Starting with Bold for Delphi/Bold for C++
Part 1: Introducing the basics

Anthony Richardson - anthony@viewpointsa.com
1 August 2002, Revision 1.1

Contents

Getting Started

Introduction

'Starting with Bold for Delphi' is a series of articles design to provide an introduction to the Bold for Delphi product and effective techniques for applying the Bold framework in the development of real world applications.

These articles are designed as an introduction. The Bold for Delphi product contains many components and is comprised of 1765 classes spread across over 665 units. The product contains many sub frameworks within these classes to support advanced development techniques and the application of industry best practices in the form of patterns, interfaces and modeling.

The design of Bold for Delphi is carefully layered to enable rapid adoption of core techniques with the gradual adoption of the more complex or sophisticated methods as required. These articles are designed to facilitate the transition from traditional programming to the core Bold for Delphi techniques.

Contacting Anthony Richardson

All feedback on this article is welcome:

Email: anthony@viewpointsa.com
Web: http://www.viewpointsa.com

Contacting BoldSoft

Getting help with using Bold for Delphi is available from BoldSoft.

Email: support@boldsoft.com
Web: http://www.boldsoft.com

Model Based Development

Introduction

The term model based development is used often when describing the Bold for Delphi product. The term refers to the process of developing your application directly from the class model, rather than traditional process of transforming an entity-relationship model into a table structure and then coding your application to the table structure.

Bold for Delphi applies Object Oriented flexibility and design to the persistence layer of your application making design easier. It is worth spending time on this upfront because this is a huge part of the productivity enhancement you will experience when using the Bold for Delphi architecture.

What is modeling?

A model is a description or view of something. Like a model aircraft is a miniature representation of a real aircraft, a software model is a representation of the domain that the software is designed for. It is possible to generate many different models for a particular software project. Popular software engineering methodologies describe models and activities at various stages of the software development lifecycle which aid in the understanding of the problem domain. Models also simplify communication between developers, users and other stakeholders. By creating the various types of models you can describe a complex software system in a series progressively more detailed set of views into the problem domain.

Traditional Entity-Relationship Modeling
Entity Relationship Modeling (ER Modeling) has traditionally been the method used to describe the business data for an application, the term data modeling is also used. Generally these models are restricted to portraying a static view of the data within a problem domain and fail to recognize the operations and interactions between entities.
After the completion of the ER model a database schema is developed. During this process the types of relationships portrayed in the ER model are transformed into a format that is suitable for the database. Certain types of relationships will result in extra tables being generated and the introduction of various primary and secondary keys. Where data may have been originally design with concepts like inheritance relationships, these models are 'flattened' to allow for the design to fit within the constraints of a relational database. This will tend to result in the loss of information from the model, not in terms of the information stored, but in the implied relationships and business rules of a fully normalized ER model.

Class Modeling
Class Modeling is a modern version of ER modeling that expands on the benefits of diagramming domain entities and their relationships and includes the interaction between those classes. Where ER Modeling is only modeling the data as required to be saved, the class model includes all domain entities and the operations performed by those entities. This allows for a much more accurate view of the business problem domain. By using class modeling and the various diagramming techniques available it is easier to build the correct software architecture earlier. The classes in a class model may also be referred to as business objects.

Business Objects

The term business object is used in many different ways and means different things to different people. Other terms used for business objects are domain objects and business classes. The process of storing and retrieving these business objects is often referred to as object persistence or object relational mapping. Frameworks that support the use of business objects are often referred to as object persistence frameworks.

Using business objects is the application of object-oriented techniques beyond coding and infrastructure into the realm of the real-world items (objects) being used by your software. An example of a using business objects is part of this article.

Unified Modeling Language (UML)

From the UML website, www.uml.org :

"The Unified Modeling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML was developed by Rational Software and its partners. It is the successor to the modeling languages found in the Booch, OOSE/Jacobson, OMT and other methods. Many companies are incorporating the UML as a standard into their development processes and products, which cover disciplines such as business modeling, requirements management, analysis & design, programming and testing."

UML is critical for the use of Bold for Delphi. The Bold for Delphi product includes its own UML modeling tool. Learning UML is easy and applying the full range of UML modeling techniques is a rewarding process.

The Example Application Class Model

This tutorial will build a simple address book application. The basic needs for our application are:

- Store contact information for people.
- Store contact information for companies.
- Allow for people to be associated with a company.

We are going to implement the following UML model for our address book application:
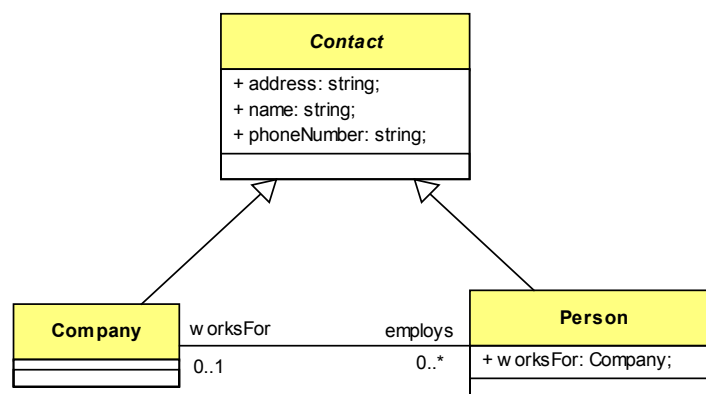
Lets start easy. This design is very basic and ignores requirements for complicated addresses and multiple phone numbers.

If you are not familiar with UML, the diagram shows three classes, represented by the rectangles. The class 'Contact' has its title displayed in italics to indicate it is an abstract class. The lines with the arrowheads indicate that both the 'Company' and 'Person' classes descend from the 'Contact' class.

The line between the 'Company' class and the 'Person' class indicates a relationship. Relationships contain a title, 'Employment', properties for each class, 'worksFor' and 'employs', as well as multiplicity specifiers, the '0..1' and '0..*'. These will be explained in greater detail soon.

The model shows the following business rules:

- A *Contact* has a name, address and phone number.
- A *Contact* is either a *Person* or a *Company*.
- A *Company* employs zero or more *Person*.
- A *Person* works for zero or one *Company*.

If this model were implemented in a RDBS more than likely compromises to the design would have to be accepted, with the responsibility for the enforcement of the rules moved to the application logic. For example depending on how the data is 'flattened' the entities Person and Company may be represented in a single table requiring the relationship type restrictions being implemented in application code.

There is danger when moving from what is essentially an Object Oriented design to a flat relational database design. The Object oriented design clearly portrays real world entities and models valid business rules via relationships and care must be taken to preserve the intentions of the design. Often this can be difficult or impossible,

resulting in the need for supplementary specifications. These document the business rules now required to be enforced in code.

In reality this transformation results in an implementation that loses the intent of the original model and may even drop off important business rules if careful documentation processes aren't followed. Indeed even if these rules are documented it is reliant on them being understood and implemented by the application programmer. As with all documentation it must also be manually maintained during the life cycle of the database.

"This is what BoldSoft terms the Executable Model Architecture (XMA)"

The ultimate solution is to use a medium that accurately expresses the data model and is executable in the application. Because the documentation becomes the code, and vice-versa, business rules are maintained and enforced during the entire life cycle of the database application.

This is what BoldSoft terms the Executable Model Architecture (XMA) and is the foundation of the Bold for Delphi product.

Building the Application

Introduction

It's time to start Delphi and ensure Bold for Delphi is installed. The following procedure should be complete enough for anyone that has used Delphi to follow easily. After each series of steps a description of what has been achieved is offered before moving on to the next steps.

