

Starting with Bold for Delphi/Bold for C++

Part 2: Extending Models

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

All trademarks are properties of their respective holders. All intellectual property claims are respected.
The publication is copyright 2002 by Anthony Richardson. Anthony Richardson grants BoldSoft MDE
AB a royalty-free non-exclusive license to distribute this publication worldwide.

I would like to acknowledge the assistance of BoldSoft MDE AB in the creation of these articles.
Especially the assistance of Jesper Hogstrom, Jonas Hogstrom and Dan Nygren, without their support
this project would not have been possible.

Contents

STARTING WITH BOLD FOR DELPHI/BOLD FOR C++	1
GETTING STARTED	3
Introduction	3
Contacting Anthony Richardson	3
Contacting BoldSoft.....	3
EXTENDING MODELS.....	4
Example Application.....	4
Build Basic Project	5
Build the Model.....	6
Generating Pascal Code	10
Adding a User Interface.....	11
Running The Application	15
USING CODE IN BOLD APPLICATIONS.....	18
Enhancing the application with code	18
INTRODUCTION TO OBJECT LIFECYCLES.....	20
Using Delphi Code	20
Model driven object life management.....	21
DERIVED & REVERSE DERIVED ATTRIBUTES.....	23
Adding the new attributes	23
Reversed Derived.....	25
Derived Attributes with OCL	27
OPERATIONS	29
Adding an Operation	29
DERIVED RELATIONSHIPS.....	33
Adding the Managed By relationship.....	33
Adding the Workforce relationship.....	34
ONE-WAY RELATIONSHIPS	36
CONSTRAINTS: A PARTING NOTE	41
SUMMARY	41
APPENDIX A: SOURCE CODE	42
Instructions.....	42
HRManager.dpr.....	43
MainForm.pas	44
MainForm.dfm	47
BusinessLayer.pas	55
BusinessLayer.dfm.....	56
HRClasses.pas.....	62
HRClasses.inc.....	69
HRClasses_Interface.inc.....	70

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>

Extending Models

Example Application

To begin the journey into more advanced modeling techniques we will start with a model for a basic Human Resource Management Application.

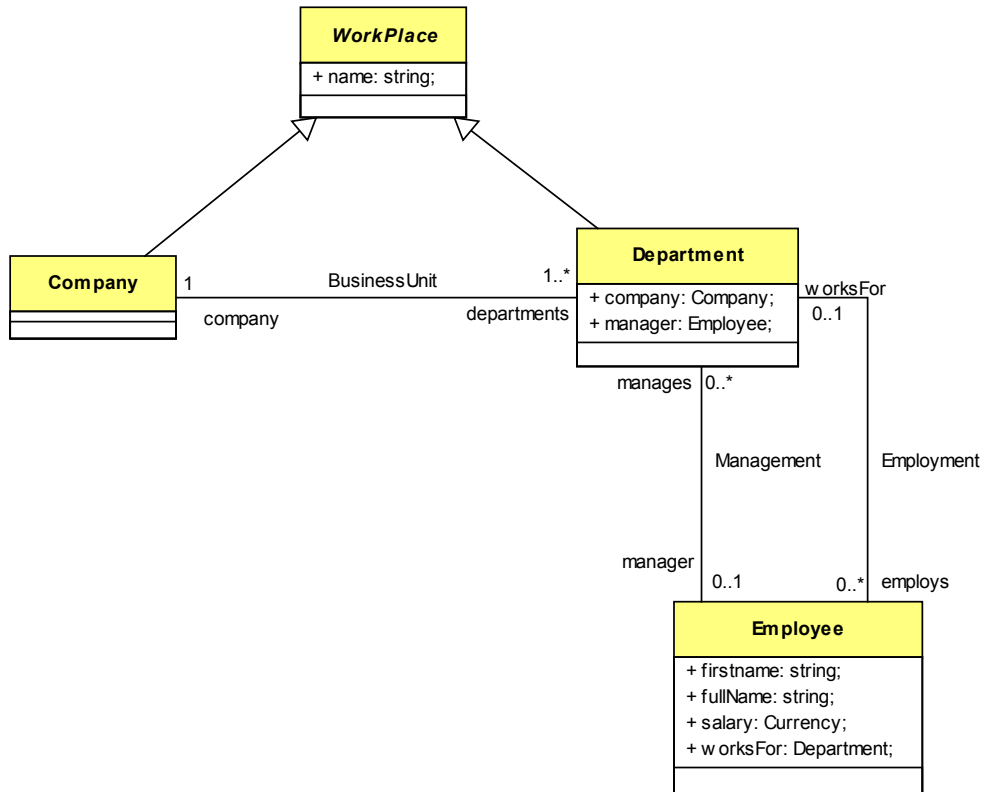


Figure 1: Basic Class Model

The first article in this series covered the construction of a basic model. The concept of classes, attributes and relationships was discussed. In Figure 1 no new concepts are introduced and the following basic business rules are enforced:

- We have Company and Departments, which are types of Workplace.
- We have Employees, which can work for a department
- Companies can contain Departments.
- A Department can have one manager, which is an Employee.
- An Employee can manage multiple departments

We will step through the creation of this with a basic GUI quickly so we can get to the real interesting modeling concepts sooner.

Build Basic Project

Groundwork

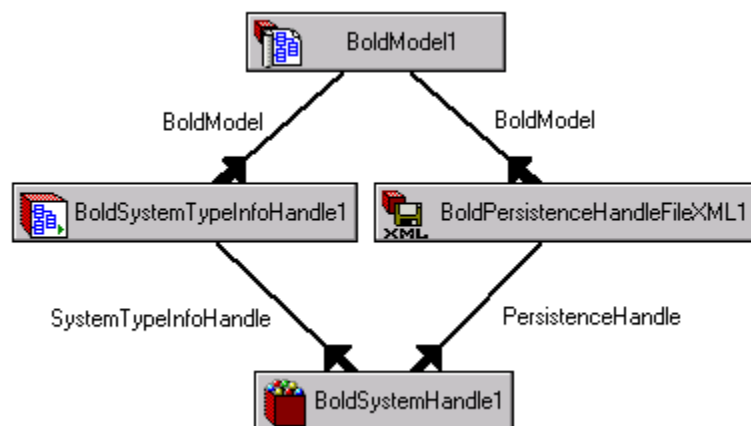
To get started we need to setup a basic Delphi project with a data module.

1. Select *File|New Application* from the Delphi menu.
2. Save the Form unit as *MainForm.pas* and the project as *HRManager.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

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*.
4. From the *Bold Persistence* component tab add a *BoldPersistenceHandleFileXML* component to the data module.
5. Link the *BoldModel* property to the *BoldModel* component.
6. Link the *PersistenceHandle* property of *BoldSystemHandle* to the *BoldPersistenceHandleFileXML* component.



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: `True`

This allows us to work directly with the model using Object Pascal.

Component: BoldPersistenceHandleFileXML1
Property: FileName
Value: HRData.XML

Component: BoldSystemHandle1
Property: AutoActive
Value: True

This property results in the BoldSystemHandle1 opening the XML file and being ready for immediate use on application startup.

Add the following event handler for the data module's **OnDestroy** event:

```
procedure TBusinessModule.DataModuleDestroy(Sender: TObject);  
begin  
    BoldSystemHandle1.System.UpdateDatabase;  
end;
```

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

Build the Model

Double click the BoldModel component to open the Bold UML Model Editor. Set the model *name* to HRClasses as per Figure 2. Change models *Unit name* to HRClasses and *Model root class* value to HRClassesRoot. Change the BusinessClassesRoot classes *name* to HRClassesRoot.

