Introduction

If you are working on software which accesses a Paradox database, you are probably using BDE (Borland Database Engine). This article shows how to use Paradox database files and read some data from them directly. You can also use the primary index to find specific records you need. This could be useful if BDE doesn't cooperate with you or there is another reason why you cannot or don't want to use BDE.

Background

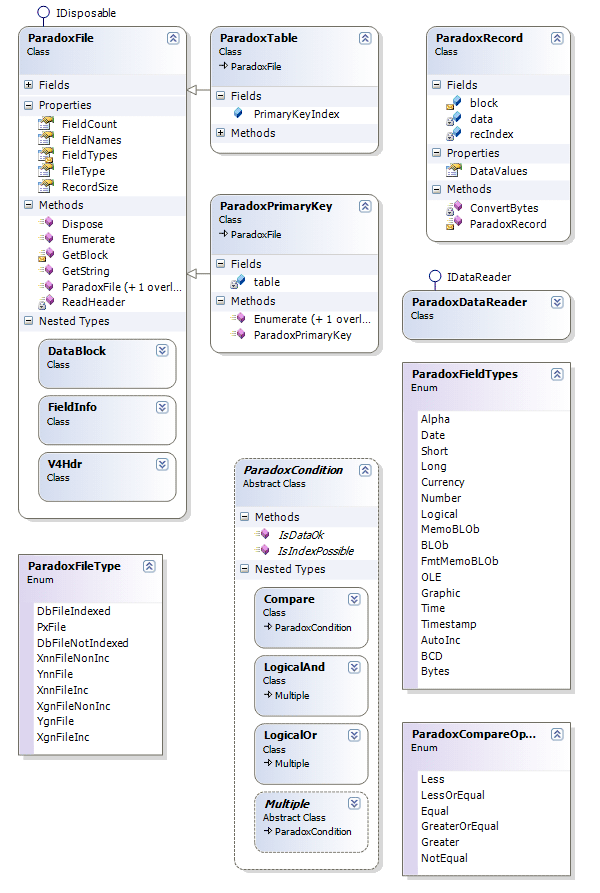
Every developer sometimes has to deal with some kind of black-box - piece of technology which is not opened, and can be accessed only from some external interface, but there is a problem - the black-box is quite mysterious to us - one time it works, another time it doesn't, and whatever you do, it seems to be having its own moods. And then I always wish it would be opened so I could debug it, trace into it, and look at its teeth. Are you also a developer who has Reflector between quick links? Then you will probably understand why I started to search for some .NET native Paradox database reader. I couldn't find any, but I have found some Paradox format specifications, so I decided to write one myself and make it public for others who could use it. Many thanks to Randy Beck and Kevin Mitchell for sharing the [Paradox internal structure](http://www.randybeck.com/paradoxformat.shtml), and because their materials are online, I will focus my efforts to describing my own code.

Paradox splits data to files using database objects - every table has its own file, and every index too. Data files and indexes have very similar structure, so we can handle them both in one class - ParadoxFile. BLOB values are also stored in their own files, but I haven't implemented the structure.

* ParadoxFile - base class working with common structures from Paradox data files and indexes
* ParadoxFile.DataBlock - represents one block of data
* ParadoxFile.FieldInfo - data type of the field
* ParadoxFile.V4Hdr - structure which is present only in certain Paradox files/versions
* ParadoxTable - represents table data file
* ParadoxPrimaryKey - represents table index
* ParadoxFileType - enum with all file types
* ParadoxFieldType - enum with all data types
* ParadoxRecord - represents a data record
* ParadoxDataReader - standard IDataReader implementation

The index file is, in fact, simply a table with indexes of datablocks and related indexed values. You can read it directly, but I also created a mechanism which browses through the index tree for you and picks up the desired record. You will have nothing more to do but specify a condition which you would like to apply on your data.

* ParadoxCondition - base class for conditions which can be used for searching in index data
* ParadoxCondition nested classes - various condition implementations
* ParadoxCompareOperator - enum with supported compare operators (==, !=, <, <=, >, >=)



Using the code

One way to go is just open a table and start enumerating. This is done by the following piece of code. We will create a ParadoxTable object and use the Enumerate method to traverse through records. Data is retrieved synchronously as needed, so if we stop reading after a few records, no redundant read operations are taken.

Hide   Copy Code

var table = new ParadoxTable(dbPath, "zakazky");

var recIndex = 1;

foreach (var rec in table.Enumerate())

{

Console.WriteLine("Record #{0}", recIndex++);

for (int i=0; i<table.FieldCount; i++)

{

Console.WriteLine(" {0} = {1}", table.FieldNames[i], rec.DataValues[i]);

}

if (recIndex > 10) break;

}

There are, of course, scenarios where this simple method will not suffice. Sometimes we also have to read some data in the middle of a large database, so we need to use an index to minimize disk operations. In the next sample, we will use a primary key index to find records with key values in the range between 1750 and 1760. At first, the index file has to be opened, then we will create a condition composed from two compare operations. We can browse data by calling the Enumerate method on the index. In this case, we will use theParadoxDataReader class with the the IDatareader implementation so we don't need to rewrite a lot of code if we have used BDE before.

Hide   Copy Code

var index = new ParadoxPrimaryKey(table, Path.Combine(dbPath, "zakazky.PX"));

var condition =

new ParadoxCondition.LogicalAnd(

new ParadoxCondition.Compare(

ParadoxCompareOperator.GreaterOrEqual, 1750, 0, 0),

new ParadoxCondition.Compare(

ParadoxCompareOperator.LessOrEqual, 1760, 0, 0));

var qry = index.Enumerate(condition);

var rdr = new ParadoxDataReader(table, qry);

recIndex = 1;

while (rdr.Read())

{

Console.WriteLine("Record #{0}", recIndex++);

for (int i = 0; i < rdr.FieldCount; i++)

{

Console.WriteLine(" {0} = {1}", rdr.GetName(i), rdr[i]);

}

}

Limitations

* read-only solution
* not all data types are supported
* no SQL, LINQ, or other complex queries
* you should not work with live database during writing

Source: <http://www.codeproject.com/Articles/101014/Paradox-database-native-NET-reader>