kind of database. db4objects Inc. mostly targets the embedded database market, which means that db4o will typically be bundled as a library with your application code. It is often installed on standalone clients, which may have no constant connection to any server. db4o is not unique in working in this mode: there are several other relational databases, including H2 and Hypersonic (HSQLDB), which are mostly used in the same way. The difference is, of course, that db4o is an object database.

db4o can, like H2 and HSQLDB, also be used in a client/server mode, so it can in fact be started as a server. We will explain how to use db4o in the client/server mode in Chapter 8. For now, we will stick to using it in standalone mode. You can learn a great deal about db4o by

using it in this mode, and this is probably the most common way in which it is used in applications at present.

Creating the Database

A db4o database is contained in a single file. If you are running db4o in the standalone mode, then you don't create a database in advance. Instead, with a single line of code, you obtain a handle to a database; this handle is then used to perform operations. The handle is actually an object reference of the db4o API type `ObjectContainer`. You get the handle by calling a static method `OpenFile` of the `Db4o` class, specifying the path to the file as a parameter. If the specified file doesn't exist, then it is created for you as a new database; otherwise, a handle to an existing database is returned. Often, for brevity, we talk about this procedure as "opening an `ObjectContainer`," which is a nicely descriptive, if not completely accurate, phrase.

Thereafter, you can benefit from one of the key advantages of native object databases as you simply store your object with no additional effort. The database and its schema are created "on the fly" for you. The database schema is simply your class definition. The great benefit of this approach is that you don't need to worry about creating tables and working out object-relational mappings. Instead, you can start storing your objects immediately.

This is fine in standalone applications. The disadvantage if you are running in client/server mode is that multiple clients using this database will usually need to have the class definitions in their classpaths. This is in turn the advantage of relational databases, where you can write different applications with different object models, which map onto the relational data model of the database.

Starting the Database

In standalone mode you don't actually need to start the database—you simply open the database file. However, there are some implications you should keep in mind when working in standalone mode:

- Since you didn't start the database, you don't need to stop it, although you do need to release the handle by closing the `ObjectContainer`.
- You don't need to connect to the database. You simply store your objects. This means that no username or password is required. Passwords may be required in client/server mode, as discussed in Chapter 8.
- New database files are created using your filesytem permissions. In standalone mode, it is likely that only you are using the database file. However, multiple clients can potentially use a database file. In this case, you have to ensure that the group permissions are set correctly for write or read-only access; db4o also directly supports the option to open an `ObjectContainer` for read-only access (see Chapter 10).

In client/server mode, db4o usage is much more conventional, and you need to consider issues like starting and stopping a server and user authentication. However, the basic database operations are identical regardless of the database mode. In the remainder of this chapter, you will see how those operations are done, while the following two chapters go further and deal with advanced queries and handling complex objects—these techniques are also independent of the operating mode.

db4o Basics

In this section you will learn to create a very simple application capable of storing, restoring, modifying, and deleting objects in a database. Later you will be able to inspect this database using the ObjectManager.

Storing Objects

Before we can do anything useful with db4o, we need to define a class. We can use the very simple `Person` class, shown in Listing 5-1 in C# and Java. The `Person` class has two attributes: a constructor and a `ToString` method.

Listing 5-1. *The Person Class in C# and Java*

```
// C#
using System;
namespace com.db4o.dg2db4o.chapter5
{
    public class Person
    {
        private string _name;
        private int _age;

        public Person() { }

         get{ return name; }

        public Person(String name, int age)
        {
            _name = name;
            _age = age;
        }

        public string Name
        {
            get
            {
                return _name;
            }
            set{
            {
                _name = value; }
            }
        }
```

```
        public int Age
        {
            get {
            {
                return _age; }
            }
            set {
            {
                _age = value };
            }
        }

         public override string ToString()
         {
             return "[" + Name + ";" + Age + "]";
        }
    }
}


package com.db4o.dg2db4o.chapter5;

public class Person
{
    private String _name;
    private int _age;

    public Person(){}

    public Person(String name, int age)
    {
        _name = name;
        _age = age;
    }

    public int getAge()
    {
       return _age;
    }

    public void setAge(int value)
    {
        _age = value;
    }
```

```
    public String getName()
    {
        return _name;
    }

    public void setName(String value)
    {
        _name = value;
    }

    public String toString()
    {
        return "[" + _name + ";"+ _age + "]";
    }
}
```

Note that the `Person` class, in either language, contains no db4o code. Other persistence solutions often require the inheritance of another class, the implementation of another interface, or a class file enhancement, making the development process rather tedious and tied tightly to a specific vendor's solution.

