Smarter code with databases and data aware controls

Jeroen Pluimers
better office benelux
jpluimers@better-office.com

Smarter code

for … in with TDataSets

Business logic versus glue…

```
procedure TXokumDataModule.GetMinMaxAbonneeNummerOldStyle(
  var MinAbonneenummer: Integer;
  var MaxAbonneeNummer: Integer);
var
  WasActive: Boolean;
begin
  MinAbonneenummer := High(Integer);
  MaxAbonneeNummer := Low(Integer);
  WasActive := XokumClientDataSet.Active;
  XokumClientDataSet.Open;
  XokumClientDataSet.First;
  while not XokumClientDataSet.Eof do
  begin
    if XokumClientDataSetabonneenummer.Value > MaxAbonneeNummer then
      MaxAbonneeNummer := XokumClientDataSetabonneenummer.Value;
    if XokumClientDataSetabonneenummer.Value < MinAbonneeNummer then
      MinAbonneeNummer := XokumClientDataSetabonneenummer.Value;
    XokumClientDataSet.Next;
  end;
  if not WasActive then
    XokumClientDataSet.Close;
end;
```

There is lots of glue

- Glue sticks, but does it stick the right way?
  - Looping over datasets
  - Mixing UI code with business logic
  - Reacting in the UI on data changes
- Smarter code means:
  - Focusing on your business logic
    - At code time
    - At design time
  - by getting glue out of your way
- In the mean time we learn bits of Delphi

Bad solutions

- With
  - It bytes when you don’t expect it to

```
with MyEdit do
begin
  Caption := MyDataSetFirstName.Value;
end;
```

  - Eof without DataSet:

```
while not Eof do
begin
  // …
end;
```

- Next

```
while not Eof do
begin
  Sum := Sum + MyDataSetSalary.Value;
end;
```

Wouldn’t it be nice to…

```
procedure TXokumDataModule.GetMinMaxAbonneeNummer(
  var MinAbonneenummer: Integer;
  var MaxAbonneeNummer: Integer);
var
  Index: TDataSetEnumerationRecord;
begin
  MinAbonneenummer := High(Integer);
  MaxAbonneeNummer := Low(Integer);
  for Index in XokumClientDataSet do
  begin
    if XokumClientDataSetabonneenummer.Value > MaxAbonneeNummer then
      MaxAbonneeNummer := XokumClientDataSetabonneenummer.Value;
    if XokumClientDataSetabonneenummer.Value < MinAbonneeNummer then
      MinAbonneeNummer := XokumClientDataSetabonneenummer.Value;
  end;
end;
```

for … in

- for Win32, it was introduced in D2005
  - On lots of built-in data types
    - Arrays
    - Strings
  - On many data types
    - Lists, Collections, Trees,
    - Components, Actions, Menus, Fields,
    - many, many more …

Data types supporting for … in

- These data types
  - provide either of these:
    - `function GetEnumerator: T…Enumerator;`
    - `function GetEnumerator: IEnumerator;`
  - and the result provides these functions:
    - `constructor Create( const AObject: T…Object);`
    - `function GetCurrent: T…;`
    - `function MoveNext: Boolean;`
    - `property Current: T… read GetCurrent;`
- The compiler then recognizes it supports for … in

Helpers

- Introduced in Delphi to support .NET
  - The .NET class hierarchy differs from Win32 VCL
    In the .NET framework, VCL methods and properties were different or missing
- Helpers can make extensions at function level
  - Yes: methods and properties
  - No: instance data
- They also work in Delphi for Win32:
  - Class helpers since Delphi 2005
  - Record helpers since Delphi 2006

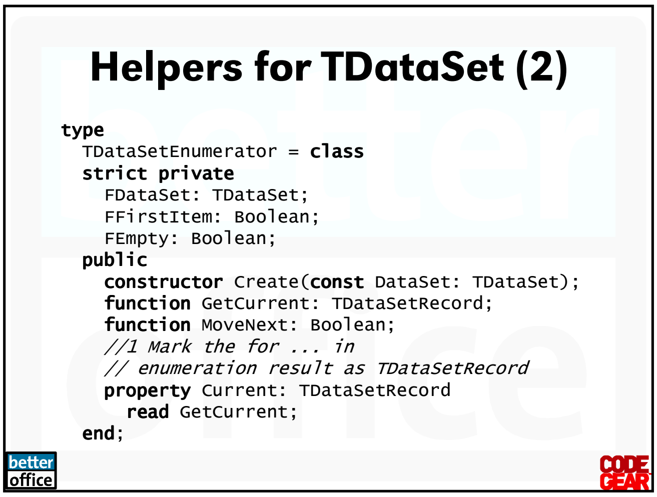
Helpers

- Helpers (class or record):
  - function as long as the helper is visible to the user
- So:
  - Helper in the same unit,
  - or helper in a unit in the uses list