Groundwork

To get started we need to setup a basic Delphi project with a data module.
1. Select *File|New Application* from the Delphi menu.
2. Save the Form unit as *MainForm.pas* and the project as *ContactManager.dpr*
3. Add a new Data Module by selecting *File/New* and selecting *Data Module* from the *New Items Dialog*.
4. Set the Data Modules name to *BusinessModule*.
5. Save the DataModule as *BusinessLayer.pas.*

Models, Handles and Persistence

The basics for a Bold for Delphi application are adding our Model to the application and the chosen persistence layer. To prepare the application to accept a model, do the following:
1. From the *Bold Handles* component tab add a *BoldModel, BoldSystemTypeInfoHandle and BoldSystemHandle* to the Data Module.
2. Link the *BoldModel* property from *BoldSystemTypeInfoHandle1* to *BoldModel1.*
3. Link the *BoldSystemTypeInfoHandle* property from *BoldSystemHandle1* to *BoldSystemTypeInfoHandle1.*
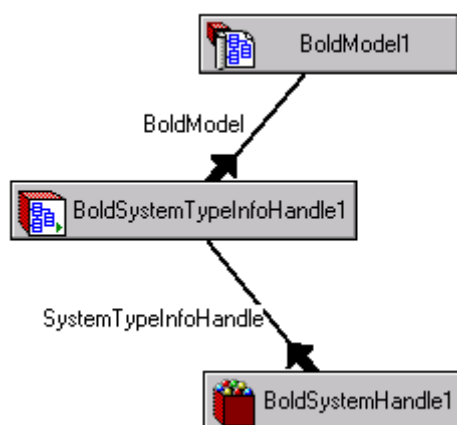

The *BoldModel* component will hold the model, that is the classes, relationships, constraints and types. Its role at design time is to hold the model and stores it as a string in the Delphi Form/Data module. At runtime it performs an intermediary transformation of the model into a format to be used by the *BoldSystemTypeInfoHandle* and the chosen persistence mechanism. The information stored in the BoldModel component can be thought of as metadata, in a similar way that you can query a database to provide table and field information. This component is generally used only at design time except for advanced operations not covered by this article.

The *BoldSystemTypeInfoHandle* component holds the runtime information needed by a *BoldSystemHandle*. The information is an efficient representation of the information in the UML-model, heavily optimized for speed. This component is generally not referenced directly in code, but rather acts as a source of information for other Bold for Delphi Components.

The *BoldSystemHandle* component is used to represent an entire system of domain elements, also known as an object-space. This component represents the available instances of objects in the model at runtime. This is the primary gateway between design time metadata and actual stored objects.

The components as they stand are fine for a system that doesn't need to save data (persist) between sessions. However few applications have value without saving information between each use. To enable a fast path to running a Bold for Delphi Application we will add an XML persistence component to the data module
1. From the *Bold Persistence* component tab add a *BoldPersistenceHandleFileXML* component to the data module.
2. Link the *BoldModel* property to the *BoldModel* component.
3. Link the *PersistenceHandle* property of *BoldSystemHandle* to the *BoldPersistenceHandleFileXML* component.
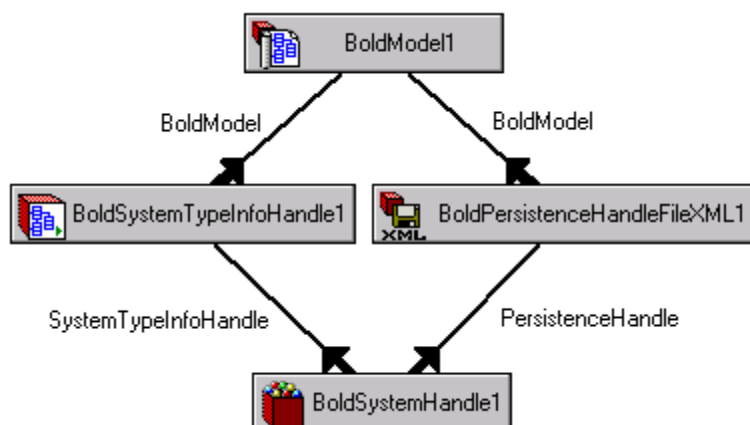
The *BoldPersistenceHandleFileXML* component allows the application to save and read objects from an XML file. This is a very handy component to use while you are still prototyping an architecture as is saves you from having to regenerate your database tables while still making significant changes to your model. Later on we will remove this component and use the BDE or Interbase components to allow use of a relational database. The design of the Bold for Delphi product allows us to do this without changing our application. This is called database independence. At that time we will also look closer at the SQL calls generated by Bold for Delphi, but for now we will forget this as it's not critical for using Bold for Delphi.

Basic configuration Changes

Initially in this example application we will use the following property values, set these in the Delphi property editor:

Component: `BoldSystemTypeInfoHandle1`
Property: `UseGeneratedCode`
Value: `False`

Later on we will enable this and work directly with the model using Object Pascal, but initially this is not required.

Component: `BoldPersistenceHandleFileXML1`
Property: `FileName`
Value: `Data.XML`

To begin with the XML file is hard coded, this of course could be set using code at runtime in a real application.

Component: `BoldSystemHandle1`
Property: `Active`
Value: `True`

This property results in the Bold for Delphi system opening the XML file and being ready for immediate use on application startup.

Building the Model

The next step is to build our model. Bold for Delphi comes with an inbuilt non-graphical UML modeling tool. It is accessed by double clicking on the *BoldModel* component.

Bold for Delphi UML Model Editor

The Bold for Delphi UML model editor (Figure 1) contains the application model information, data type information, and relational database mapping information.
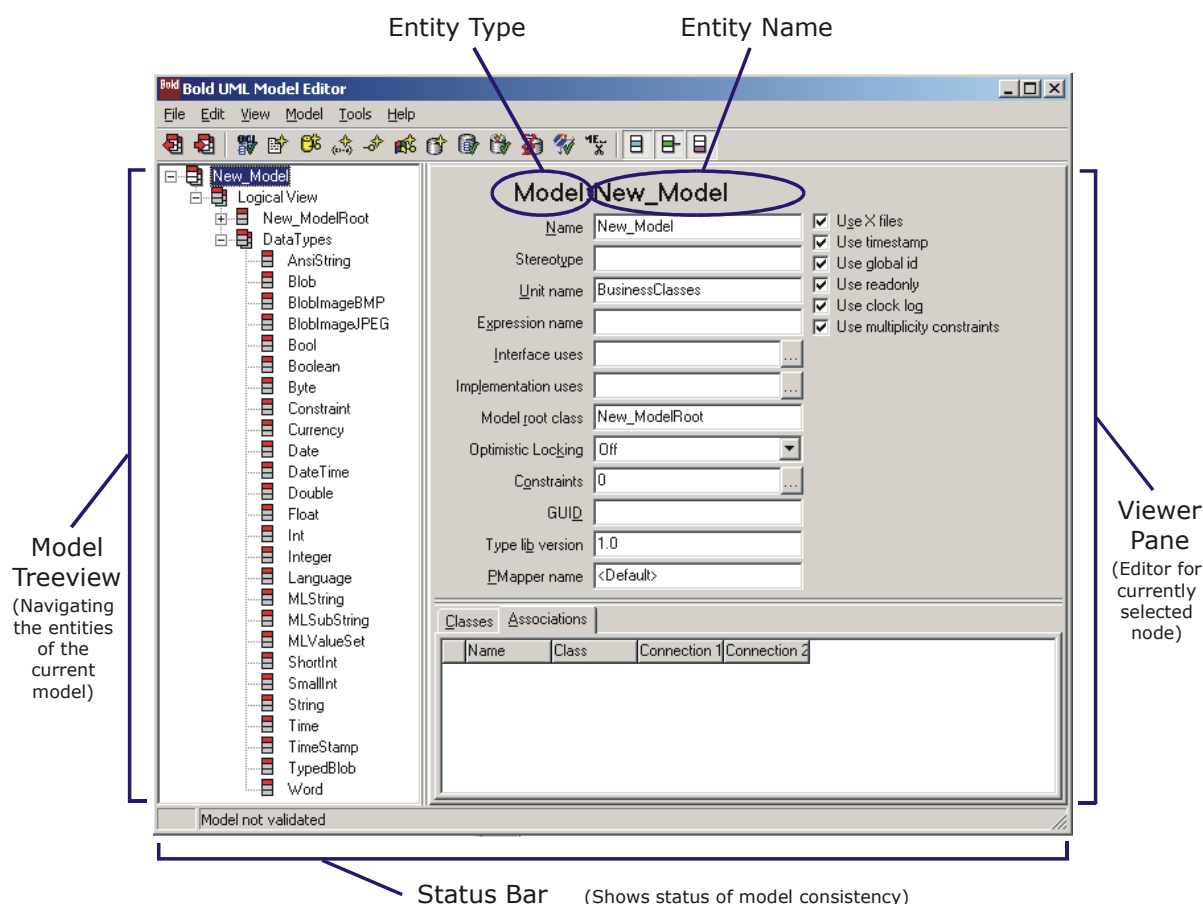