To access a database, you need to open an `ObjectContainer` using the following code. The string parameter in the method call is the full path to the database file that you want to create or access.

```
// C#
ObjectContainer db = Db4o.OpenFile("C:/myDb.yap");


ObjectContainer db = Db4o.OpenFile("C:/myDb.yap");
```

The static method `openFile` is called directly on the class `Db4o`. The return value of this call is the `ObjectContainer` reference, named `db` in this example, which is your handle to the database.

As you can see from this example, db4o stores the data in a file using a name and location of your choosing. This file can grow up to 256GB and will contain any data objects you choose to store. If this still isn't enough space, you can set up more database files and put different classes in different files. This gives you virtually unlimited storage space.

If the database file you named does not exist, it will be created for you. If the database file already exists, it will be reopened for your access. For testing purposes, you might want to place the database in a known state before running your tests—for example, within a suite of unit tests. A very simple way to do this is to delete the database before running the test, which can be done by calling the method `File.Delete("C:/myDb.yap")` in C# or `new File ("C:/myDb.yap").delete()` in Java.

■ **Note** db4o database files are by convention given the extension `.yap`. This stands for "yet another protocol" and has no obvious meaning, but is an extension not used by any other application (as far as we are aware). This helps to distinguish db4o databases in your file system. If you wish, you can, however, give your database file any name you like.

Now let's create and store an object:

```
// C#
Person p = new Person("Gandhi",42);
db.Set(p);

// JAVA
Person p = new Person("Gandhi", 79);
db.set(p);
```

That's it. Nothing more. This object is assigned an internal object ID (OID), the value of which is not important for now, and is stored in the file you named. If you open the file in a text editor, you may be able to see fragments of the object's representation in the database.

Closing the Database

It is important to make sure you close the `ObjectContainer` by calling `db.Close()`. This ends the transaction that began when you opened the `ObjectContainer`. It causes all database operations to be written to the database and closes the database properly. A common problem arises when an exception occurs and the database file is not properly closed. This can leave the database in a locked state, which leads to a "database locked" exception upon reopening the database file. In this case, you will also lose any data stored since the beginning of the transaction.

To avoid this kind of problem, you should take care that all your operations are enclosed in a `try/catch` block, with a call to `Close` in a corresponding `finally` block.

■ **Note** The data you store with a call to `Set` is not actually written to the database immediately: it is stored at end of the transaction. db4o provides further features for committing and rolling back transactions—these are described in Chapter 9.

Retrieving Objects

The ability to selectively retrieve data by querying is a key feature of any database. The basic query mechanism provided by db4o is known as query-by-example (QBE). Let's see how to use it to retrieve the object that has just been stored.

Using QBE, you create an example, or template, object of the class you want to look up. You then set the values of the attribute (or attributes) you want to use as search criteria—all other attributes are left with null values, or zero values in the case of numerical primitives. To execute the query, you call the `Get` method of the `ObjectContainer`, specifying the template object as the method parameter. The query returns a collection that holds all the objects that match the attribute pattern you provided.