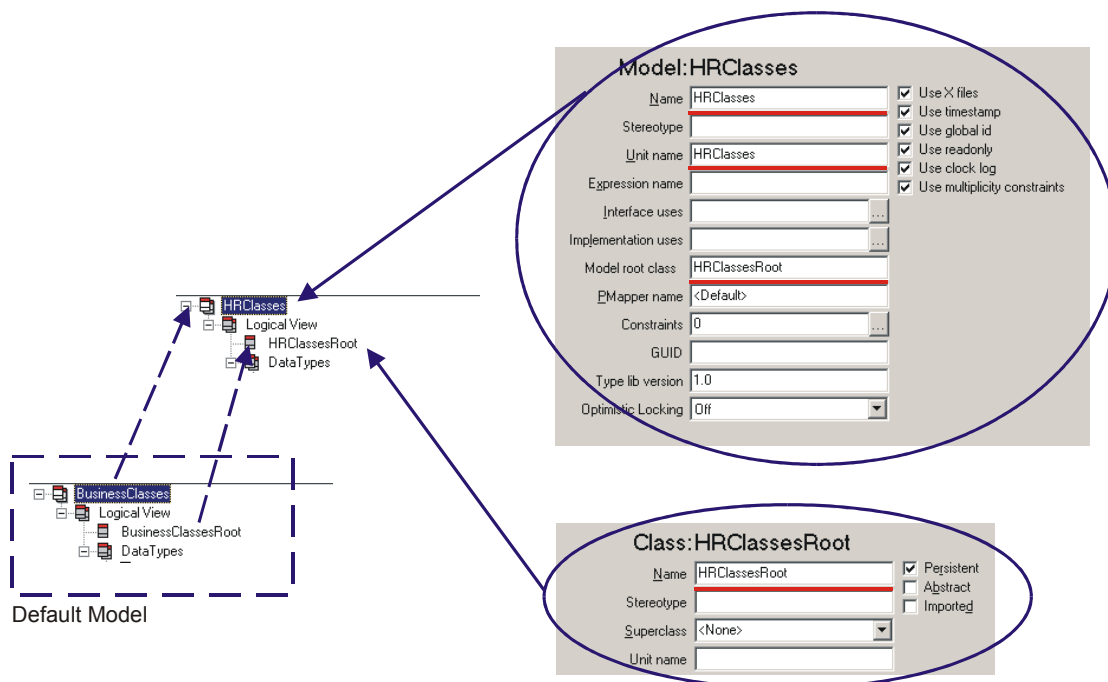


Figure 2: Model Properties

Changing the model's default values above should become a habit. Leaving the default names may cause problems in future projects. When generating source code Bold for Delphi will search the current project directory, directories of other files in the project and the Delphi search path for files matching *Unit name*. If the file is found it

will be overwritten. This is the desired behavior if the file is for the current project but can cause problems if working with multiple models in related projects.

For the reason given above it is also a good idea to ensure the project is saved to a directory before generating code from your model.

Construct the classes and set the properties as indicated:

The image shows a screenshot of a modeling tool interface. On the left is a tree view of a project named 'HRClasses'. The tree view includes a 'Logical View' folder containing 'Company', 'Department', and 'Employee' classes, and an 'Associations' folder containing 'BusinessUnit', 'Employment', and 'Management' associations. On the right are four property panels for the classes: 'Class: Company', 'Class: Department', 'Class: Employee', and 'Class: Workplace'. Each panel has a 'Name' field, a 'Stereotype' field, and a 'Superclass' dropdown menu. The 'Company' panel has 'Name' set to 'Company' and 'Superclass' set to 'Workplace'. The 'Department' panel has 'Name' set to 'Department' and 'Superclass' set to 'Workplace'. The 'Employee' panel has 'Name' set to 'Employee' and 'Superclass' set to 'HRClassesRoot'. The 'Workplace' panel has 'Name' set to 'Workplace' and 'Superclass' set to 'HRClassesRoot'. The 'Abstract' checkbox is checked and underlined in the 'Workplace' panel. The 'Persistent' checkbox is checked in all panels. The 'Imported' checkbox is unchecked in all panels.

Class Name	Superclass	Persistent	Abstract	Imported
Company	Workplace	Checked	Unchecked	Unchecked
Department	Workplace	Checked	Unchecked	Unchecked
Employee	HRClassesRoot	Checked	Unchecked	Unchecked
Workplace	HRClassesRoot	Checked	Checked (Underlined)	Unchecked

Figure 3: Basic Classes

And the attributes:

The image shows a logical view of the HRClasses model on the left and four attribute configuration panels on the right. The logical view includes classes like Company, Department, Employee, and Workplace, with their respective attributes and associations. The attribute panels are:

- Attribute: Employee.firstName**: Name: firstName, Type: String, Persistent: checked, Allow null: unchecked, Derived: unchecked, Reverse derive: unchecked, Delayed fetch: unchecked, Visibility: public.
- Attribute: Employee.lastName**: Name: lastName, Type: String, Persistent: checked, Allow null: unchecked, Derived: unchecked, Reverse derive: unchecked, Delayed fetch: unchecked, Visibility: public.
- Attribute: Employee.monthlySalary**: Name: monthlySalary, Type: Currency, Persistent: checked, Allow null: unchecked, Derived: unchecked, Reverse derive: unchecked, Delayed fetch: unchecked, Visibility: public.
- Attribute: Workplace.name**: Name: name, Type: String, Persistent: checked, Allow null: unchecked, Derived: unchecked, Reverse derive: unchecked, Delayed fetch: unchecked, Visibility: public.

Figure 4: Basic Attributes

And the associations:

The screenshot displays the configuration for several associations in a modeling tool. On the left, a tree view shows the project structure under 'HRClasses', with 'Associations' expanded to show 'BusinessUnit', 'Employment', and 'Management'. On the right, configuration panels for each association and its roles are shown:

- Association: BusinessUnit**
 - Name: BusinessUnit
 - Class: <none>
 - Constraints: 0
 - Properties: Persistent, Derived
- Role: Company.departments -> Department**
 - Name: departments
 - Class: Department
 - Multiplicity: 0..*
 - Properties: Navigable, Multi, Ordered, Mandatory, Embed
- Role: Department.company -> Company**
 - Name: company
 - Class: Company
 - Multiplicity: 1
 - Properties: Navigable, Multi, Ordered, Mandatory, Embed
- Association: Employment**
 - Name: Employment
 - Class: <none>
 - Constraints: 0
 - Properties: Persistent, Derived
- Role: Department.employs -> Employee**
 - Name: employs
 - Class: Employee
 - Multiplicity: 0..*
 - Properties: Navigable, Multi, Ordered, Mandatory, Embed
- Role: Employee.worksFor -> Department**
 - Name: worksFor
 - Class: Department
 - Multiplicity: 0..1
 - Properties: Navigable, Multi, Ordered, Mandatory, Embed
- Association: Management**
 - Name: Management
 - Class: <none>
 - Constraints: 0
 - Properties: Persistent, Derived
- Role: Employee.manages -> Department**
 - Name: manages
 - Class: Department
 - Multiplicity: 0..*
 - Properties: Navigable, Multi, Ordered, Mandatory, Embed
- Role: Department.manager -> Employee**
 - Name: manager
 - Class: Employee
 - Multiplicity: 0..1
 - Properties: Navigable, Multi, Ordered, Mandatory, Embed

Figure 5: Basic Associations

Generating Pascal Code

In the Bold UML Model Editor select **Generate Code** from the **Tools menu**. You will be prompted to confirm or change the default include file created to hold class definitions. Accept the default, `HRClasses_Interface.inc` name. You will then be prompted with a name for the unit to store class information, again accept the default.