Introduce GetEnumerator in a helper

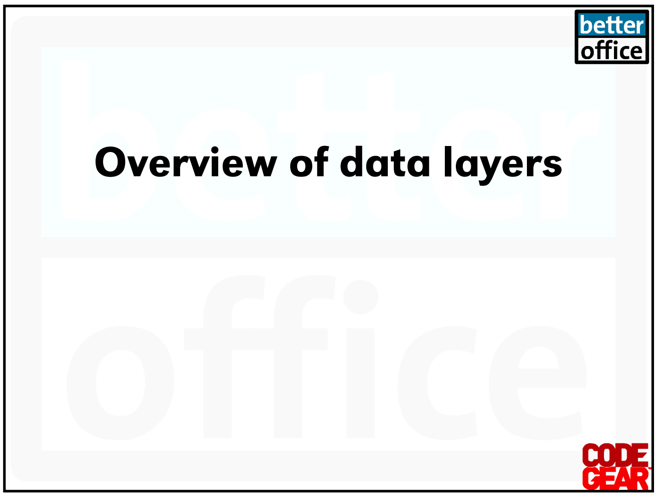
Helpers for TDataSet & TParams

```
type
  TDataSetHelper = class helper for TDataSet
  public
    //1 Support for ... in loop providing TDataSetRecord
    function GetEnumerator: TDataSetEnumerator;

    //1 Returns first field that matches,
    // if no matching field then exception
    function FieldByAnyName(
      const FieldNames: array of WideString): TField;
  end;

  TParamsHelper = class helper for TParams
  public
    function ParameterByAnyName(
      const ParameterNames: array of WideString): TParam;
  end;
```

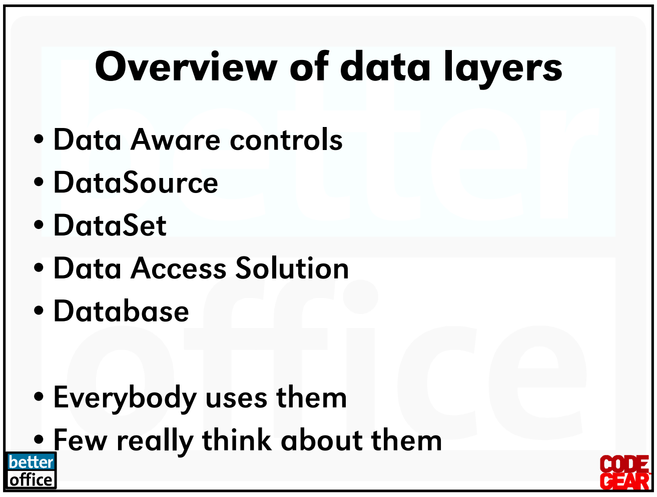


Helpers for TDataSet (2)

```
type
  TDataSetEnumerator = class
  strict private
    FDataSet: TDataSet;
    FFirstItem: Boolean;
    FEmpty: Boolean;
  public
    constructor Create(const DataSet: TDataSet);
    function GetCurrent: TDataSetRecord;
    function MoveNext: Boolean;
    //1 Mark the for ... in
    // enumeration result as TDataSetRecord
    property Current: TDataSetRecord
      read GetCurrent;
  end;
```



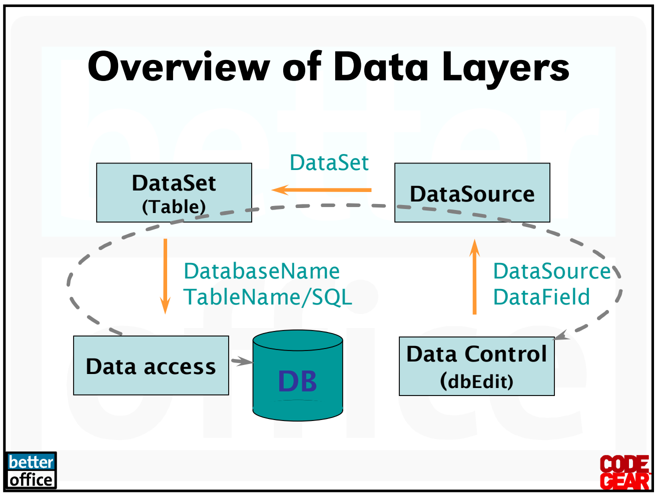
Overview of data layers



Overview of data layers

- Data Aware controls
- DataSource
- DataSet
- Data Access Solution
- Database

- Everybody uses them
- Few really think about them



Overview of Data Layers

DataSet
(Table)