To make things a little bit more realistic, we will first add another object to the database:

```
// C#
db.Set(new Person("Lincoln",46));  // Later we correct Lincoln's age to 56...


db.set(new Person("Lincoln",46));
```

The following code sets up a template `Person` object. Let's assume we want to find Mahatma Gandhi's age. To achieve this, we fill the template object with the appropriate `Name` attribute and pass it to the `Get` method:

```
// C#
Person p = new Person(); // Java
p.Name = "Gandhi";
ObjectSet res = db.Get(p);
while(res.HasNext())
   Console.WriteLine(res.Next()); // followed by Commit and Close


Person p = new Person();
p.setName("Gandhi");
ObjectSet result = (ObjectSet) db.get(p);
while (result.hasNext()) {
   System.out.println((Person) result.next()); // followed by commit and close
```

This shows that the `Get` call returns a reference to an `ObjectSet`. `ObjectSet` is a db4o API class that behaves like a classical `Iterator/(I)Enumeration`. Using the `HasNext` and `Next` methods, you can obtain all the objects in the database that have matched your template object. The output will now be

```
[Gandhi;79]
```

Note that the `ToString` method of the `Person` class specifies exactly what is output for each object returned by the query. Incidentally, you should make sure you never call `Next` twice before calling `HasNext`, or you will bypass some data.

To achieve a better code readability, we can define a `ListResult` method that simply prints out all the objects contained in the `ObjectSet`:

```
// C#
private void ListResult(ObjectSet result)
{
    while(result.HasNext())
        Console.WriteLine(result.Next());
}


public static void listResult(ObjectSet result)
{
    while (result.hasNext())
        System.out.println(result.next());
}
```

With this definition, the query example is simplified to:

```
// C#
Person p = new Person();
p.Name = "Gandhi";
ObjectSet res = db.Get(p);
ListResult(res);


Person p = new Person();
p.setName("Gandhi");
ObjectSet result = (ObjectSet) db.get(p);
listResult(res);
```

If you need to retrieve all objects of a specific class, you can provide null values for the template object by using the default constructor, or by setting the default values explicitly:

```
// C# & JAVA
Person p1 = new Person(null,0);  // or
Person p2 = new Person();
```

Furthermore if you run a query specifying a null object as the template object, you will get a collection of all objects in the database.

■ **Note** Since db4o version 4.6, the `ObjectSet` interface implements several superinterfaces such as `IList`, `ICollection`, and `IEnumerable` in C#, and `java.util.Collection`, `java.lang.Iterable`, `java.util.Iterator`, and `java.util.List` in Java. This implies that you have a wide range of methods available to handle your results. For example, if your database contains the two persons Gandhi and Lincoln, you can use the `ObjectSet` returned to apply, for example, the Java `List` method `subList(1,2)` to retrieve a subset of the list. Furthermore, sorting is quite simple using the `Sort` methods provided by C# and Java collections.

You have now seen a simple example of querying. db4o supports two additional query mechanisms that provide the capability to perform much more sophisticated queries. These will be discussed in Chapter 6.

Updating Objects

To update an object you simply retrieve it, make the required changes, and call `Set` again to store the modified object.

In the following example we modify Abraham Lincoln's age:

```
// C#
ObjectSet result = (ObjectSet) db.Get(new Person("Lincoln", 0));
Person lincoln = (Person) result.Next();
lincoln.Age = 56;
db.Set(lincoln);
```

```
ObjectSet result = (ObjectSet) db.get(new Person("Lincoln", 0));
Person lincoln = (Person) result.next();
lincoln.setAge(56);
db.set(lincoln);
```

If you were to now query again, you would find Lincoln's age to be set to 56 in the database.

Actually, the process is a bit subtler than this example suggests. The result of the query is that a `Person` object is created in memory, and db4o uses IDs to maintain a connection between the in-memory object and the equivalent object stored in the database. It does this by caching the db4o OID as a weak reference. This connection is maintained only until the database is closed.

To update an object in the database, you need to have a fresh reference in memory that is connected to the database object (see the Caution that follows). This can be obtained by querying as already described. Alternatively, creating a new object and calling `Set` has a similar effect—the in-memory object is connected to a new database object. This means that you can, if you wish, store a new object and update it immediately afterward.

To do the update, you call `Set` for the modified object, and db4o will automatically find and update this object in the database as the Java/C# and db4o IDs match.