The generated code files will be loaded into the Delphi code editor as in Figure 6. The use of include files in Bold for Delphi is to ensure that code completion works correctly. The generated code files (`*.pas` & `*_Interface.inc`) should not be altered, as they will be overwritten on next code generation. Other generated code files (`*.inc`, excluding `*_Interface.inc`) will only be appended to; these are generated when using operations or derived attributes/relationships.



```
(*****  
(* This file is autogenerated *)  
(* Any manual changes will be LOST! *)  
(*****  
(* Generated 2/08/2002 8:14:52 PM *)  
(*****  
(* This file should be stored in the *)  
(* same directory as the form/datamodule *)  
(* with the corresponding model *)  
(*****  
(* Copyright notice: *)  
(* *)  
(*****  
  
unit HRClasses;  
  
{$DEFINE HRClasses_unitheader}  
{$INCLUDE HRClasses_Interface.inc}  
  
{ Includefile for methodimplementations }  
  
{$INCLUDE HRClasses.inc}
```

Figure 6: Generated Code

It is interesting to have a look at the generated classes.

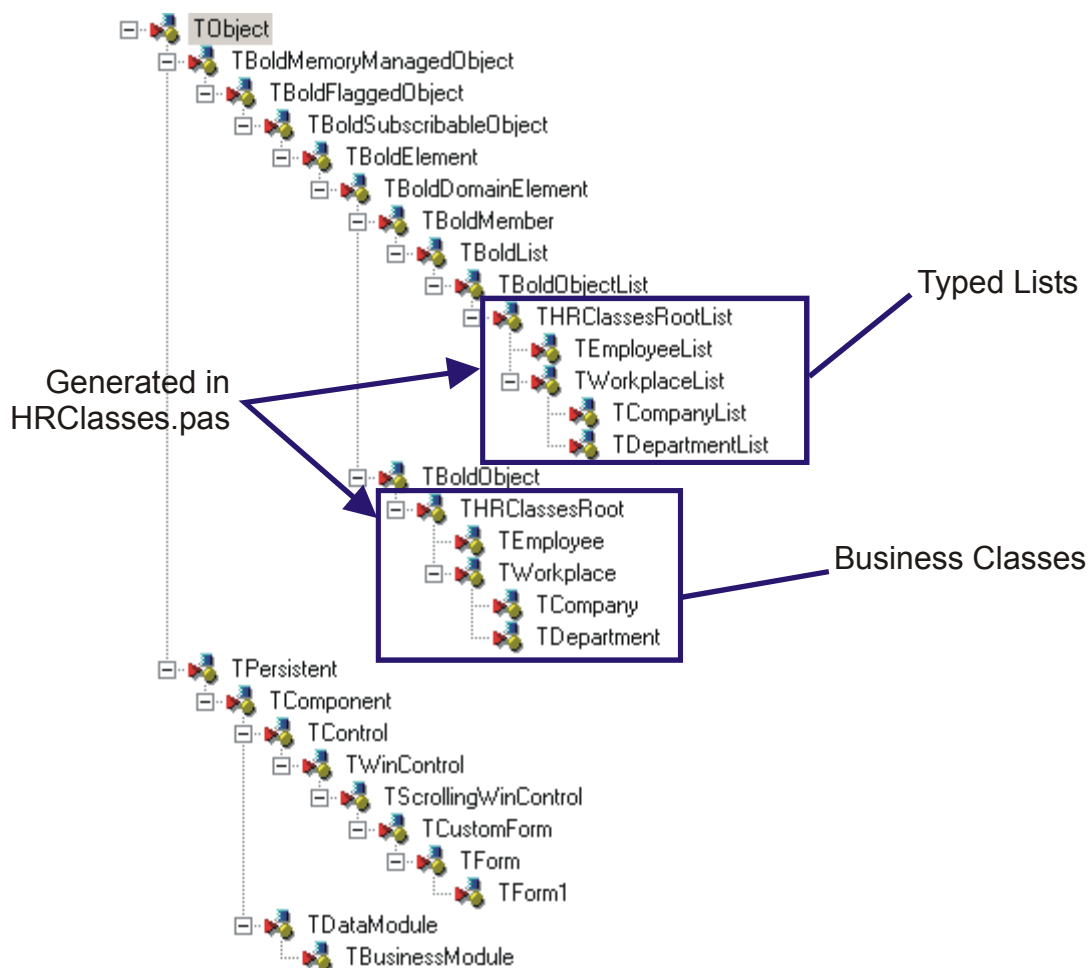


Figure 7: Class Hierarchy

Figure 7 shows the class hierarchy for the project. The classes marked in a blue outline are the auto-generated classes in HRClasses.pas. You will notice for each class created that Bold for Delphi automatically creates a matching typed list. This allows for clean code when working with lists of objects and multi-relations, the full benefit of compiler type checking is available to your code. Bold for Delphi also maintains the model hierarchy allowing for normal Delphi class comparisons when working with objects in code.

Adding a User Interface

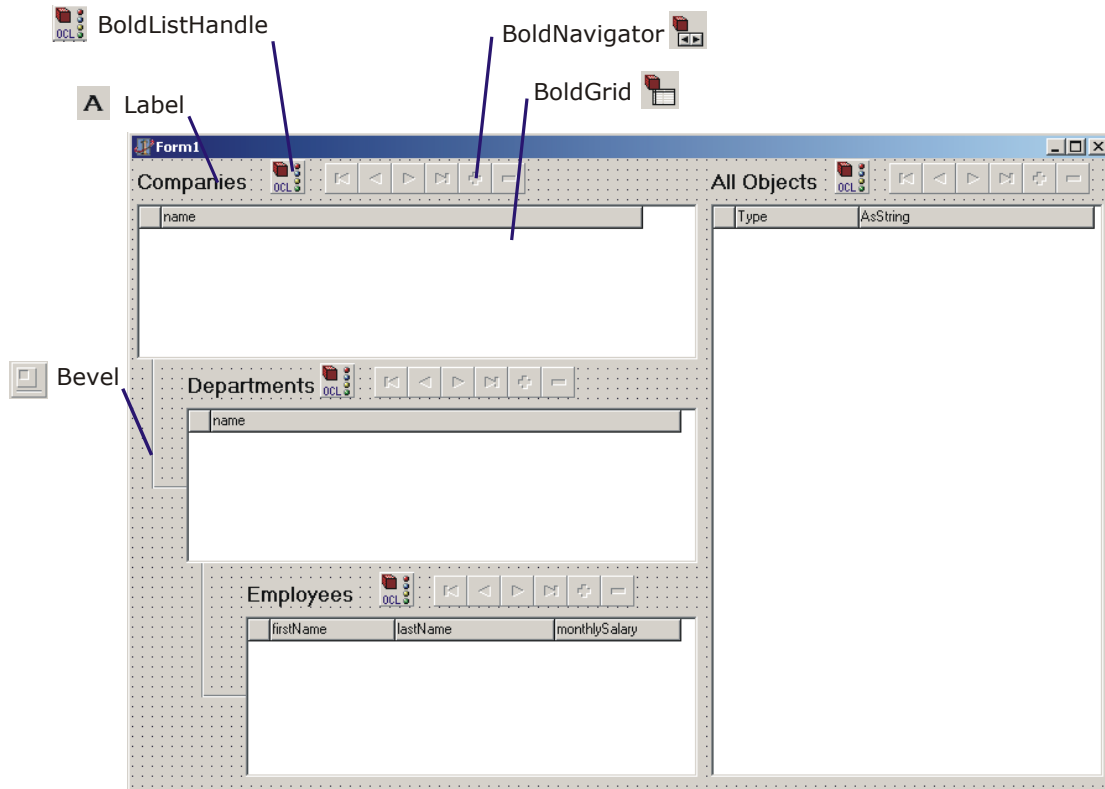
Simple Main Form

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-aware Controls to see the model on the data module.

