Dainty Documentation

Mark van Renswoude

Sep 13, 2020

Table of contents

CHAPTER 1

Introduction

Dainty is a simple object mapper for Delphi.

Heavily inspired by Dapper, Dainty aims to provide a lightweight layer to map objects from a TDataSet descendant or to TParams. It is intentionally not a fully fledged ORM framework.

Dainty has been written and tested primarily in Delphi XE2 and 10.2.

1.1 Key features

- Read rows from a DataSet into objects
- Fill TParams using an object
- Helper methods for single row results
- Object mapping cache to reduce runtime RTTI overhead

CHAPTER 2

Getting started

2.1 Including the source

Dainty comes with runtime packages for a few Delphi versions which you can build to use the dcu's in your project. There is no designtime package or components, so you can also include the source directly if desired in which case you need both Dainty.pas and Dainty.Converter.Default.pas.

Dainty provides two ways to use it. The easiest way is to use the included class helpers and directly call Dainty's methods on any TDataSet or TParams instance. This is the method all the examples below will use. If however you have a conflict in class helpers and can't use them, you can instead call the same methods directly using the TDainty class instead and pass the dataset or params as the first parameter.

2.2 Reading a dataset

Use Rows to iterate through a dataset and map each row to an object.

```
uses
  Dainty;


type
  TCustomerRow = class
  public
    CustomerID: Integer;
    FullName: string;
    Active: Boolean;
  end;


var
  query: TIBQuery;
  customer: TCustomerRow;
```

(continues on next page)

(continued from previous page)

```
begin
  query := TIBQuery.Create(nil);
  try
    query.Database := MyDatabase;
    query.Transaction := MyTransaction;
    query.SQL.Text := 'select CustomerID, FullName, Active from Customer';
    query.Open;

    for customer in query.Rows<TCustomerRow> do
    begin
      { You can read the customer properties here. Note that you do not have ownership
        of the objects when using Rows<> and must not keep a reference, as they are
        destroyed during and after the loop.

        The author is of the opinion that 'database' objects should not be used as␣
↪business
        objects but instead mapped to and from, to provide separation of the domains.
        However, Dainty does not enforce this and you can use query.List<> instead to
        get ownership of the list. }
    end;
  finally
    FreeAndNil(query);
  end;
end;
```

Note that the customer object is destroyed right before the next iteration or after the loop is finished. If you want to keep a reference to the customer object, use List<> instead to get a list of objects which you have to Free yourself.

If you only need one row there are a few helpers you can use:

GetFirst will retrieve one row from the dataset and expects at least one row to be present. GetSingle is similar, but in addition it verifies that there is exactly one row and not more. Both have an OrDefault version which will not throw an exception but return nil instead if the requirements are not met. An example:

```
var
  query: TIBQuery;
  oldestCustomer: TCustomerRow;

begin
  query := TIBQuery.Create(nil);
  try
    query.Database := MyDatabase;
    query.Transaction := MyTransaction;
    query.SQL.Text := 'select CustomerID, FullName, Active from Customer order by Age␣
↪desc rows 1';
    query.Open;

    oldestCustomer := query.GetFirstOrDefault<TCustomerRow>;
    try
      { ... }
    finally
      FreeAndNil(oldestCustomer);
    end;
  finally
    FreeAndNil(query);
  end;
```

(continues on next page)

(continued from previous page)

```
end;
```

Note that you must destroy the resulting object yourself in this scenario.

For full control over the dataset position and the resulting object, GetRowReader<> is available.

2.3 Writing parameters

```
uses
  Dainty;


type
  TCustomerParams = class
  public
    FullName: string;
    Active: Boolean;
  end;


var
  query: TIBQuery;
  customParams: TCustomerParams;

begin
  query := TIBQuery.Create(nil);
  try
    query.Database := MyDatabase;
    query.Transaction := MyTransaction;
    query.SQL.Text := 'select CustomerID, FullName, Active from Customer where␣
↪FullName = :FullName and Active = :Active';

    customerParams := TCustomerParams.Create;
    try
      customerParams.FullName := 'John Doe';
      customerParams.Active := True;

      query.Params.Apply(customerParams);
      query.Open;

      { ... }
    finally
      FreeAndNil(customerParams);
    end;
  finally
    FreeAndNil(query);
  end;
end;
```

2.4 FieldName and ParamName attributes

TODO: explain how these are used

Note that ParamName and FieldName are aliases for the same attribute and can be used interchangeably. Pick whichever one makes the most sense for the object in question.