■ **Caution** Make sure that you have stored or retrieved the object within the same transaction (i.e., since the `ObjectContainer` was last opened) before your update. If not, db4o will assume that this is a new object, not one that has been stored previously. The result will be to clone Abraham Lincoln in the database with different ages (and different OIDs), which is probably prohibited by the U.S. Constitution! db4o does not provide primary keys to prevent duplicate Lincolns being stored.

Deleting Objects

As you have seen, updating an object and saving an object are similar in that they use the same `ObjectContainer` method. While deleting an object requires a different method, the basic idea

is similar. You simply retrieve the object and call the `Delete` method of the `ObjectContainer`. As was the case for updating, the database has to know what object you want to delete, so the object to be deleted either has to be set (which doesn't make much sense in this case), or more likely retrieved within the same transaction.

Let's assume we want to delete the first `Person` object from the database:

```
// C#
ObjectSet result = (ObjectSet) db.Get(new Person());
Person p = (Person) result.Next();
db.Delete(p);
```

```
ObjectSet result = (ObjectSet) db.get(new Person());
Person p = (Person) result.next();
db.delete(p);
```

If you then query the database for all `Person` objects and list the results, you find only one object, either Lincoln or Gandhi, depending on exactly how they were stored. Alternatively, you can easily query for a specific object, for example by specifying that you want the object with the name Gandhi, and then delete that target object.

Startup Options

Before you start working with the `ObjectContainer`, you have a wide range of options for configuring the database and the container itself to suit your application. db4o's configuration features will be described in detail in Chapter 10. To get a flavor of this, here are a few useful items you might consider configuring now:

- A call to `Db4o.main` prints the version of db4o you are currently running (e.g., `db4o 5.2.004`) to the console. This can also be done by outputting the value of `Db4o.version()`.
- Customers who have licensed db4o might need to pass an email address in order to use the database: `Db4o.LicensedTo("ray@charles.com")`.
- A call `Db4o.configure().MessageLevel(int level)` provides a simple db4o logging. It supports the four logging levels shown in Table 5-1.

Table 5-1. *db4o Logging Levels*

db4o Logging Levels	Description
0	No messages
1	Open and close messages
2	Messages for new, update, and delete
3	Messages for activate and deactivate

If you save an object with message level 2 or 3 configured, you will see output like this, which shows the db4o database object ID:

```
396 new com.db4o.dg2db4o.chapter5 Person
```

A Complete Example

The next example contains everything we have covered in this chapter in one simple console application. Note that the majority of examples in the book are console applications, as this is the simplest way to try out and learn about db4o's API and capabilities.

In this example, two `Person` objects are stored; then they both are retrieved from the database and the age of one `Person` is updated. Finally, that `Person` is deleted from the database.

```
//C#
using System;
using System.IO;
using com.db4o;

namespace com.db4o.dg2db4o.chapter5;
{
    public class CompleteExample
    {
        public static void Main(string[] args)
        {
            File.Delete("C:/complete.yap"); // reset database
            Db4o.Configure().MessageLevel(0); // 0=silent, 3=loud
            ObjectContainer db = Db4o.OpenFile("C:/complete.yap");
            try {
            {
                db.Set(new Person("Gandhi", 79));
                db.Set(new Person("Lincoln", 56));
                ObjectSet result = (ObjectSet) db.Get(new Person());
                ListResult(result); // get all

                Person p = new Person();
                p.Name = "Gandhi";
                ObjectSet result2 = (ObjectSet) db.Get(p);
                Person p2 = (Person) result2.Next();
                p2.Age = 90; // Increase Gandhi's age again
                result2.Reset(); // reset the ObjectSet cursor
                ListResult(result2);

                db.Delete(p2); // Remove the Gandhi dataset
                ObjectSet result3 = (ObjectSet) db.Get(new Person());
                ListResult(result3); // get all
                Console.ReadLine();
            }
```

```
            finally {
            {
                db.Close();
            }
        }

        private static void ListResult(ObjectSet res){
        {
            while(res.HasNext())
            {
                Console.WriteLine(res.Next());
            }
            Console.WriteLine ("---------------");
        }
    }
}


package com.db4o.dg2db4o.chapter5;

import java.io.File;
import com.db4o.Db4o;
import com.db4o.ObjectContainer;
import com.db4o.ObjectSet;

public class CompleteExample
{
    public static void main(String[] args)
    {
        new File("C:/complete.yap").delete();  // reset the database
        Db4o.configure().messageLevel(0); // 0=silent, 3=loud
        ObjectContainer db = Db4o.openFile("C:/complete.yap");
        try
        {
            db.set(new Person("Gandhi", 79));
            db.set(new Person("Lincoln", 56));
            ObjectSet result = (ObjectSet) db.get(new Person());
            listResult(result); // get all

            Person p = new Person();
            p.setName("Gandhi");
            ObjectSet result2 = (ObjectSet) db.get(p);
            Person p2 = (Person) result2.next();
            p2.setAge(90); // Increase Gandhi's age
            result2.reset();
            listResult(result2);
```

```
            db.delete(p2); // Remove the Gandhi object
            ObjectSet result3 = (ObjectSet) db.get(new Person());
            listResult(result3); // get all
        }
        finally
        {
            db.close();
        }
    }

    public static void listResult(ObjectSet result)
    {
        while (result.hasNext())
        {
            System.out.println(result.next());
        }
        System.out.println("---------------");
    }
}
```