From the **Bold Handles** component palette add 4 **BoldListHandle** components to the form. From the **Bold Controls** component palette add 4 **BoldNavigator** components and 4 **BoldGrid** components. Also add 4 standard Delphi **Label** & **Bevel** 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:

Companies

BoldListHandle

Name: CompanyList
RootHandle: BusinessModule.BoldSystemHandle1
Expression: Company.allInstances
Enabled: True

BoldNavigator

Name: CompanyNavigator
BoldHandle: CompanyList
Enabled: True

BoldGrid

Name: CompanyGrid
BoldHandle: CompanyList

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

Departments

BoldListHandle

Name: DepartmentList
RootHandle: CompanyList
Expression: departments
Enabled: True

Using the **CompanyList** handle as the root handle allows the **DepartmentList** handle to create a master detail relationship. The expression **department** means that this list will automatically contain all departments linked to the currently selected company.

BoldNavigator

Name: DepartmentNavigator
BoldHandle: DepartmentList
Enabled: True

BoldGrid

Name: DepartmentGrid
BoldHandle: DepartmentList

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

Employees

BoldListHandle

Name: EmployeeList
RootHandle: DepartmentList
Expression: employs
Enabled: True

Using the **DepartmentList** handle as the root handle allows the **EmployeeList** handle to create a master detail relationship. The expression **employs** means that this list will automatically contain all employees employed by the currently selected department.

BoldNavigator

Name: EmployeeNavigator
BoldHandle: EmployeeList
Enabled: True

BoldGrid

Name: EmployeeGrid
BoldHandle: EmployeeList

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

All Objects

BoldListHandle

Name: ObjectList

RootHandle: `BusinessModule.BoldSystemHandle1`
Expression: `HRClassesRoot.allInstances`
Enabled: `True`

BoldNavigator

Name: `ObjectNavigator`
BoldHandle: `ObjectList`
Enabled: `True`

BoldGrid

Name: `ObjectGrid`
BoldHandle: `ObjectList`

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

Running The Application

When you run the application add some entries for Company, Department and Employees. The main form will look similar to Figure 8.

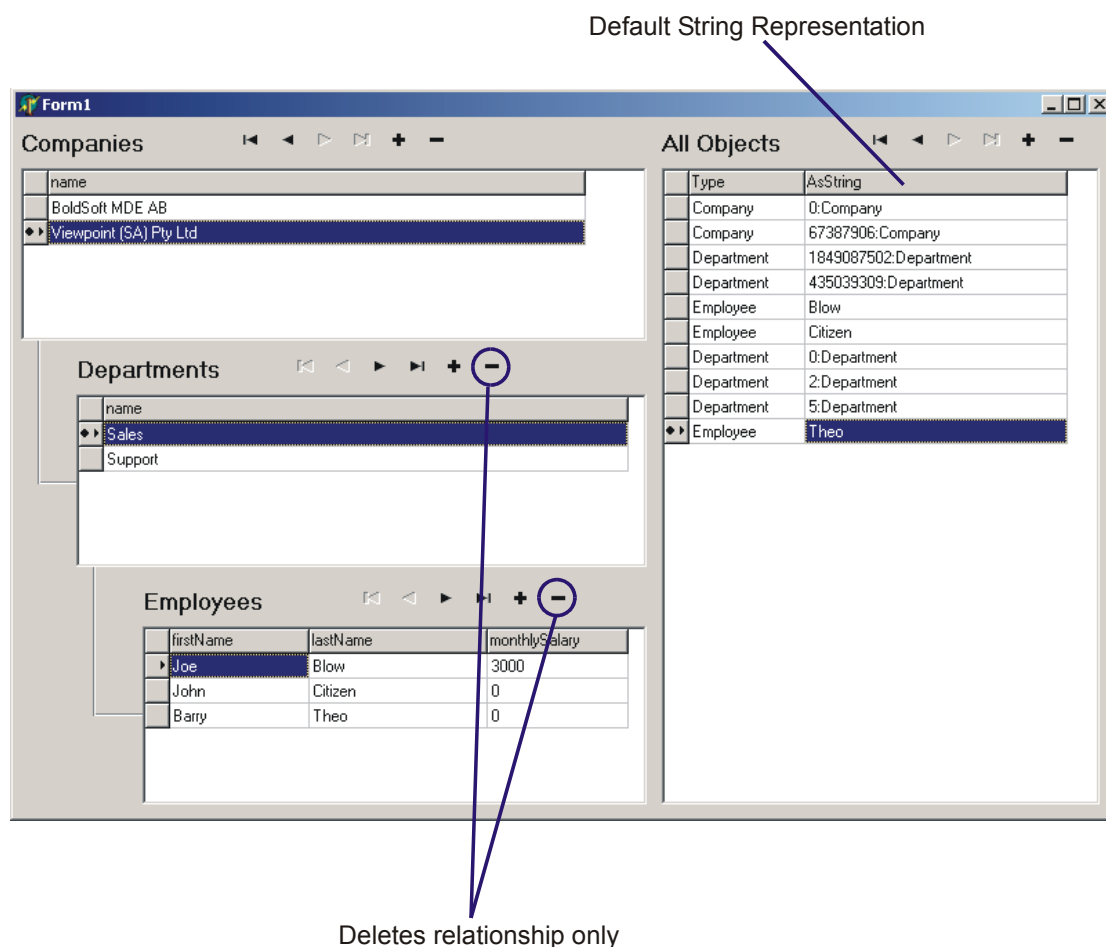


Figure 8: Basic Application at Runtime

When you add a new department, using the BoldNavigator, Bold for Delphi will automatically establish the relationship with the company, likewise adding employees adds the relationship to departments. When you delete from the Company grid or the All Objects grid the object is deleted. When you delete from the Department or Employee grid the object is **not** deleted, but the relationship is unlinked. This change in behavior is the default when you cascade list handles. This can be changed by altering the *BoldDeleteMode* property of the *TboldNavigator*. It is possible to set this to *dmUnlinkAllandDelete* to force the actual deletion of the object.

Double clicking a row will open the default editor for that object and allow drag and drop relationship linking.

The all objects grid is interesting; here Bold for Delphi has created a column to display the type of the object and a column to display the object 'AsString'. The AsString column is the string representation of the object. In the example Bold for Delphi has automatically used the lastName of employees to represent them but has created a mangled string for both companies and departments. The reason for this is both the company and department classes don't have any attributes (they rely on inheritance). Since this is confusing we can tell Bold for Delphi how to represent the objects in a more human readable way.

Default String Representation

Figure 9 shows the settings for each class. By using an OCL expression a human readable identifier can be created for each class.

For the Company class we simply use the company *name* as inherited from the Workplace class. For Department we grab the *name* of the *company* associated with the department, add a hyphen and append the department *name*. For the Employee class the aggregation of the *firstName* and *lastName* are used.

The figure consists of three vertically stacked screenshots of class configuration panels. Each panel has a title 'Class: [Class Name]' and contains several input fields and checkboxes. The 'Default string rep' field in each panel is highlighted with a red underline.

- Class: Company**
 - Name: Company
 - Stereotype:
 - Superclass: Workplace
 - Unit name:
 - Include file name:
 - Default string rep: name
 - Table mapping: Own
 - Checkboxes: Persistent (checked), Abstract (unchecked), Imported (unchecked), Remove Superclass on unboldify (unchecked), Remove class on unboldify (unchecked), Is Root Class (unchecked)
- Class: Department**
 - Name: Department
 - Stereotype:
 - Superclass: Workplace
 - Unit name:
 - Include file name:
 - Default string rep: company.name + '-' + name
 - Table mapping: Own
 - Checkboxes: Persistent (checked), Abstract (unchecked), Imported (unchecked), Remove Superclass on unboldify (unchecked), Remove class on unboldify (unchecked), Is Root Class (unchecked)
- Class: Employee**
 - Name: Employee
 - Stereotype:
 - Superclass: HRClassesRoot
 - Unit name:
 - Include file name:
 - Default string rep: firstName + ' ' + lastName
 - Table mapping: Own
 - Checkboxes: Persistent (checked), Abstract (unchecked), Imported (unchecked), Remove Superclass on unboldify (unchecked), Remove class on unboldify (unchecked), Is Root Class (unchecked)

Figure 9: Setting Default String Representation

This results in a much more readable display as shown in Figure 10.