Figure 1: Model Editor

Model Entities

The modeler supports the following entity types:

- **Model:** Collection of all entities representing the problem domain.
- **Package:** A collection of entities representing a subset of the entire model, useful for reducing complexity by breaking a model into a collection of smaller models.
- **Class:** An object, similar in concept to Delphi classes (Delphi classes can be generated for an UML class) but can contain information and relationships not directly available to object pascal classes. Fortunately Bold for Delphi wraps this extra functionality into object pascal classes by using association classes and special list classes that remove this complexity for the pascal programmer.
- **Attribute:** This is the equivalent to Delphi properties, however in Bold for Delphi these can be derived using an OCL expression or have initialization values set directly in the model. In a lot of circumstances this removes the

need to create getter and setter methods like in traditional object pascal
programming.
- **Operation:** This is the equivalent to a Delphi method (Function/Procedure).
- **Association:** Associations represent the relationships between classes.
  Depending on the type of relationship an association maybe represented by a
  class itself, this allows for Operations & Attributes to be part of an association
  between classes. Associations in Bold for Delphi are much more powerful than
  properties of a class type in Delphi objects, again the complexity of mapping
  this extra functionality in Delphi classes is performed by Bold for Delphi.
- **Role:** A Role represents the connection of an Association to a class.
- **Data Type:** Represents the different Data Types supported within the model.
  This can be extended to include custom Data Types.

Figure 2 shows the how different entities are represented in the Model Editor. These
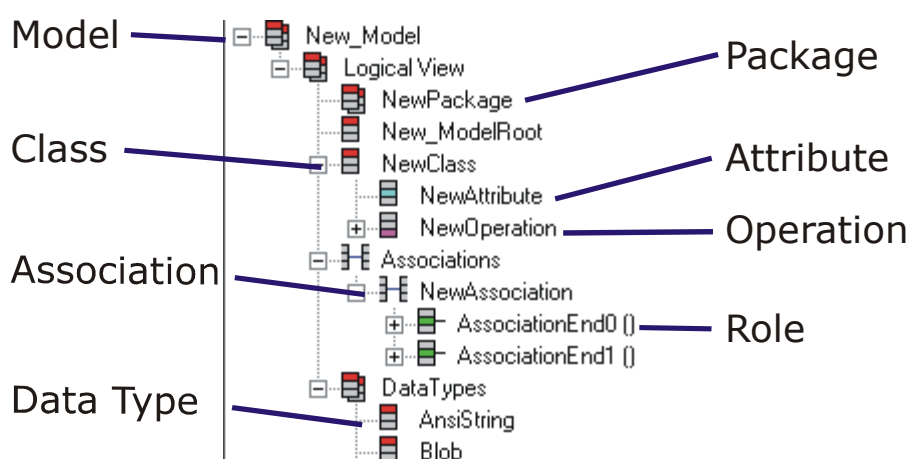will be explained in more detail as we build our model.

Figure 2: Model Entity Types

The *Model* menu in the editor will change depending on the entity selected, these
options are also available from the right-click context menu in the Model Treeview.
For example, to add a Class you need to have a Model entity or Package entity
selected, as these are the only entities that can contain a Class.

Most of the options are fine 'out of the box' and this is how we will start with Bold for
Delphi.

In Figure 1 The root node *New_Model* is selected and the viewer pane on the right
shows the global options for our model, to begin with the most important are:

- **Name:** The name for the model.

- **Unit name:** The pascal unit that will be generated to hold our classes if we
  choose to generate code. Generating code is optional. However if you do
  generate code this is the default unit that the code is placed. This can be
  overridden for each class if desired.

- **Model root class:** This should be set to the name of the class in our model that is the root of all other classes.

Change the **Name** property to *BusinessClasses.* This is clearer than *New_Model*.

The default super class for all business objects is the *New_ModelRoot* class. Change the **Name** of this class to *BusinessClassesRoot*, this is clearer and represents a valid Delphi class name. When you change the root class in a model ensure you update the model's **Model root class** property to match. In this case we need to set it to *BusinessClassesRoot*.

Adding our Business Objects
With the newly named node *BusinessClasses* highlighted, select the *New Class* option from the *Model* menu. This option is only available if a *model* entity is selected in the treeview.

Change the name of the newly created class to *Contact* and then repeat creating a new class for both *Company* and *Person.* Don't forget you will need to select the *BusinesClasses* model each time for the *New Class* menu item to be visible. (Optionally you can use the right-click context menu in the treeview)

Set the following options for the new classes:
- **Contact:** Set the abstract checkbox as checked.
- **Person:** Set the *SuperClass* combo box to *Contact.*
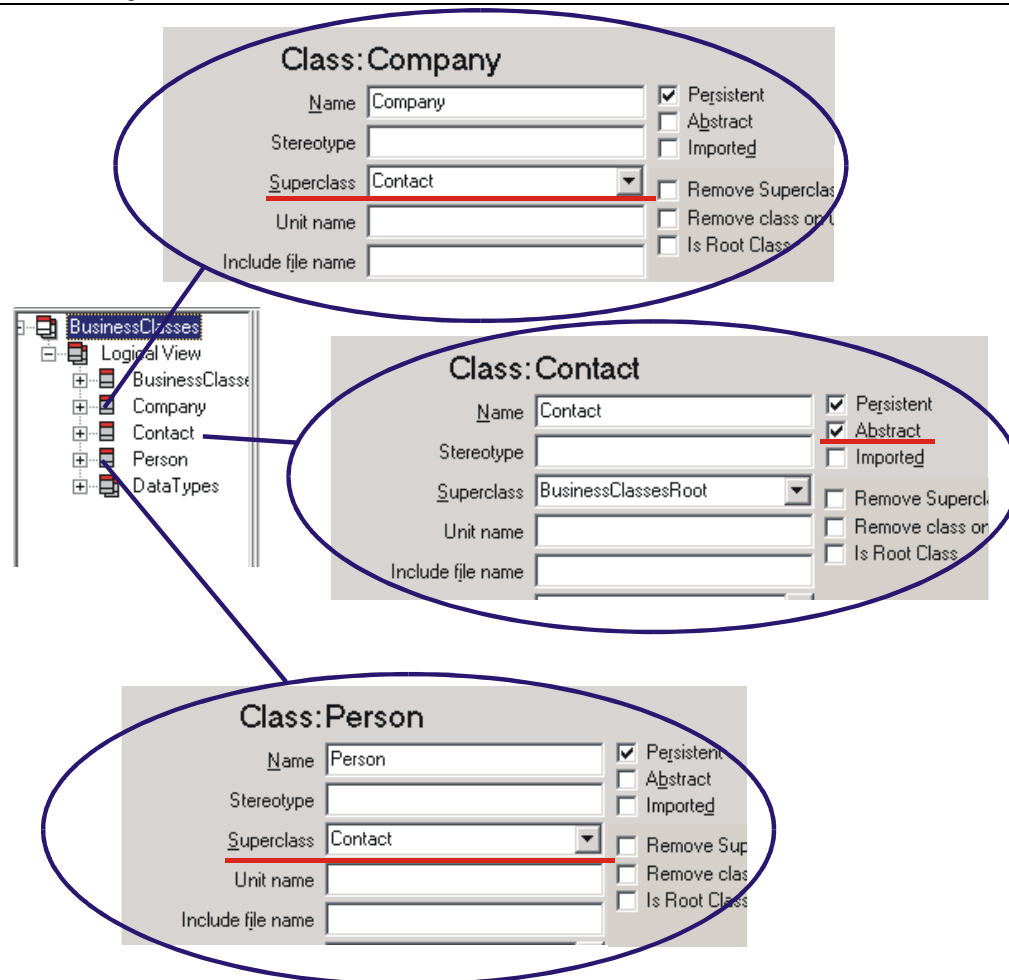- **Company:** Set the *SuperClass* combo box to *Contact.*

Figure 3: Basic Classes

Figure 3 shows the changes applied in the model editor. We have now created our basic classes, but more importantly we have enforced some of the business rules mentioned earlier.