The output should look like this:

```
[Lincoln;56]
[Gandhi;79]
---------------
[Gandhi;90]
---------------
[Lincoln;56]
---------------
```

Getting Started with the ObjectManager

Many database users prefer to use a graphical database administration tool such as MySQL Administrator or TOAD (`www.toadsoft.com`) rather than relying on the command line. db4o also provides a graphical tool for browsing, running queries, and editing primitive types, known simply as the ObjectManager. This section gives a brief overview of using the ObjectManager, while a fuller guide can be found in Appendix A.

The db4o administration tool is a little different from most, because managing tables strongly differs from managing objects. Most developers are used to browsing tables, but browsing huge object tree structures is a new experience for most of us.

The ObjectManager is a Java application, but through the magic of something that db4objects calls the *generic reflector*, it can also browse databases containing .NET objects. ObjectManager is

available for download from the Download Center area of the db4objects website. The download consists of a zip archive, which contains a folder called `lib`, as shown in Figure 5-4, which in turn contains the libraries that make up the application. For use in Windows environments, there is also a simple `.bat` script that starts a class from the `objectmanager.jar`. On Linux you have to make sure that the bin directory of your JDK is in your path, and then run the application with the command `java -jar objectmanager.jar`.

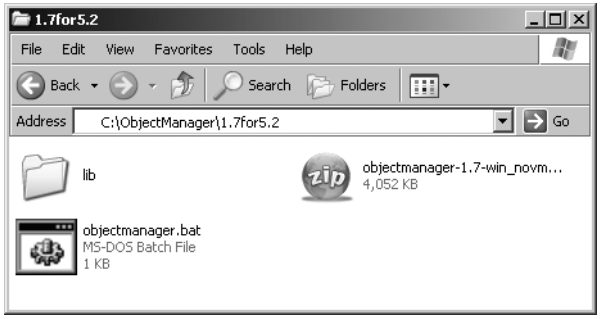

Figure 5-4. *The ObjectManager directory after you extract the zip file contents*

■ **Note** Currently there is no Windows `.exe` version of the ObjectManager. If you really wish to avoid Java, then you should have a look at `http://db4oboobrowser.sourceforge.net` for a demo or `http://sourceforge.net/projects/db4oboobrowser` for a download. This alternative db4o database browser is written in a scripting language that can be used on top of the .NET CLI.

Once you have started the ObjectManager, you can open a database using one of the following methods:

- Select File ➤ Open and choose your `.yap` file. Take care here: if you named your database using a different ending (e.g., `mydata.db`), then you need to change the file type in the file selection box.
- If you choose Open Encrypted instead, you can open an encrypted file.
- You can open a connection to a db4o database that is running in client/server mode. In this case, you need to enter authentication and host/port information. In Chapter 8 you will see how to run a db4o server.