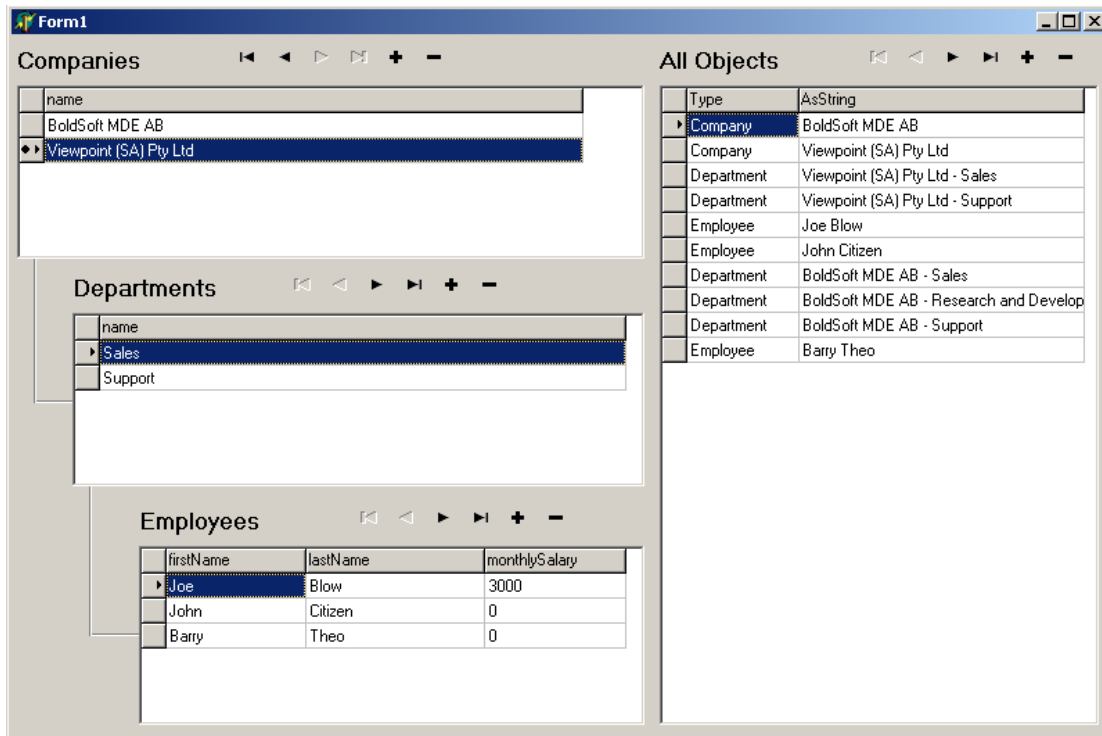


Figure 10: Application at Runtime (with readable object string representation)

Before we move in to write Pascal code let's add a new column to the department grid and display the manager for the department.

1. Right-click the DepartmentGrid and select 'Edit Columns'.
2. Add a new column in the column editor
3. Set the *BoldProperties.Expression* parameter to 'manager' (without the quotes).

Now when you assign a manager to the department the employee will be displayed in the grid. Because the result of the expression manager actually returns an employee object, the *TBoldGrid* will actually use the employee classes *Default String Representation* set earlier. This results in the employee's full name being displayed in the grid. (Hint: Use the default forms and *drag and drop* to set the manager for a department)

Using Code in Bold for Delphi Applications

Yet another powerful aspect of Bold for Delphi is the ability to use native Pascal objects for your business objects. When using traditional database programming the data has no type within the code you are writing (except for basic types like string or integer). When using the *Generate Code* functionality of Bold for Delphi you can use native Delphi classes that wrap the Domain Classes you have created in the Bold UML Modeler.

The benefits of being able to do this are enormous in terms of:

- Code readability.
- Code maintainability.
- Code accuracy.
- Development speed.

When coding with Bold for Delphi you inherently gain the benefit of Delphi's strong type-checking compiler. Delphi's integrated code editor prompts you with all your classes' information via code completion. Your code reads closer to the level of intent of what you are trying to achieve, that means your code is a lot closer to becoming self documenting.

Enhancing the application with code

We are going to add a pop-up menu to the employee grid and add the ability to delete, add new and promote an employee to manager.

- 1) Add **HRClasses** to the MainForm's unit uses clause. This is required to allow use of your domain classes in the main form.
- 2) Add a **TPopupMenu** component to the form and set its name to **EmployeePopupMenu**.
- 3) Set the **EmployeeGrid**'s **PopupMenu** property to this new popup menu.
- 4) Add a menu item to the popup menu with the caption 'Make Manager'.

This sets up the basic framework for adding our code. Begin by setting the **EmployeeGrid**'s **OnContextPopup** event with the following code:

```
procedure TForm1.EmployeeGridContextPopup(Sender: TObject;
  MousePos: TPoint; var Handled: Boolean);
var
  EmployeeAssigned: Boolean;
begin
  // Enable/Disable Menu items depending if an Employee is
  // selected.
  EmployeeAssigned := EmployeeGrid.CurrentBoldElement is TEmployee;
  MakeManager1.Enabled := EmployeeAssigned;
end;
```

This code controls the availability of the popup menu item depending on if an object is available in the appropriate grid. If no object is selected in a grid the grid's *CurrentBoldElement* property will return *nil*. If an object is selected this property will return that item as a *TBoldElement*, this is a base class for all classes created from the model, and all lists of classes as well. (Refer to Figure 7 from earlier in this article).

Add an *OnClick* event handler for the 'Make Manager' menu item.

```
procedure TForm1.MakeManager1Click(Sender: TObject);  
var CurrentEmployee: TEmployee;  
begin  
  // Check that the object is an Employee object and  
  // assigned it to our local variable  
  if EmployeeGrid.CurrentBoldElement is TEmployee then  
    begin  
      CurrentEmployee := TEmployee(EmployeeGrid.CurrentBoldElement);  
  
      // Check if the employee works for a department and set the  
      // employee as the manager  
      if assigned(CurrentEmployee.worksFor) then  
        CurrentEmployee.worksFor.manager := CurrentEmployee;  
    end;  
end;
```

The comments in the code should be almost unnecessary. The power of working with native Pascal code is evident. Even the mundane task of validating the selection is simplified using the Delphi **is** operator and a typecast to a local variable. The sheer simplicity of using a statement like 'CurrentEmployee.worksFor.manager := CurrentEmployee' in terms of maintainability and readability is incredible. This can be read as 'For the current employee, set the manager of the department for which they work to that employee'.

It would be good object orientated design to move this code into the *Employee* class as an operation call *Promote*. Then any error checking would be encapsulated in one location and any business rules about managing the promotion process is abstracted from the presentation layer.

Introduction to Object lifecycles

Many Bold for Delphi controls support the creation and deletion of objects, for example the BoldNavigator used in the example application. Other methods exist to work with object creation and deletion as well. By using Delphi Code you can add special creation code to class operations or derived attributes, make use of the code in general GUI code, like buttons OnClick events. Bold for Delphi also supports options within the model itself to help manage an objects lifecycle.

Using Delphi Code

Add two more menu items to the employee grid's popup menu with the captions 'Add New Employee' and 'Delete Employee'.