By marking the *Contact* class as abstract, Bold for Delphi will not allow you to create instances of this class. Because both *Company* and *Person* have the superclass of *Contact,* they will inherit any attributes and operations of *Contact.* Importantly any instance of *Company* or *Person* can be treated as if they are a class of *Contact*, making it possible, where required to treat these classes the same.

Adding Attributes
Select the *Contact* class in the model editor treeview. Using the *Model* menu again you'll notice the options have changed to suit a class entity. Select *Add Attribute* three times and name them *name, address* and *phoneNumber* respectively. You will notice the capitalization I have used is a bit different. This is an UML convention, although not enforced in the model, it is a good habit to follow. When writing OCL expressions the UML capitalization style is important and errors will result form poorly formed expressions.

Check each attribute as *Persistent*; this is required for Bold for Delphi to save the values to the database. Leave the default type as *String* and leave the default length of *255*. Strings are the only type that recognize the length parameter.

Because we have added these attributes to the *Contact* class and both *Company* and *Person* descend from *Contact*, they will both inherit the attributes *name, address* and *phoneNumber.*
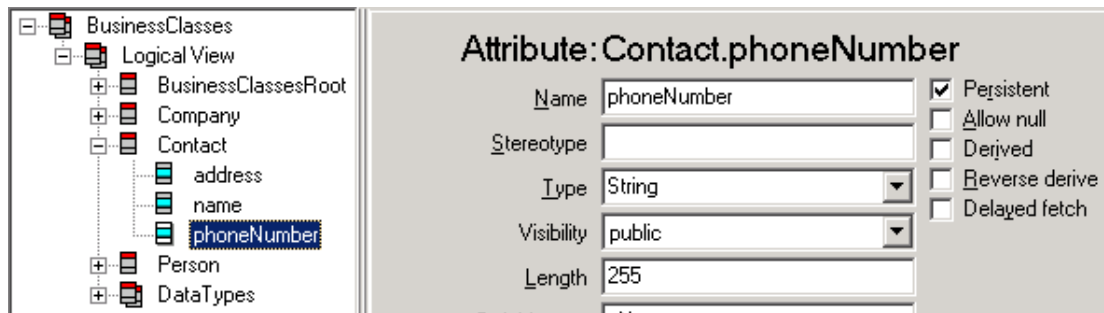

Figure 4: Basic Attributes

Adding the Association
With *BusinessClasses* highlighted, select the *New Association* option from the *Model* menu. This option is only available if a *model* entity is selected in the treeview.

Name the new association *Employment* and name each of the association ends (Roles) *worksFor* and *employs.* For each of the roles set the following parameters:

- **worksFor:** Select *Company* as the class for this role.
- **employs:** Select *Person* as the class for this role. Unselect the *Embed* checkbox and choose `0..*` as the multiplicity for this role. You will see on changing the multiplicity that Bold for Delphi automatically checks the *Multi* checkbox.
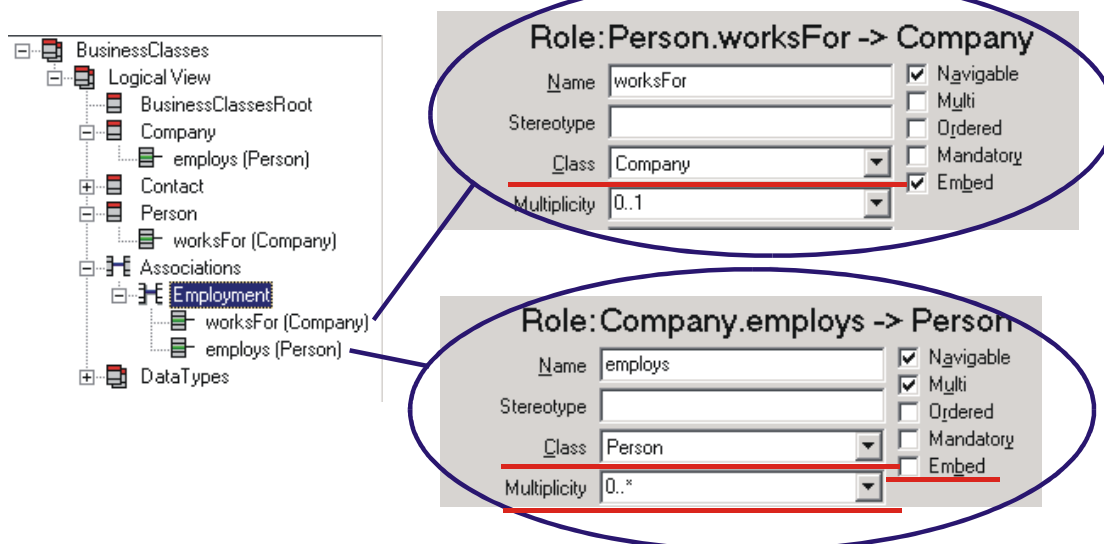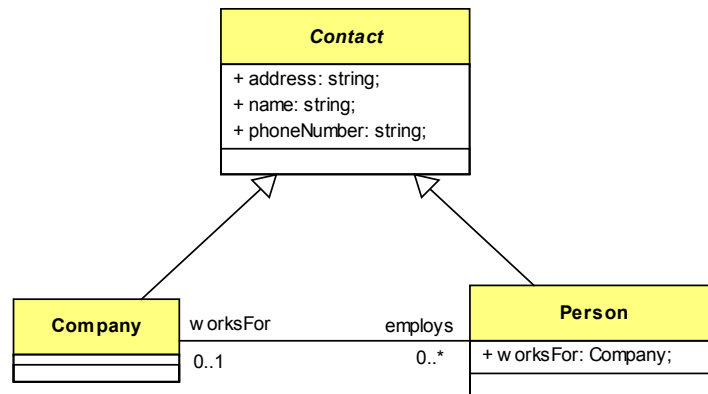
These changes are shown in Figure 5.

Figure 5: Employment Association

This is all that is required to complete the modeling of our original design. Using Bold for Delphi we have accurately represented all shown business rules simply and without complicating the design due to language limitations.

Before we leave the model editor lets have a look at *multiplicity* and *embed* options a little further.

Multiplicity is the measure of the number of 'connections' at the association end point. There are four basic options:

- `0..1:` Zero or one connection.
- `1..1:` Must have one and only one connection.
- `0..*:` Zero or many connections.
- `1..*:` One or many connections.

It is also possible to use n-cardinality relationships like 4..7 or 2..2. Embedding is the process of using a field in association end's class to store the associated objects reference, for now all you need to know is that:

- To be able to embed an association end it must be of single cardinality. E.g. 0..1 or 1..1.
- If both associations can have two or more connections (i.e. a multi-multi association) then a link class must be used to hold the association. This is also true if you decide not to embed both ends of a single cardinality association.
- It is not good practice to embed both ends of an association.

This topic will be revisited in more detail later.

Adding a User Interface

Simple Main Form

To get the application up and running as soon as possible we will build a simple main form allowing us to edit the business objects. This will allow for testing of the design and a demonstration of a few Bold for Delphi features.

To the **MainForm.pas** unit add **BoldAFPDefault** and **BusinessLayer** to the uses clause. The **BoldAFPDefault** unit allows Bold for Delphi to automatically generate editor forms for the business objects that can be launched automatically from Bold-aware controls. The **BusinessLayer** unit allows the Bold for Delphi Controls to see the model on the data module.

From the **Bold Handles** component palette add 3 **BoldListHandle** components to the form. From the **Bold Controls** component palette add 3 **BoldNavigator** components and 3 **BoldGrid** components. Also add 3 standard Delphi **Label** components to help keep everything in order.