Figure 5-5 shows the ObjectManager browsing a database created with the code in this chapter. On the left side you see all the objects in the database, just one in this case. If you have a bigger database with deep objects, you can open the object tree. If you select an entry, you will see information about that object on the right side. You can also edit the values of primitive attributes. For example, you can directly change Lincoln's age in the database.

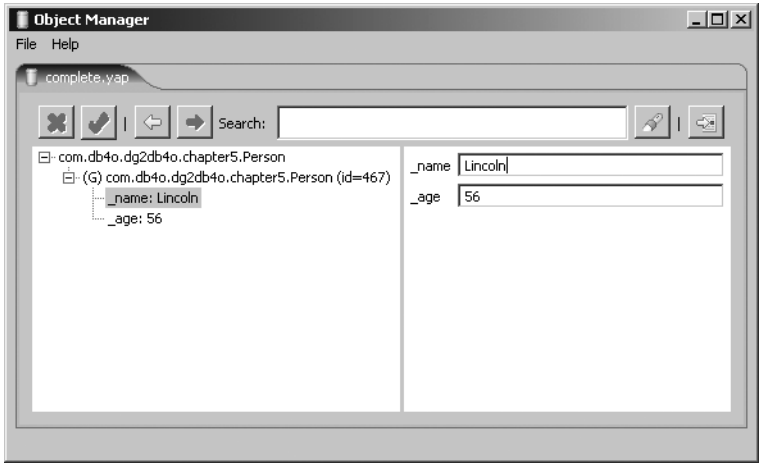

Figure 5-5. *ObjectManager showing our small example database*

Note that the objects show their database OID, which in most cases you don't need to care about. Every object is listed as a top-level object, and also as a part of the tree if it is referenced by another object.

The object in Figure 5-5 has the symbol (`G`) next to it, which signifies that this object was read by the generic reflector. This means that the ObjectManager doesn't have access to the original class definition. You can help the ObjectManager in displaying the object more naturally (e.g., showing the real objects IDs) by providing the original C# or Java classes. The File ➤ Preferences menu option allows you to enter either the directory or the library (`.dll` or `.jar`) files that contain the classes.

Using the ObjectManager to Query Your Objects

It can be difficult to find specific objects in a large object tree, so the ObjectManager allows you to filter the objects with queries. You can start a query using one of the following methods:

- Double-click on a top-level class in the tree view on the left side. Choosing a class deeper in the tree will do nothing.
- Click the New Query button on the very top right side.
- Choose Query from the File menu.

Each action opens a new tab that allows you to enter the query and view the results. Figure 5-6 shows an example of a query tab filtering a larger database with complex objects. You define the query by filling out fields of your class, using the symbols shown in Table 5-2 for the comparisons.

Table 5-2. *db4o ObjectManager Query Constraints*

Constraint	Description
`=`	Identity
`id`	The internal db4o object ID
`>`	Greater than
`<`	Smaller than
`~`	Like

To start the query just click the Run button on the upper-left side of the manager (the white arrow in the green circle). The tree view on the left shows the filtered results your query constraints have selected.