Enhance the previously created Context Popup event of the Employee grid to include the following:

```
procedure TForm1.EmployeeGridContextPopup(Sender: TObject;
  MousePos: TPoint; var Handled: Boolean);
var
  EmployeeAssigned: Boolean;
begin
  // Enable/Disable Menu items depending if an Employee is
  // selected.
  EmployeeAssigned := EmployeeGrid.CurrentBoldElement is TEmployee;
  MakeManager1.Enabled := EmployeeAssigned;
  DeleteEmployee1.Enabled := EmployeeAssigned;

  // Enable/Disable Menu items depending if a department is selected.
  AddNewEmployee1.Enabled :=
    DepartmentGrid.CurrentBoldElement is TDepartment;
end;
```

Add on *OnClick* event handler for the 'Add New Employee' menu item:

```
procedure TForm1.AddNewEmployee1Click(Sender: TObject);
var NewEmployee: TEmployee;
    CurrentDepartment: TDepartment;
begin
  // Because the Context Pop-up code has already checked
  // that the current element is a TDepartment we can
  // typecast to TDepartment safely
  CurrentDepartment := TDepartment(DepartmentGrid.CurrentBoldElement);

  // Create a new Employee object
  NewEmployee := TEmployee.Create(CurrentDepartment.BoldSystem);
  // Set the department of the new employee to the current
  // department
  NewEmployee.worksFor := CurrentDepartment;
end;
```

The key concept from this code is the ease of creating a new object; just create an instance of the class like any other Delphi object. Of course then you can proceed to use that object however you like.

Deleting an object is just as easy; add the following code to the *OnClick* event of the 'Delete Employee' menu item:

```
procedure TForm1.DeleteEmployee1Click(Sender: TObject);  
var CurrentEmployee: TEmployee;  
begin  
    // Check that the object is an Employee object and  
    // Delete it  
    if EmployeeGrid.CurrentBoldElement is TEmployee then  
        TEmployee(EmployeeGrid.CurrentBoldElement).Delete;  
end;
```

Model driven object life management

Some common operations in Bold for Delphi support automatic maintenance of an object's existence.

One of the properties of an association's role is *Delete Action*. This allows Bold for Delphi to control how objects in the relationship are handled when the object is deleted. The following options are available:

- **Allow**: The object can be removed. In this case the links will be severed.
- **Prohibit**: TBoldObject.CanDelete will return false. An attempt to delete the object anyway will raise an exception.
- **Cascade**: The related objects will be deleted as well.
- **<Default>**: See below

The default depends on the value of the *Aggregation* property:

- **none**: Allow
- **aggregate**: Prohibit
- **composite**: Cascade

We will now enhance our model to allow *Departments* to delete all *Employees* it has, when it is deleted. Likewise a *Company* will delete all *Departments* when it is deleted. Change the *Delete Action* property for the *Business Unit*, *departments* role and the *Employment*, *employs* role to *Cascade*.

Role: Company.departments -> Department

Name	departments	<input checked="" type="checkbox"/> Navigable
Stereotype		<input checked="" type="checkbox"/> Multi
Class	Department	<input type="checkbox"/> Ordered
Multiplicity	0..*	<input type="checkbox"/> Mandatory
Aggregation	none	<input type="checkbox"/> Embed
Visibility	public	
Changeability	changeable	
Delete action	Cascade	

Figure 11: Business Unit, departments role

Role: Department.employs -> Employee

Name	employs	<input checked="" type="checkbox"/> Navigable
Stereotype		<input checked="" type="checkbox"/> Multi
Class	Employee	<input type="checkbox"/> Ordered
Multiplicity	0..*	<input type="checkbox"/> Mandatory
Aggregation	none	<input type="checkbox"/> Embed
Visibility	public	
Changeability	changeable	
Delete action	Cascade	

Figure 12: Employment, employs role

Now the model will automatically cleanup the designated objects.

Run the application and checkout the change. Previously when deleting a department, the employees would still remain (visible in the All Object grid). Now they are correctly cleaned up.

Derived & Reverse Derived Attributes

So far we have only use simple attributes within our application. Each attribute has been a basic type (string & currency) and saved/restored from the database. Similar to calculated fields in a traditional Delphi database application, in Bold for Delphi it is possible to calculate the value of an attribute 'on the fly'. I say similar because Bold for Delphi extends the calculate field concept far beyond the capabilities of traditional Delphi database applications.

The most significant difference is the ability to make a derived attribute write-able. This is called reverse derived. The process of enabling this functionality is simple in a Bold for Delphi application.

Adding the new attributes

The new model will include a derived attribute on the *Workplace* class and a reverse-derived attribute on the *Employee* class. The *Workplace* class now includes the *monthlyCost* attribute, which is then inherited and overridden, in both the *Company* and *Department* classes. The *Employee* class now includes the attribute *fullName* which, when changed, can make the appropriate changes to the *firstName* and *lastName* attributes.

Figure 13 shows the updated UML model.

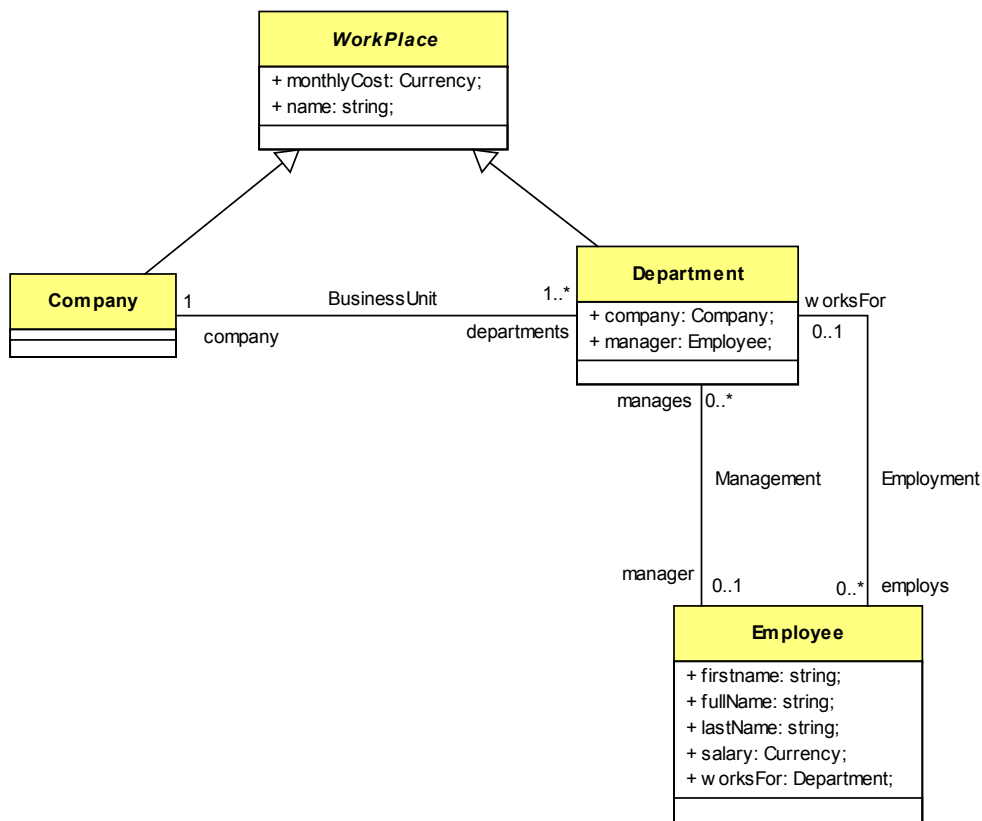


Figure 13: UML Model including new attributes

Figure 14 shows the new attributes in the Bold UML Model editor.