Use the following layout to position the controls. The caption of each label has been set to describe the control:
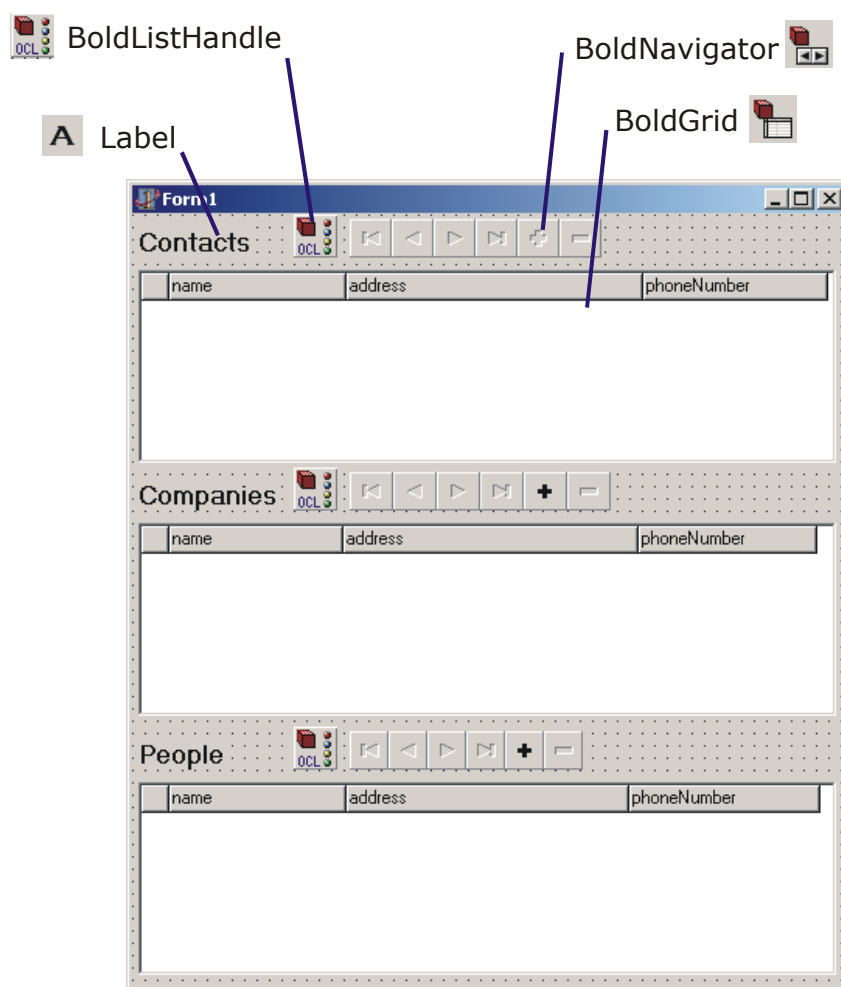
For each group of controls set the following properties:

Contacts
BoldListHandle
Name: `ContactList`
RootHandle: `BusinessModule.BoldSystemHandle1`
Expression: `Contact.allInstances`

BoldNavigator
Name: `ContactNavigator`
BoldHandle: `ContactList`

BoldGrid
Name: `ContactGrid`
BoldHandle: `ContactList`

After setting the **BoldHandle** property, right-click on the grid and select **Create Default Columns** from the context menu.

Companies
BoldListHandle
Name: `CompanyList`
RootHandle: `BusinessModule.BoldSystemHandle1`
Expression: `Company.allInstances`

BoldNavigator
Name: `CompanyNavigator`
BoldHandle: `CompanyList`

BoldGrid
Name: `CompanyGrid`
BoldHandle: `CompanyList`

After setting the **BoldHandle** property, right-click on the grid and select **Create Default Columns** from the context menu.

People
BoldListHandle
Name: `PersonList`
RootHandle: `BusinessModule.BoldSystemHandle1`
Expression: `Person.allInstances`

BoldNavigator
Name: `PersonNavigator`
BoldHandle: `PersonList`

BoldGrid
Name: `PersonGrid`
BoldHandle: `PersonList`

After setting the **BoldHandle** property, right-click on the grid and select **Create Default Columns** from the context menu.

Almost ready to run, add the following event handler for the form's **OnClose** event:

```
procedure TForm1.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
  BusinessModule.BoldSystemHandle1.System.UpdateDatabase;
end;
```

This ensures that any information entered at runtime will be saved to the XML file.

Running The Application

Entering Test Data

Run the application and enter some companies and people. Figure 6 shows the application at runtime with some sample data.

'New Contact' disabled

Figure 6: Basic Application at Runtime

It is important to spend sometime discussing what the features of this basic screen are in comparison to traditional Delphi database applications. The distinctions are important because the Bold for Delphi data-aware solution overcomes many shortcomings of traditional Delphi data-aware controls.

Delphi Data-aware and Bold-aware Comparison

There are some important behavioral differences between Delphi data-aware and Bold-aware components. The key to the difference is meta-data, that is data about data. This may be confusing at first but it's the meta-data in the model driven Bold for Delphi approach that allows for the more sophisticated controls in a Bold-aware application.

A simple example is the **Contacts Navigator**, it recognizes that **Contacts** is an abstract class and cannot be created directly. Therefore it disables the '**+**' append button (As indicated in Figure 6). Although only a simple example, the benefits of controls that understand the context of the data they are presenting is extremely valuable and assists in writing correct code.

Another important distinction between Bold-aware and standard data-aware solutions is the synchronization of data between grids. In a traditional data-aware application you would need to resort to using:

A) Multiple Queries
Three grids each linked to a query containing the SQL statement required to retrieve the desired data.
This solution is relatively easy to implement but suffers the problem of no synchronization between each grid. Updating each grid requires re-running the queries. This can be problematic and requires careful consideration with the status of each query (e.g. Is it currently in edit mode?).

B) Filtered Grids
By using a single query and purchasing a third-party grid with inbuilt filtering it may be possible to replicate the functionality without resorting to desynchronizing queries. However depending on the nature of the SQL statement it may not be updateable, leaving you with a read-only solution. Another, not so obvious, limitation of this method is that a single record only can be the 'active' or 'current' record. Depending on the third-party grid supplied this can cause problems.

"There are some important behavioral differences between Delphi Data-aware and Bold-aware components"

In Figure 6 that each grid has a separate object selected and entering or changing data the changes are automatically propagated to all other controls. Bold for Delphi supports various methods to ensure this behavior is supported between applications running on different computers across the network. A change in an object on one computer can automatically be reflected and updated on every other computer on the network.

More examples of the differences will eventuate as the series progresses.

Default Object Editor Forms

With the application running double click the index marker on the left hand side of a grid. An object editor dialog will be created, customized to the object type. These