DataSet

DataSource

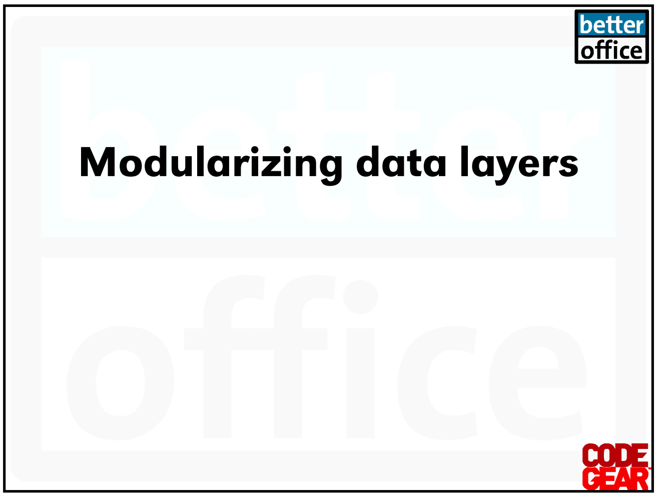
DatabaseName
TableName/SQL

DataSource
DataField

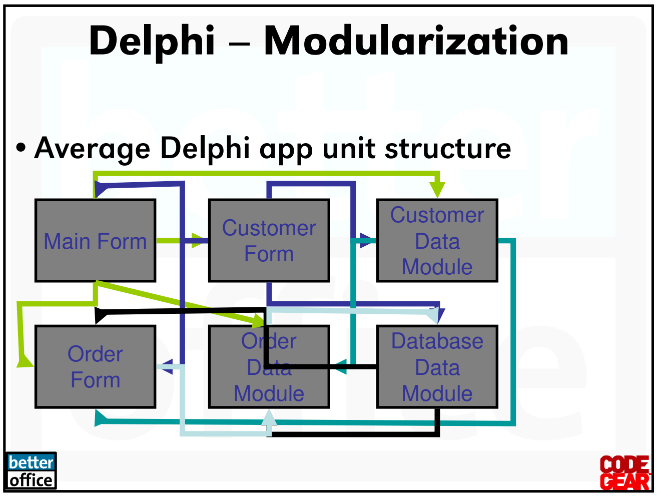
Data access

DB

Data Control
(dbEdit)



Modularizing data layers



Delphi – Modularization

- Average Delphi app unit structure

Main Form

Customer Form

Customer Data Module

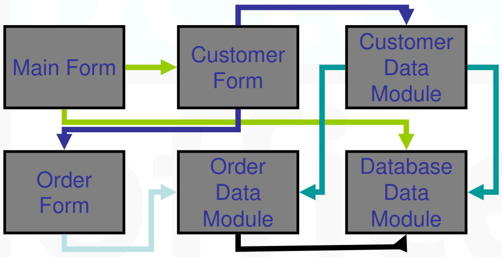
Order Form

Order Data Module

Database Data Module

Delphi – Modularization

- Good Delphi app unit structure

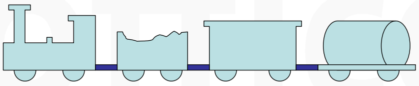


How come good is better than bad?

- Look for modularization in real life…
  - Houses - rooms
  - Cities - suburbs/blocks
  - People - digestion system
  - Parliament - parties
- Sometimes modularization works
- Sometimes it doesn't ☺

Modules are everywhere…

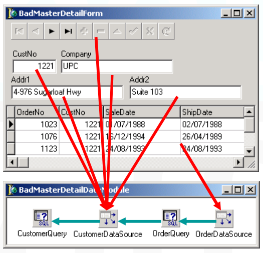
- So why does it work?
  - Internal Cohesion **HIGH**
  - External Coupling **LOW**
    - If also 'uniform': great!
    - If also 'directional': even better!



