Random Access Database in Free Pascal

April 7, 2015

The Random Access Database (TRaDB) class is a random access file implementation written in Free Pascal. In the old days Turbo Pascal as well as QuickBASIC and even Visual Basic up until version 6 supported random access files. Free Pascal still supports random access files but in a procedural way and not object oriented. This file type can be used to create fixed length record files. In the old QuickBASIC days we could easily setup User Defined Types (structures) to set up records:

```
Type Customer
  Name As String * 46
  ID As Integer
End Type

Dim c As Customer

Open "customers" For Random As #1 Len = Len(c)
  Put #1, , c
Close #1
```

In most programming languages this feature, along with fixed length strings is no longer available. Instead you are supposed to use one of the available front-end database solutions such as SQL, Oracle or other build-in databases. True, these solutions have much more to offer than simple fixed-length records. However, some projects do not at all require any of the advanced features, so if fixed-length records is all you need, well too bad!

Luckily, Free Pascal still supports random access files, but as I said, not in an object oriented way. However, with the ability to setup records and move blocks of data in memory, it is not so difficult to setup a random access database class. TRaDB is just that, and it allows for the basic database operations: adding, editing and deleting records. It also features an encryption algorithm to encrypt your records.

How to use TRaDB

Create a new project and add the unit uRaDB to 'uses' at the top of your main unit. Create a record type, e.g. `TPerson = packed record`. **Always use packed records with TRaDB!** Create two variables, one of type TRaDB, e.g. `PersonDB: TRaDB;` and one of type TPerson, e.g. `Person: TPerson`.

☞ *Add a field 'ID' to your record. This allows you to find your record at any time.*

Now you can set up the database:

```
PersonDB := TRaDB.Create;

PersonDB.Name := 'Persons';
PersonDB.FileName := 'persons.rdb';
PersonDB.RecordSize := SizeOf(Person);
```

... and open it for read/write operations:

```
if PersonDB.Open then
  ; // database successfully opened
```

Adding records

To add a record to the database, simply define each field of the record Person and add it to the database using the AddRecord function:

```
if PersonDB.FreeRecord then
  begin
    Person.ID := PersonDB.RecordID; // save ID with record
    if PersonDB.AddRecord(@Person, Person.ID) then
      ; // record successfully added
  end
else
  ; // error
end if
```

In the above example I assumed there is a record field called ID of type longint to be saved with the record. If you want to save your record ID in another way, you can simply do:

```
If PersonDB.AddRecord(@Person) then
  YourID := PersonDB.RecordID;
```

When you do any operation like adding, updating or deleting a record with TRaDB, you refer to your record with `@Person` which is the pointer to your record data in memory.

Retrieving records

To retrieve a record from the database file , use the function GetRecord:

```
if PersonDB.GetRecord(@Person, Person.ID) then
  // record loaded
else
  ; // error
```

Updating records

To update a record, use the function UpdateRecord:

```
if PersonDB.UpdateRecord(@Person, Person.ID) then
  // successfully updated
else
  ; // error
```

Deleting records

To delete a record, simply use the function DeleteRecord:

```
if PersonDB.DeleteRecord(Person.ID) then
  // successfully deleted
else
  ; // error
```

Note that record data are not actually wiped from your database file. Instead only the deleted flag is set for that specific record space. When adding a new record, AddRecord will use the first available record space to store the data. It actually uses the function FreeRecord to accomplish this. In our first example above, we called FreeRecord ourselves and let AddRecord know we already have an ID for our new record. When you use FreeRecord and subsequently omit your obtained ID when calling AddRecord, then FreeRecord will be called twice (not a good idea).

Populating lists

The following example shows how to populate a list:

```
function LoadPersons: boolean;
var
  i: longint;
  id: TItemValue;
begin
  for i := 1 to PersonDB.RecordSpaces do
    begin
      if PersonDB.GetRecord(@Person, i, true) then
        begin
          id := TItemValue.Create;
          id.Value := Person.ID;
          List.AddItem(PersonName, id);
        end
      else if PersonDB.Error then
        exit(false);
    end;
  result := true;
end;
```

In this example I have used a class called TItemValue to store each record's id with its associated list item. This way we know what record is referred when a user clicks an item in the list. In order to do this you need to setup a class like this:

```
TItemValue = class(TObject)
  Value: longint;
end;
```

If you use this technique, make sure you dereference all objects before freeing the list! You can easily do this by means of a simple procedure:

```
procedure ClearList;
var
  i: integer;
begin
  for i := 0 to List.Count - 1 do
    List.Items.Objects[i].Free;
  List.Clear;
end;
```

Note that the function GetRecord has a third parameter called 'IgnoreDeleted', which I have set to 'true'. This prevents a deleted record along the way from raising the exception 'invalid record id'. You only need this parameter when populating lists.

Closing the database

Before closing the application, you should close the database:

```
PersonDB.Close;
```

Encryption

TRaDB supports encryption of records. The encryption algorithm used is RC4. In order to use encryption, you either provide your own encryption key:

```
PersonDB.EncryptionKey := MyKey;
```

or you set the encryption flag:

```
PersonDB.Encryption = true;
```

An encryption key tells TRaDB to encrypt/decrypt records with a user defined key. The encryption key could be a password provided by a user, or a hidden predefined key. Keys longer than 256 bytes will be truncated to 256 bytes. Setting the encryption flag tells TRaDB to use its built in default key, which is of course less secure.

Counting records

In order to keep track of the number of records stored in the database, TRaDB maintains two values: Records and RecordSpaces. We have already seen the second one in the example where we populated a list. The difference between the two is that Records tells us how many *occupied* records are stored in the database file, whereas RecordSpaces gives us the number of all record spaces, both occupied and deleted. With these to values we can easily find out how many deleted records the database contains:

```
DeletedRecords := RecordSpaces - Records
```

You can also use RecordSpaces to add a record without FreeRecord searching for the next available record space, for example if you want to be able to retrieve deleted records or to keep deleted records' id's from being re-used:

```
AddRecord(@Person, PersonDB.RecordSpaces + 1);
```

Error handling

TRaDB's error handling is such that messages are being displayed from within its functions. We have already seen the boolean variable Error in the populating list example. If Error is set to 'true' then ErrorCode and ErrorMessage will give you more information about the error. But remember that the message in ErrorMessage has already been displayed. Usually you only need to test for a function's result:

```
if not AddRecord(@Person) then
  // do something
```

Optimization

When designing records, you may want to consider the record size for some optimization and speed. A record size of 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, etc bytes will provide optimal storage in memory and on disk. If you choose to do so, beware that TRaDB adds a one-byte delete flag to each record. So in order to create a 512 byte record, the record should actually be 511 bytes long. And remeber: **ALWAYS USE PACKED RECORDS**.

Comments, suggestions and bug reports are welcome!

Frank Hoogerbeets
e-mail: info@mematis.com