editors are enabled by the inclusion of the **BoldAFPDefault** unit anywhere in the application.
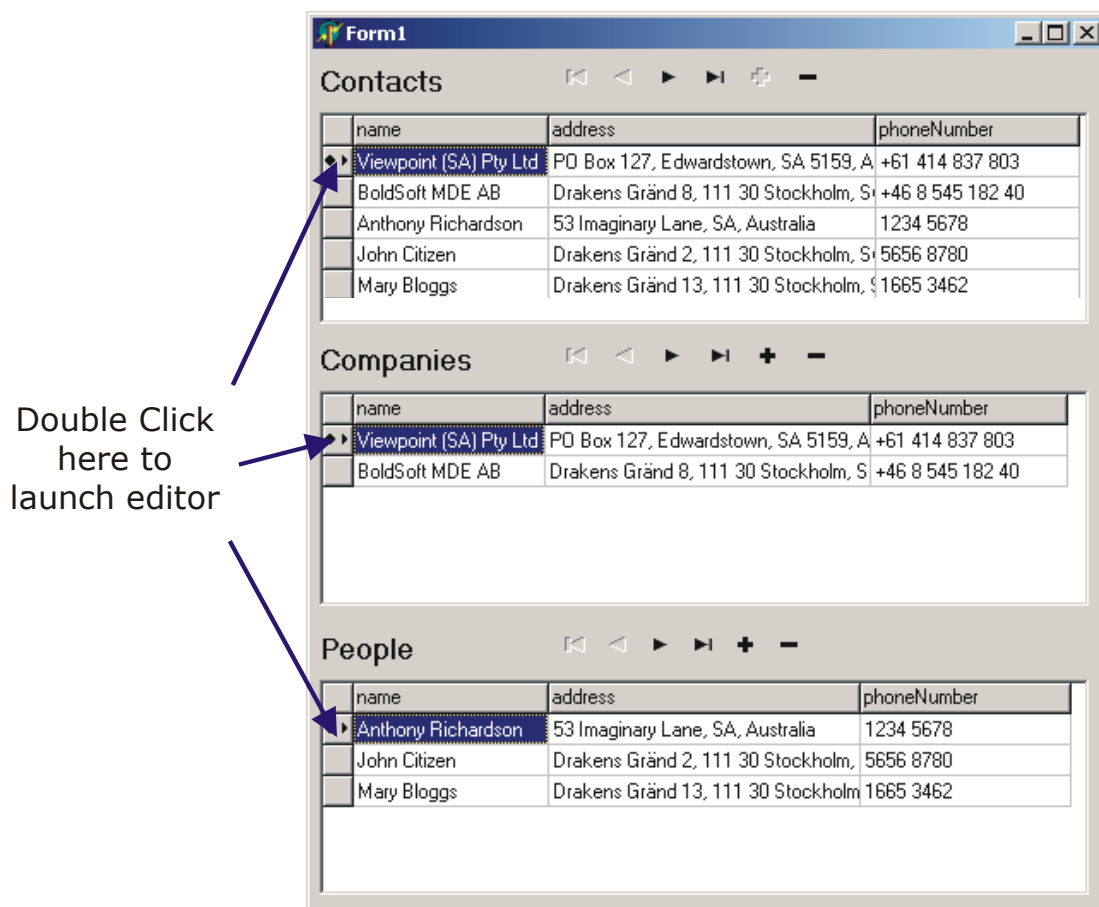
Figure 7: Editor Launch Points

In Figure 7 the selection column of the grid displays a small circle to indicate that the object in that row is selected and a small arrowhead to indicate that the object for that row is the currently active object. It is possible to multiple select objects (rows) in the grid by *ctrl+clicking* each object. It is also possible to select all objects in the grid by clicking the top left cell in the column header.

The default object editors aren't much to look at but they are fully functional and support all relationships. You may have noticed that our application was incapable of displaying and editing the relationships between companies and people. The built in editors support these relationships.

Using the Drag Points on the editors (refer Figure 8) you can establish the links by dragging a **Person** into the employs grid of the **Company** object.
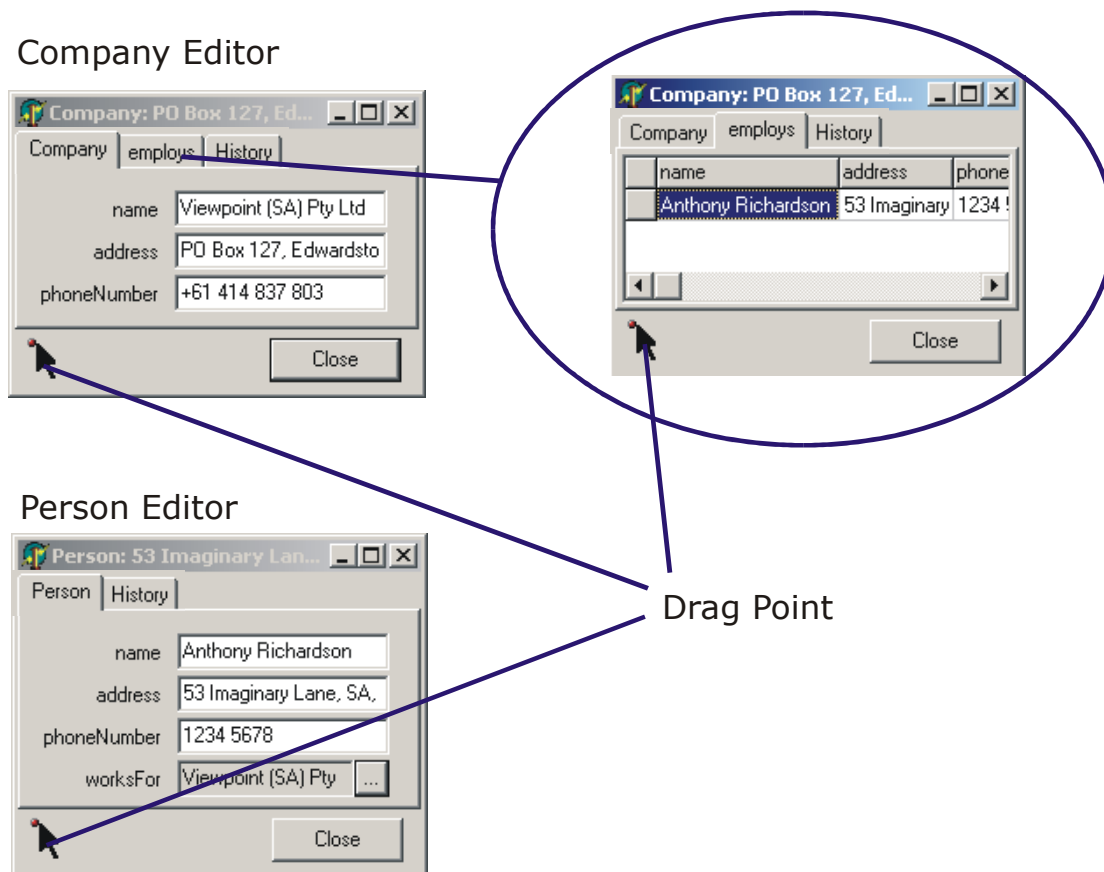
Figure 8: Object Runtime Editors

It is also important to note the other differences with these editors. They are not linked to the grid so you can have multiple editors open and navigate the grid without problem. It is problematic in a traditional data-aware application to do this as when the cursor changes the edit controls change.

The 'History' tab is used if you have purchased the optional Object Versioning Extension. With this powerful add-on Bold for Delphi will record the changes in your objects over time. By marking a class as versioned in the Bold for Delphi UML editor this recording is enabled. By using OCL or direct method calls of your domain objects, you can request an instance of your objects as they were at any point in time. Bold for Delphi even versions the relationships so you can navigate your model as it was at that time.

Summary

This article has demonstrated the basics a building a Bold for Delphi Model driven application. It has shown the key concepts of a simple Bold for Delphi Application and discussed briefly the usage of these concepts. A basic comparison to traditional database programming was supplied where possible.

As much as this article has introduced the basics, many questions remain and only a small part of the Bold for Delphi product has been shown.

Future

Future articles will explore some of the existing concepts in more depth and venture into the specifics of using various Bold for Delphi constructs to solve the common challenges faced in application development.

Appendix A: Source Code

Instructions

In this section you will find the source code for the project used in this article. The article clear shows how to build this project step by step, however you may choose to use this source code. Maybe you prefer to run the program before reading the article or you are having trouble understanding the article and wish to look at the source.

Whatever you reason to create the source files is simple. All files can be created using an standard ASCII text editor like notepad.exe. Cut and paste the contents of each file and save the file using the correct name and extension.

The format of the listing is:

Start listing Filename

```
Contents of file
In between the lines.
```

End listing Filename

All files need to placed in the same directory, then simply load the DPR file into Delphi and compile.