Modularization – database apps

- Let's apply this knowledge to our database apps
- Especially: where to put the DataSource…
- Where is your DataSource?

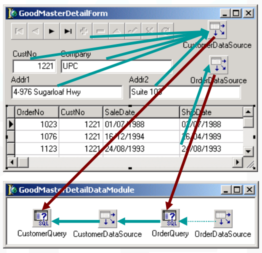
Modularization – DataSource



CustNo: 1221
Company: UPC
Addr1: 4-976 Sugarloaf Hwy
Addr2: Suite 103

OrderNo	CustNo	SaleDate	ShipDate
1023	1221	01/07/1988	02/07/1988
1076	1221	16/12/1994	26/04/1989
1123	1221	24/08/1993	24/08/1993

Bad 3 Internal Links
6 External Links



CustNo: 1221
Company: UPC
Addr1: 4-976 Sugarloaf Hwy
Addr2: Suite 103

OrderNo	CustNo	SaleDate	ShipDate
1023	1221	01/07/1988	02/07/1988
1076	1221	16/12/1994	26/04/1989
1123	1221	24/08/1993	24/08/1993

Good 6+2 Internal
2 External

Modularization – the datasource

- DataSource has **two** goals
  - Binding GUI controls
  - Providing Master-Detail relations
- GUI binding:
  put **DataSource** on **Form**
- MD-relations:
  put **DataSource** on **DataModule**

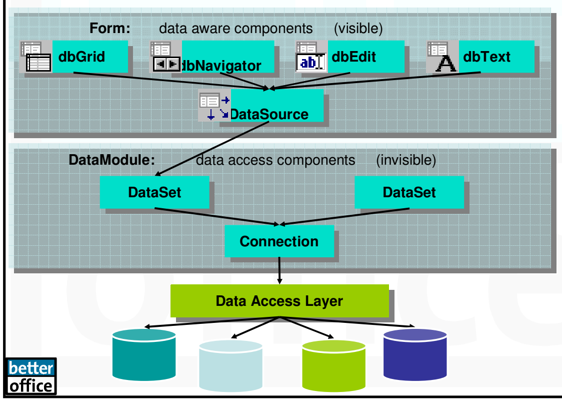
Modularization – gain

- Flexibility
  - Change of GUI
  - Change of Data Access
  - Re-use of modules across projects

Back to data layers overview

How does this apply to BDE/dbExpress/ADO/...?

Overview – Generic application



Form: data aware components (visible)

dbGrid dbNavigator dbEdit dbText

DataSource

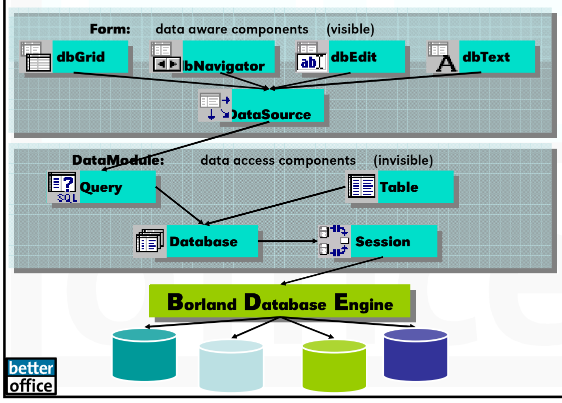
DataModule: data access components (invisible)

DataSet DataSet

Connection

Data Access Layer

BDE application



Form: data aware components (visible)

dbGrid dbNavigator dbEdit dbText

DataSource

DataModule: data access components (invisible)

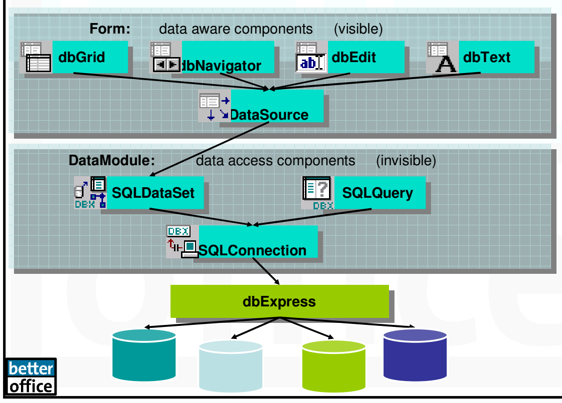
Query Table

Database Session

Borland Database Engine

---

dbExpress application



Form: data aware components (visible)