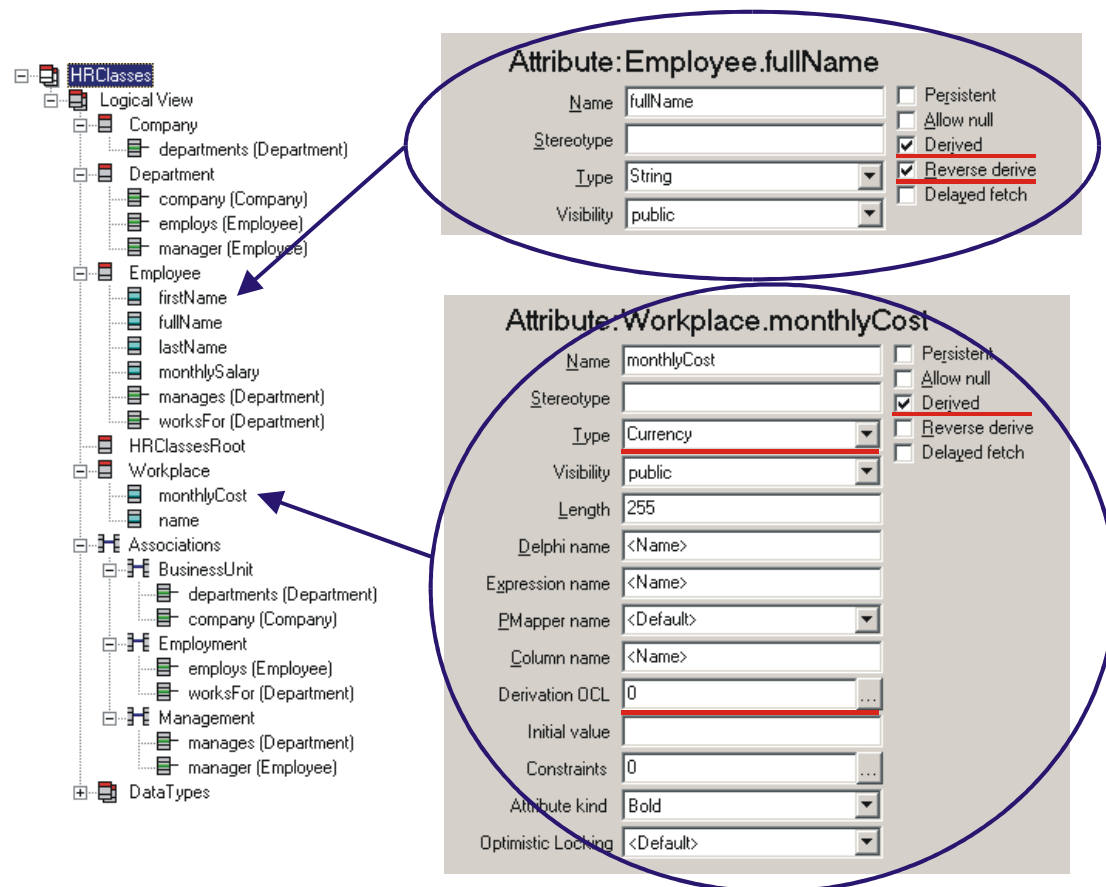


Figure 14: Adding the new attributes

After adding the new attributes it is important to use the **Generate Code** menu option from the Bold UML Editor to ensure the correct procedure stubs are generated. After generating the code a new HRClasses.inc file is created and loaded in the IDE editor.

The following method stubs are created for deriving and reverse deriving the employee's fullname:

```
procedure TEmployee._fullName_DeriveAndSubscribe(DerivedObject: TObject;  
Subscriber: TBoldSubscriber);
```

```
procedure TEmployee._fullName_ReverseDerive(DerivedObject: TObject);
```

Methods are not created for the derived *monthlyCost* attribute because we supplied an OCL expression. The expression returns a default value of zero; this will be overridden in each descendant class later.

Reversed Derived

Add the following code for the employee's *fullName* attribute:

```
procedure TEmployee._fullName_DeriveAndSubscribe(DerivedObject: TObject;  
Subscriber: TBoldSubscriber);  
begin  
    // Set the fullname  
    M_FullName.AsString := firstName + ' ' + lastName;  
  
    // subscribe to notifications of either the first  
    // or last name changing  
    M_FirstName.DefaultSubscribe(subscriber);  
    M_LastName.DefaultSubscribe(subscriber);  
end;  
  
procedure TEmployee._fullName_ReverseDerive(DerivedObject: TObject);  
var aFullName: String;  
    p: integer;  
begin  
    // strip away leading and trailing spaces  
    aFullName := trim(fullName);  
    // Check if a space was found  
    p := pos( ' ', aFullName );  
    if p <> 0 then  
        begin  
            // the first name is everything up to the first space  
            // the last name is the rest  
            firstName := copy( aFullName, 1, p-1 );  
            lastName := trim(copy(aFullName, p+1, maxint ));  
        end else  
        begin  
            // No space found, the first name is everything,  
            // the last name is set blank  
            firstName := aFullName;  
            lastName := '';  
        end;  
end;
```

To understand what these methods are doing it is helpful to have a look at the declaration for the TEmployee class. The following has been stripped down to include on properties and methods important to the description of the above methods.

```
TEmployee = class(THRClassesRoot)  
    protected  
        procedure _fullName_DeriveAndSubscribe(DerivedObject: TObject;  
            Subscriber: TBoldSubscriber); virtual;  
        procedure _fullName_ReverseDerive(DerivedObject: TObject);  
            virtual;  
  
    public  
        property M_firstName: TBAStrng read _Get_M_firstName;  
        property M_lastName: TBAStrng read _Get_M_lastName;  
        property M_fullName: TBAStrng read _Get_M_fullName;  
        property firstName: String read _GetfirstName write  
            _SetfirstName;  
        property lastName: String read _GetlastName write _SetlastName;  
        property fullName: String read _GetfullName write _SetfullName;  
end;
```

From this you can clearly see the methods we have just entered (in the protected section). However, it's interesting to see that we seem to have a double up of properties for each attribute.

For each property Bold for Delphi creates, it also creates the same property with a `M_` prefix. This second property returns a special Bold for Delphi object. This object is used to represent the base type of the property, as an object. The reasons Bold for Delphi creates these runtime information classes are many:

- Allow for subscriptions to allow notifications when the value changes.
- Cache old value so changes can be discarded without refreshing from the database.
- Allow for fine-grained control of persistence at the attribute level.
- To support optimistic locking.

If first we look at the method declaration:

```
procedure TEmployee._fullName_DeriveAndSubscribe(DerivedObject:
  TObject; Subscriber: TBoldSubscriber);
```

Here we have two parameters, *DerivedObject* and *Subscriber*. The *DerivedObject* in this case is the internal *TBAString* object used to represent the *fullName* attribute. This is the same object return form the *M_FullName* property. In the code I have decided to directly access the *M_FullName property* as this is more readable. We could have typecast the *DerivedObject* object as a *TBAString* if we chose. The *Subscriber* is the internal subscription object for this property, it is our responsibility to use this object to subscribe to any other object attributes that we need to know have changed. This subscription allows for the *fullName* property to respond to the change of *firstName* or *lastName* and correctly show the updated value.

Setting the property value:

```
M_FullName.AsString := firstName + ' ' + lastName;
```

The employee's full name is simply the first name and last name separated with a space.

```
M_FirstName.DefaultSubscribe(subscriber);
M_LastName.DefaultSubscribe(subscriber);
```

Here the subscriber for the *fullName* attribute attaches itself to the subscription mechanism for each of the other name attributes.

If you run the application and use the default object form, you can see the read/write behavior of the new *fullName* attribute in action.

Derived Attributes with OCL

In the descendant class of *Workplace* we can override the OCL for the *monthlyCost* attribute. This allows for the full use of polymorphism by changing the behavior of the attribute for each descendant class.