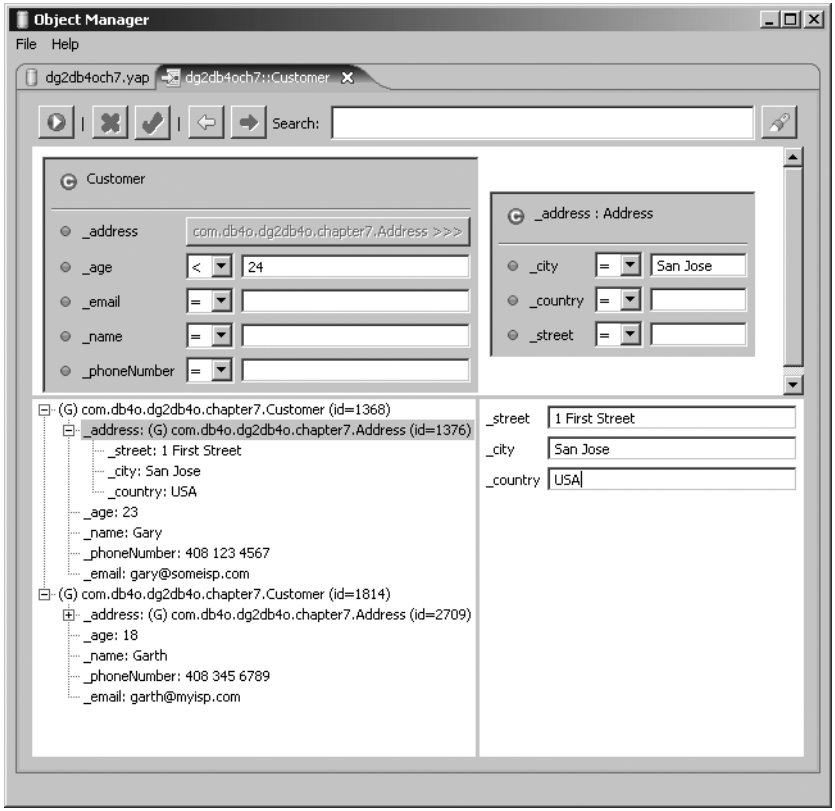

Figure 5-6. *ObjectManager running a complex query*

Troubleshooting

db4o is a relatively simple program, and there are not too many configurations that can go wrong or produce errors in your application. Nevertheless, there are some common mistakes you can make that can generally be avoided quite easily. Some of the problems you are most likely to encounter are listed here. Note that some of these issues will become clearer to you as you read the later chapters in Part II of this book.

- **db4o.jar is not in your classpath / dll not found**: This error can easily be detected when you are developing in your IDE because the compiler will simply not know the class `Db4o` and will in turn display error messages. A little more difficult would be the case when you don't use an IDE but where you are deploying your database solution (using db4o). In a standalone environment you would receive "class not found" messages when calling db4o methods. In an application server environment, such as Tomcat, you need to make sure that the DLL or JAR is put in the directory that the server uses for libraries.

- **The class is not in your classpath**: The objects in a database generally don't make sense to an application unless the real class (e.g., `Person`) appears in the classpath. The exception is the case when the generic reflector is used—as you have seen, ObjectManager can use this if it doesn't have the class files.

- **The database file is locked**: This is a common problem during development. Imagine the case where your program has a bug, and throws an unhandled exception while an ObjectContainer is open (this *will* happen to you). You correct the bug and run your program again. You get a database file locked exception. Why? It is simply because you have an old process running that has locked the database. So if you switch to the debugging mode, you can usually spot that there are old processes or threads running. Simply kill them, and restart your program in a fresh environment and the problem will disappear.

- **Changes to your class schema**: This is of course a difficult topic and will be discussed along with other advanced features in Chapter 12. Some cases of schema changes are handled quite simply, however. For example, if you add fields or remove fields, db4o ensures that you have to do nothing and can continue to work. Chapter 12 shows you how to refactor your database in more complex ways.

- **Double objects entered**: This error is very common if you are not careful. If you get an object from the database and then close the transaction, you lose the reference to the actual stored object. If you then open a new transaction on the same database file and save the in-memory object again, then you get two identical objects of the same kind. It is also possible to get similar problems in a networked environment. This means that you sometimes have to be aware of the object identity (look for `getID`), which is discussed in Chapters 7 and 10.

- **Objects in my object are are retrieved with null**: **object attributes**: A deep object is one where some attributes are themselves objects. It is possible to have a situation where you correctly retrieve your top-level object, but the object attributes are not retrieved. Chapters 7 and 10 describe the levels that determine how deep objects are stored, retrieved, and updated.

Summary

In this chapter you have learned that working with an embedded database like db4o is very different compared to working with a typical RDBMS. You don't need to create a database schema by hand, and you don't start the database. Instead, you can simply include the db4o library in your classpath, open an `ObjectContainer`, and start saving your objects, with just a few lines of code.

You have stored, retrieved, updated, and deleted objects, so you now understand the basic operations in db4o. The following chapters will guide you through the more complex topics such as advanced queries, transactions, client/server mode, and much more.