dbGrid dbNavigator dbEdit dbText

DataSource

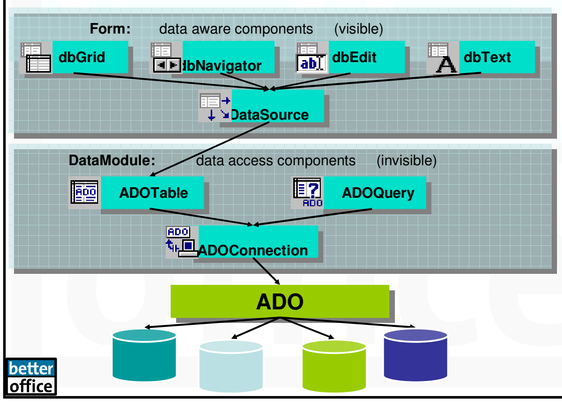
DataModule: data access components (invisible)

SQLDataSet SQLQuery

SQLConnection

dbExpress

ADO application



Form: data aware components (visible)

dbGrid dbNavigator dbEdit dbText

DataSource

DataModule: data access components (invisible)

ADOTable ADOQuery

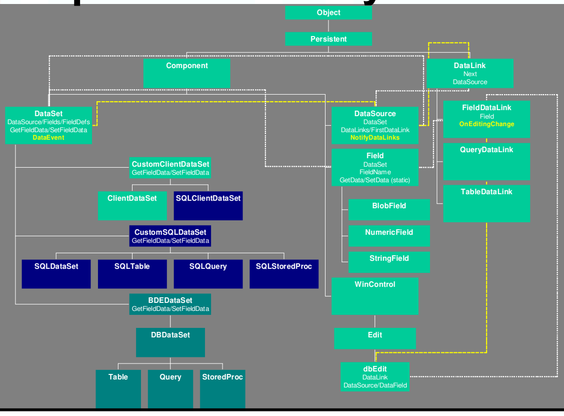
ADOConnection

ADO

Component hierarchy

Delphi 5 Object Hierarchy.pdf

Component hierarchy – overview



Object

Persistent

Component

DataSet
DataSource/Fields/FieldDefs
GetFieldData/SetFieldData
DataEvent

CustomClientDataSet
GetFieldData/SetFieldData

ClientDataSet

SQLClientDataSet

CustomSQLDataSet
GetFieldData/SetFieldData

SQLDataSet

SQLTable

SQLQuery

SQLStoredProc

BDEDataSet
GetFieldData/SetFieldData

DBDataSet

Table

Query

StoredProc

DataLink
Next
DataSource

DataSource
DataSet
DataLinks/FirstDataLink
NotifyDataLinks

Field
DataSet
FieldName
GetData/SetData (static)

BlobField

NumericField

StringField

WinControl

Edit

dbEdit
DataLink
DataSource/DataField

FieldDataLink
Field
OnEditingChange

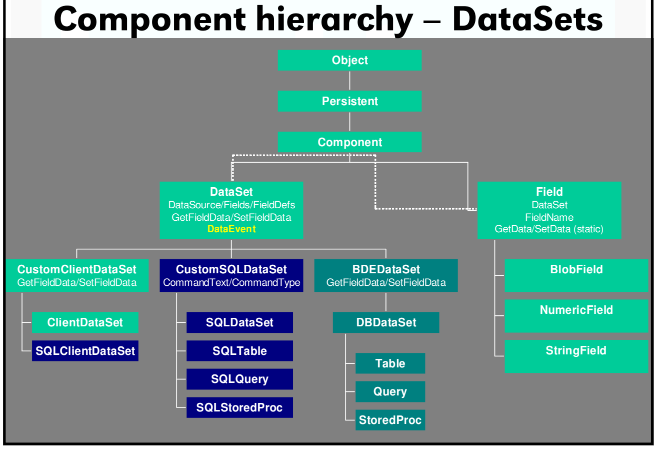
QueryDataLink

TableDataLink

DataSets: data binding

GetFieldData/SetFieldData
and more...