ContactManager.dpr

Start listing ContactManager.dpr

```
program ContactManager;

uses
  Forms,
  MainForm in 'MainForm.pas' {Form1},
  BusinessLayer in 'BusinessLayer.pas' {BusinessModule: TDataModule};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TBusinessModule, BusinessModule);
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

End listing ContactManager.dpr

MainForm.pas

Start listing MainForm.pas

```pascal
unit MainForm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  BusinessLayer, ExtCtrls, BoldNavigator, Grids, BoldGrid,
  BoldSubscription, BoldHandles, BoldRootedHandles, BoldAbstractListHandle,
  BoldCursorHandle, BoldListHandle, BoldAFPDefault, StdCtrls,
  BoldNavigatorDefs;

type
  TForm1 = class(TForm)
    ContactGrid: TBoldGrid;
    ContactNavigator: TBoldNavigator;
    CompanyGrid: TBoldGrid;
    PersonGrid: TBoldGrid;
    ContactList: TBoldListHandle;
    CompanyList: TBoldListHandle;
    PersonList: TBoldListHandle;
    CompanyNavigator: TBoldNavigator;
    PersonNavigator: TBoldNavigator;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  BusinessModule.BoldSystemHandle1.System.UpdateDatabase;
end;

end.
```

End listing of MainForm.pas

MainForm.dfm

Start listing MainForm.dfm

```
object Form1: TForm1
  Left = 368
  Top = 199
  Width = 467
  Height = 528
  Caption = 'Form1'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  OnClose = FormClose
  PixelsPerInch = 96
  TextHeight = 13
  object Label1: TLabel
    Left = 4
    Top = 8
    Width = 73
    Height = 20
    Caption = 'Contacts'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -16
    Font.Name = 'MS Sans Serif'
    Font.Style = [fsBold]
    ParentFont = False
  end
  object Label2: TLabel
    Left = 4
    Top = 172
    Width = 90
    Height = 20
    Caption = 'Companies'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -16
    Font.Name = 'MS Sans Serif'
    Font.Style = [fsBold]
    ParentFont = False
  end
  object Label3: TLabel
    Left = 4
    Top = 340
    Width = 56
    Height = 20
    Caption = 'People'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -16
    Font.Name = 'MS Sans Serif'
    Font.Style = [fsBold]
    ParentFont = False
  end
  object ContactGrid: TBoldGrid
    Left = 4
    Top = 36
    Width = 449
    Height = 125
    AddNewAtEnd = False
    BoldAutoColumns = False
    BoldShowConstraints = False
    BoldHandle = ContactList
    BoldProperties.NilElementMode = neNone
    Columns = <
      item
        Color = clBtnFace
        Font.Charset = DEFAULT_CHARSET
        Font.Color = clWindowText
        Font.Height = -11
        Font.Name = 'MS Sans Serif'
        Font.Style = []
      end
      item
```

```
      BoldProperties.Expression = 'name'
      Font.Charset = DEFAULT_CHARSET
      Font.Color = clWindowText
      Font.Height = -11
      Font.Name = 'MS Sans Serif'
      Font.Style = []
    end
    item
      BoldProperties.Expression = 'address'
      Font.Charset = DEFAULT_CHARSET
      Font.Color = clWindowText
      Font.Height = -11
      Font.Name = 'MS Sans Serif'
      Font.Style = []
    end
    item
      BoldProperties.Expression = 'phoneNumber'
      Font.Charset = DEFAULT_CHARSET
      Font.Color = clWindowText
      Font.Height = -11
      Font.Name = 'MS Sans Serif'
      Font.Style = []
    end>
  DefaultRowHeight = 17
  EnableColAdjust = False
  TabOrder = 0
  TitleFont.Charset = DEFAULT_CHARSET
  TitleFont.Color = clWindowText
  TitleFont.Height = -11
  TitleFont.Name = 'MS Sans Serif'
  TitleFont.Style = []
  ColWidths = (
    17
    113
    190
    118)
end
object ContactNavigator: TBoldNavigator
  Left = 140
  Top = 3
  Width = 162
  Height = 25
  BoldHandle = ContactList
  Flat = True
  TabOrder = 1
  ImageIndices.nbFirst = -1
  ImageIndices.nbPrior = -1
  ImageIndices.nbNext = -1
  ImageIndices.nbLast = -1
  ImageIndices.nbInsert = -1
  ImageIndices.nbDelete = -1
  ImageIndices.nbMoveUp = -1
  ImageIndices.nbMoveDown = -1
  DeleteQuestion = 'Delete %0:s?'
  UnlinkQuestion = 'Unlink "%1:s" from "%2:s"?'
  RemoveQuestion = 'Remove "%1:s" from the list?'
end
object CompanyGrid: TBoldGrid
  Left = 4
  Top = 200
  Width = 449
  Height = 125
  AddNewAtEnd = False
  BoldAutoColumns = False
  BoldShowConstraints = False
  BoldHandle = CompanyList
  BoldProperties.NilElementMode = neNone
  Columns = <
    item
      Color = clBtnFace
      Font.Charset = DEFAULT_CHARSET
      Font.Color = clWindowText
      Font.Height = -11
      Font.Name = 'MS Sans Serif'
      Font.Style = []
    end
    item
      BoldProperties.Expression = 'name'
      Font.Charset = DEFAULT_CHARSET
      Font.Color = clWindowText
```

```
        Font.Height = -11
        Font.Name = 'MS Sans Serif'
        Font.Style = []
      end
      item
        BoldProperties.Expression = 'address'
        Font.Charset = DEFAULT_CHARSET
        Font.Color = clWindowText
        Font.Height = -11
        Font.Name = 'MS Sans Serif'
        Font.Style = []
      end
      item
        BoldProperties.Expression = 'phoneNumber'
        Font.Charset = DEFAULT_CHARSET
        Font.Color = clWindowText
        Font.Height = -11
        Font.Name = 'MS Sans Serif'
        Font.Style = []
      end>
    DefaultRowHeight = 17
    EnableColAdjust = False
    TabOrder = 2
    TitleFont.Charset = DEFAULT_CHARSET
    TitleFont.Color = clWindowText
    TitleFont.Height = -11
    TitleFont.Name = 'MS Sans Serif'
    TitleFont.Style = []
    ColWidths = (
      17
      111
      189
      114)
  end
  object PersonGrid: TBoldGrid
    Left = 4
    Top = 368
    Width = 449
    Height = 125
    AddNewAtEnd = False
    BoldAutoColumns = False
    BoldShowConstraints = False
    BoldHandle = PersonList
    BoldProperties.NilElementMode = neNone
    Columns = <
      item
        Color = clBtnFace
        Font.Charset = DEFAULT_CHARSET
        Font.Color = clWindowText
        Font.Height = -11
        Font.Name = 'MS Sans Serif'
        Font.Style = []
      end
      item
        BoldProperties.Expression = 'name'
        Font.Charset = DEFAULT_CHARSET
        Font.Color = clWindowText
        Font.Height = -11
        Font.Name = 'MS Sans Serif'
        Font.Style = []
      end
      item
        BoldProperties.Expression = 'address'
        Font.Charset = DEFAULT_CHARSET
        Font.Color = clWindowText
        Font.Height = -11
        Font.Name = 'MS Sans Serif'
        Font.Style = []
      end
      item
        BoldProperties.Expression = 'phoneNumber'
        Font.Charset = DEFAULT_CHARSET
        Font.Color = clWindowText
        Font.Height = -11
        Font.Name = 'MS Sans Serif'
        Font.Style = []
      end>
    DefaultRowHeight = 17
    EnableColAdjust = False
    TabOrder = 3
```

```
      TitleFont.Charset = DEFAULT_CHARSET
      TitleFont.Color = clWindowText
      TitleFont.Height = -11
      TitleFont.Name = 'MS Sans Serif'
      TitleFont.Style = []
      ColWidths = (
        17
        113
        181
        122)
    end
    object CompanyNavigator: TBoldNavigator
      Left = 140
      Top = 167
      Width = 168
      Height = 25
      BoldHandle = CompanyList
      Flat = True
      TabOrder = 4
      ImageIndices.nbFirst = -1
      ImageIndices.nbPrior = -1
      ImageIndices.nbNext = -1
      ImageIndices.nbLast = -1
      ImageIndices.nbInsert = -1
      ImageIndices.nbDelete = -1
      ImageIndices.nbMoveUp = -1
      ImageIndices.nbMoveDown = -1
      DeleteQuestion = 'Delete %0:s?'
      UnlinkQuestion = 'Unlink "%1:s" from "%2:s"?'
      RemoveQuestion = 'Remove "%1:s" from the list?'
    end
    object PersonNavigator: TBoldNavigator
      Left = 140
      Top = 335
      Width = 150
      Height = 25
      BoldHandle = PersonList
      Flat = True
      TabOrder = 5
      ImageIndices.nbFirst = -1
      ImageIndices.nbPrior = -1
      ImageIndices.nbNext = -1
      ImageIndices.nbLast = -1
      ImageIndices.nbInsert = -1
      ImageIndices.nbDelete = -1
      ImageIndices.nbMoveUp = -1
      ImageIndices.nbMoveDown = -1
      DeleteQuestion = 'Delete %0:s?'
      UnlinkQuestion = 'Unlink "%1:s" from "%2:s"?'
      RemoveQuestion = 'Remove "%1:s" from the list?'
    end
    object ContactList: TBoldListHandle
      RootHandle = BusinessModule.BoldSystemHandle1
      Expression = 'Contact.allInstances'
      Left = 104
    end
    object CompanyList: TBoldListHandle
      RootHandle = BusinessModule.BoldSystemHandle1
      Expression = 'Company.allInstances'
      Left = 104
      Top = 164
    end
    object PersonList: TBoldListHandle
      RootHandle = BusinessModule.BoldSystemHandle1
      Expression = 'Person.allInstances'
      Left = 104
      Top = 332
    end
end
```

End listing MainForm.dfm

BusinessLayer.pas

Start listing BusinessLayer.pas