Component hierarchy – DataSets



Object

Persistent

Component

DataSet
DataSource/Fields/FieldDefs
GetFieldData/SetFieldData
DataEvent

Field
DataSet
FieldName
GetData/SetData (static)

CustomClientDataSet
GetFieldData/SetFieldData

ClientDataSet

SQLClientDataSet

CustomSQLDataSet
CommandText/CommandType

SQLDataSet

SQLTable

SQLQuery

SQLStoredProc

BDEDataSet
GetFieldData/SetFieldData

DBDataSet

Table

Query

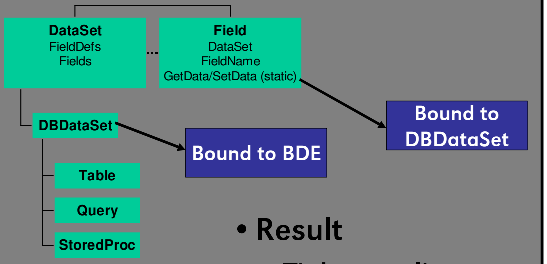
StoredProc

BlobField

NumericField

StringField

DataSet/Field – until D2



DataSet
FieldDefs
Fields

Field
DataSet
FieldName
GetData/SetData (static)

DBDataSet

Table

Query

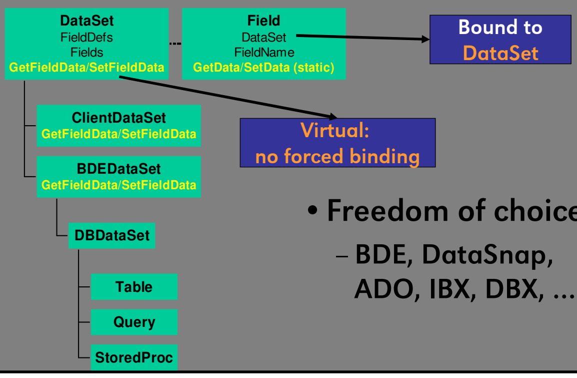
StoredProc

Bound to BDE

Bound to DBDataSet

- Result
  - Tight coupling with BDE

DataSet/Field – D3 and up



DataSet
FieldDefs
Fields
GetFieldData/SetFieldData

Field
DataSet
FieldName
GetData/SetData (static)

Bound to DataSet

ClientDataSet
GetFieldData/SetFieldData

BDEDataSet
GetFieldData/SetFieldData

Virtual:
no forced binding

DBDataSet

Table

Query

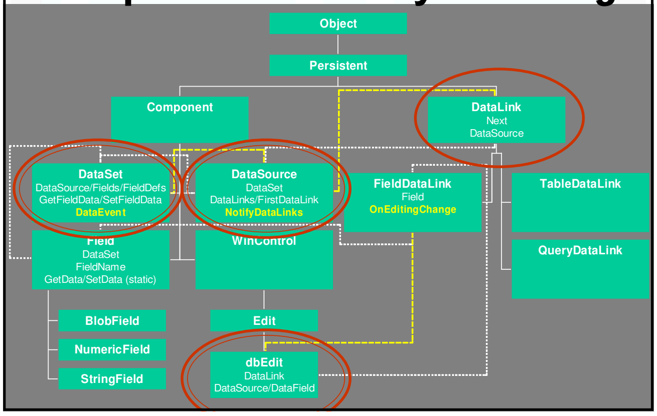
StoredProc

- Freedom of choice
  - BDE, DataSnap, ADO, IBX, DBX, ...

GUI: data binding

DataLinks

Component hierarchy – Binding



So: TDataLink is interesting

- Choose which TDataLink (or descendant to use)
  - TDataLink
    - TDataSourceLink
    - TDBCtrlGridLink
    - TDetailDataLink
      - TIBDataLink
      - TMasterDataLink
    - TFieldDataLink
    - TGridDataLink
    - THTTPDataLink
    - TListSourceLink
    - TMultiDimDataLink
    - TNavDataLink

Demo's

- TDbDisplayLabel
  - Shows the TField.DisplayLabel in a data aware control
  - Uses TFieldDataLink
- Fields editor
  - More than just adding TFields
  - Drag & Drop to other forms
  - Navigate through record

Demo's (2)

- TDataLinkReflector
  - Makes the virtual functions of a TDataLink available as Events
- TDataAwareControlController
  - Dynamically change the appearance of your data aware controls depending on the properties of the TField objects they bind to

---

Q & A

- Questions after the conference?
- Jeroen Pluimers
  jpluimers@better-office.com