```
unit BusinessLayer;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  BoldAbstractModel, BoldModel, BoldHandles, BoldSubscription,
  BoldSystemHandle, BoldHandle, BoldPersistenceHandle,
  BoldPersistenceHandleFile, BoldPersistenceHandleFileXML,
  BoldRootedHandles, BoldAbstractListHandle, BoldCursorHandle,
  BoldListHandle;

type
  TBusinessModule = class(TDataModule)
    BoldSystemHandle1: TBoldSystemHandle;
    BoldSystemTypeInfoHandle1: TBoldSystemTypeInfoHandle;
    BoldModel1: TBoldModel;
    BoldPersistenceHandleFileXML1: TBoldPersistenceHandleFileXML;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  BusinessModule: TBusinessModule;

implementation

{$R *.DFM}

end.
```

End Listing BusinessLayer.pas

BusinessLayer.dfm

Start listing BusinessLayer.dfm

```
object BusinessModule: TBusinessModule
  OldCreateOrder = False
  Left = 365
  Top = 223
  Height = 438
  Width = 686
  object BoldSystemHandle1: TBoldSystemHandle
    IsDefault = True
    SystemTypeInfoHandle = BoldSystemTypeInfoHandle1
    Active = True
    PersistenceHandle = BoldPersistenceHandleFileXML1
    Left = 68
    Top = 128
  end
  object BoldSystemTypeInfoHandle1: TBoldSystemTypeInfoHandle
    BoldModel = BoldModel1
    UseGeneratedCode = False
    Left = 68
    Top = 68
  end
  object BoldModel1: TBoldModel
    UMLModelMode = ummNone
    Boldify.EnforceDefaultUMLCase = False
    Boldify.DefaultNavigableMultiplicity = '0..1'
    Boldify.DefaultNonNavigableMultiplicity = '0..*'
    Left = 68
    Top = 12
    Model = (
      'VERSION 19'
      '(Model'
      #9'"BusinessClasses"'
      #9'"BusinessClassesRoot"'
      #9'""'
      #9'""'

        #9'"_Boldify.boldified=True,_BoldInternal.flattened=True,_BoldInte' +
        'rnal.ModelErrors=,Bold.DelphiName=<Name>,Bold.RootClass=Business' +
        'ClassesRoot"'
      #9'(Classes'
      #9#9'(Class'
      #9#9#9'"BusinessClassesRoot"'
      #9#9#9'"<NONE>"'
      #9#9#9'TRUE'
      #9#9#9'FALSE'
      #9#9#9'""'
      #9#9#9'""'
      #9#9#9'"persistence=persistent"'
      #9#9#9'(Attributes'
      #9#9#9')'
      #9#9#9'(Methods'
      #9#9#9')'
      #9#9')'
      #9#9'(Class'
      #9#9#9'"Contact"'
      #9#9#9'"BusinessClassesRoot"'
      #9#9#9'TRUE'
      #9#9#9'TRUE'
      #9#9#9'""'
      #9#9#9'""'
      #9#9#9'"persistence=persistent"'
      #9#9#9'(Attributes'
      #9#9#9#9'(Attribute'
      #9#9#9#9#9'"name"'
      #9#9#9#9#9'"String"'
      #9#9#9#9#9'FALSE'
      #9#9#9#9#9'""'
      #9#9#9#9#9'""'
      #9#9#9#9#9'2'
      #9#9#9#9#9'""'
      #9#9#9#9#9'"persistence=persistent"'
      #9#9#9#9')'
      #9#9#9#9'(Attribute'
      #9#9#9#9#9'"address"'
      #9#9#9#9#9'"String"'
      #9#9#9#9#9'FALSE'
```

```
#9#9#9#9#9'""'
#9#9#9#9#9'""'
#9#9#9#9#9'2'
#9#9#9#9#9'""'
#9#9#9#9#9'"persistence=persistent"'
#9#9#9#9')'
#9#9#9#9'(Attribute'
#9#9#9#9#9'"phoneNumber"'
#9#9#9#9#9'"String"'
#9#9#9#9#9'FALSE'
#9#9#9#9#9'""'
#9#9#9#9#9'""'
#9#9#9#9#9'2'
#9#9#9#9#9'""'
#9#9#9#9#9'"persistence=persistent"'
#9#9#9#9')'
#9#9#9')'
#9#9#9'(Methods'
#9#9#9')'
#9#9')'
#9#9'(Class'
#9#9#9'"Person"'
#9#9#9'"Contact"'
#9#9#9'TRUE'
#9#9#9'FALSE'
#9#9#9'""'
#9#9#9'""'
#9#9#9'"persistence=persistent"'
#9#9#9'(Attributes'
#9#9#9')'
#9#9#9'(Methods'
#9#9#9')'
#9#9')'
#9#9'(Class'
#9#9#9'"Company"'
#9#9#9'"Contact"'
#9#9#9'TRUE'
#9#9#9'FALSE'
#9#9#9'""'
#9#9#9'""'
#9#9#9'"persistence=persistent"'
#9#9#9'(Attributes'
#9#9#9')'
#9#9#9'(Methods'
#9#9#9')'
#9#9')'
#9')'
#9'(Associations'
#9#9'(Association'
#9#9#9'"Employment"'
#9#9#9'"<NONE>"'
#9#9#9'""'
#9#9#9'""'
#9#9#9'"persistence=persistent,Bold.DelphiName=<Name>"'
#9#9#9'FALSE'
#9#9#9'(Roles'
#9#9#9#9'(Role'
#9#9#9#9#9'"worksFor"'
#9#9#9#9#9'TRUE'
#9#9#9#9#9'FALSE'
#9#9#9#9#9'"Person"'
#9#9#9#9#9'""'
#9#9#9#9#9'"0..1"'
#9#9#9#9#9'""'
#9#9#9#9#9'0'
#9#9#9#9#9'2'
#9#9#9#9#9'0'
#9#9#9#9#9'""'
#9#9#9#9#9'(Qualifiers'
#9#9#9#9#9')'
#9#9#9#9')'
#9#9#9#9'(Role'
#9#9#9#9#9'"employs"'
#9#9#9#9#9'TRUE'
#9#9#9#9#9'FALSE'
#9#9#9#9#9'"Company"'
#9#9#9#9#9'""'
#9#9#9#9#9'"0..*"'
#9#9#9#9#9'""'
#9#9#9#9#9'0'
```

```
        #9#9#9#9#9'2'
        #9#9#9#9#9'0'
        #9#9#9#9#9'"Bold.Embed=False"'
        #9#9#9#9#9'(Qualifiers'
        #9#9#9#9#9')'
        #9#9#9#9')'
        #9#9#9')'
        #9#9')'
        #9')'
        ')')
  end
  object BoldPersistenceHandleFileXML1: TBoldPersistenceHandleFileXML
    FileName = 'data.xml'
    CacheData = False
    BoldModel = BoldModel1
    Left = 68
    Top = 184
  end
end
```

End Listing BusinessLayer.dfm