To override OCL derived attributes in Bold for Delphi place an entry in the *Derived Expressions* property of the class. This is shown in Figure 15.

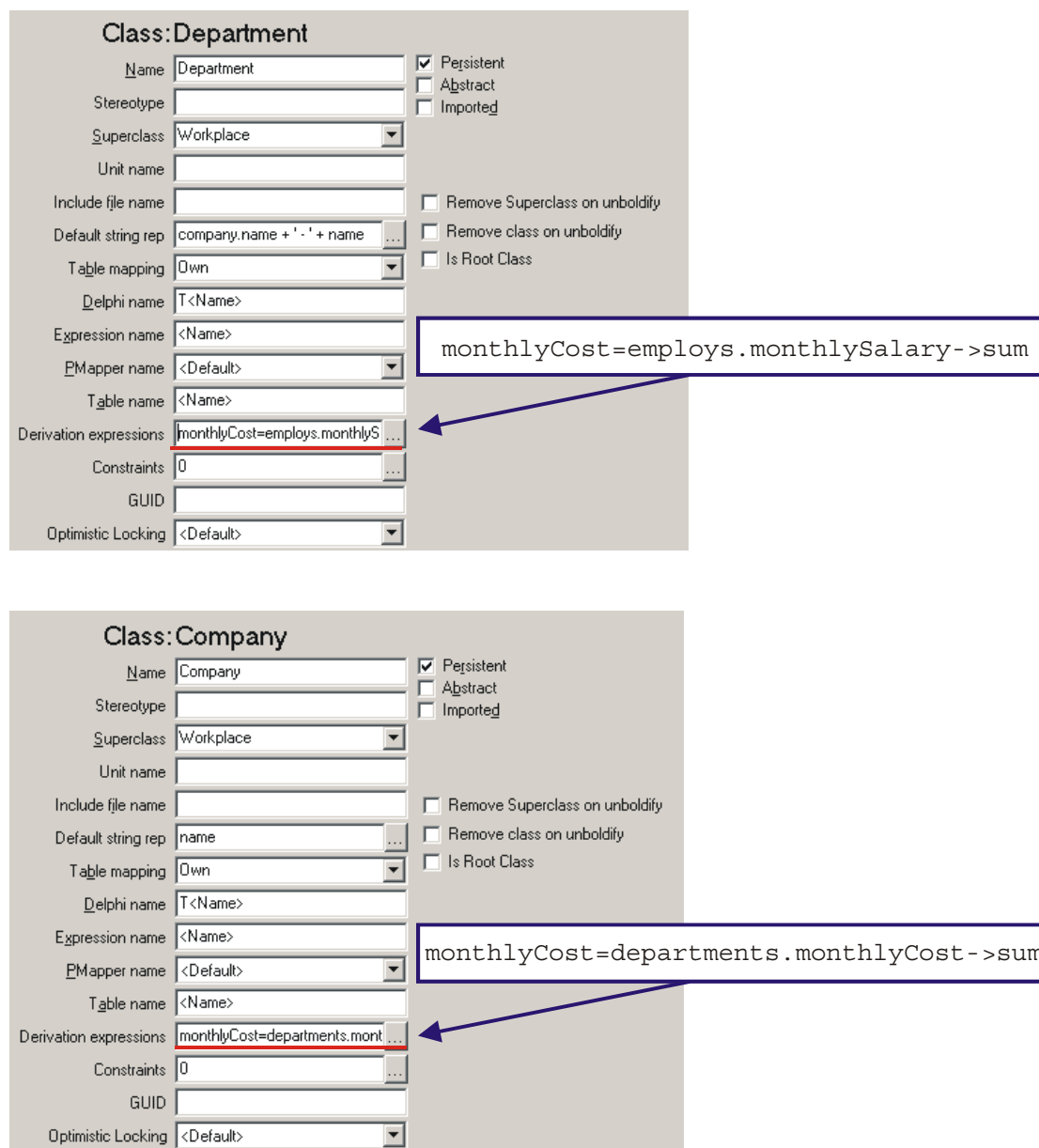


Figure 15: Overriding derived OCL expressions

The *Derivation Expressions* property of the class contains a string list of all the overridden derived OCL expressions. The format of each line is:

```
<inherited attribute name>=<new OCL expression>
```

In the example above the *Department* class now calculates the *monthlyCost* value as being the sum of all employee's salaries for that department. The *Company* class

calculates the *monthlyCost* value as being the sum of all monthly costs for each department.

For both the Company and the Department grid we can add a new column and set the OCL expression to *monthlyCost*. At runtime the screen will look similar to Figure 16.

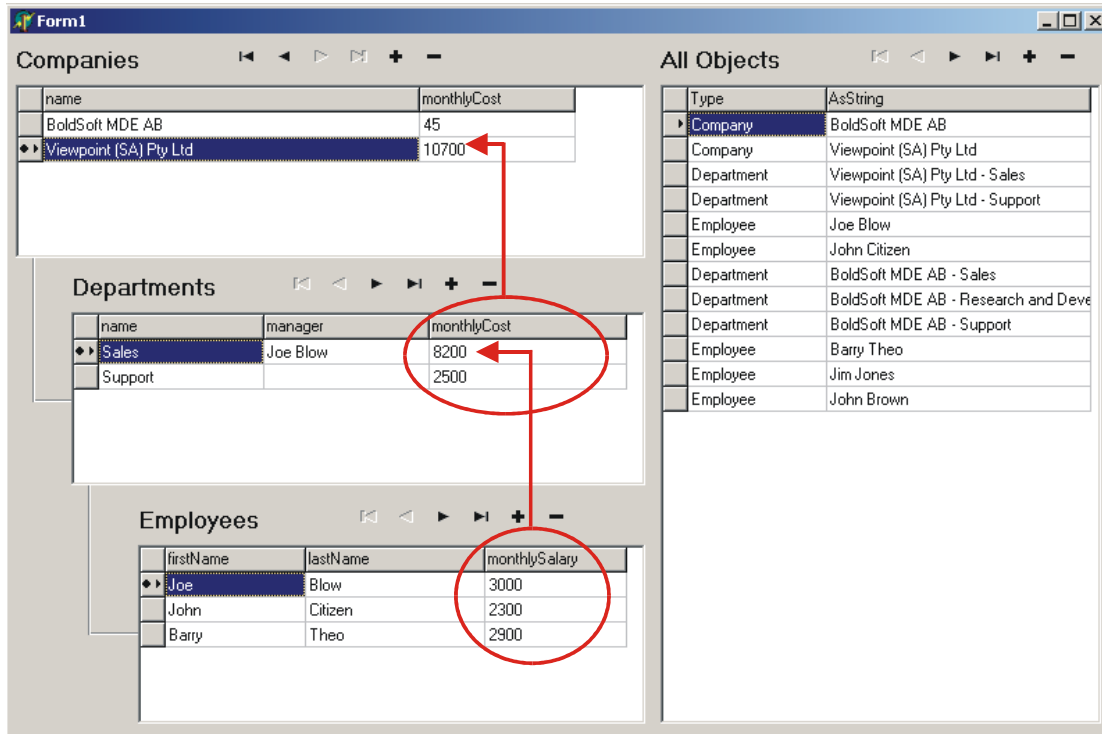


Figure 16: Derived Attributes at Runtime

Operations

So far we have discussed attributes and relationships, another very important part of Bold for Delphi is *Operations*. Operations allow you to add behavior to your classes and are vital to allowing you to implement all the required business rules for an application.

Operations are implemented using standard Delphi Object Pascal. They are the equivalent to methods in standard Delphi classes. Operations are used to implement tasks within your classes, like *LightBulb.change*.

Adding an Operation

Add the *adjustSalary* operation shown in the updated UML model in Figure 17.

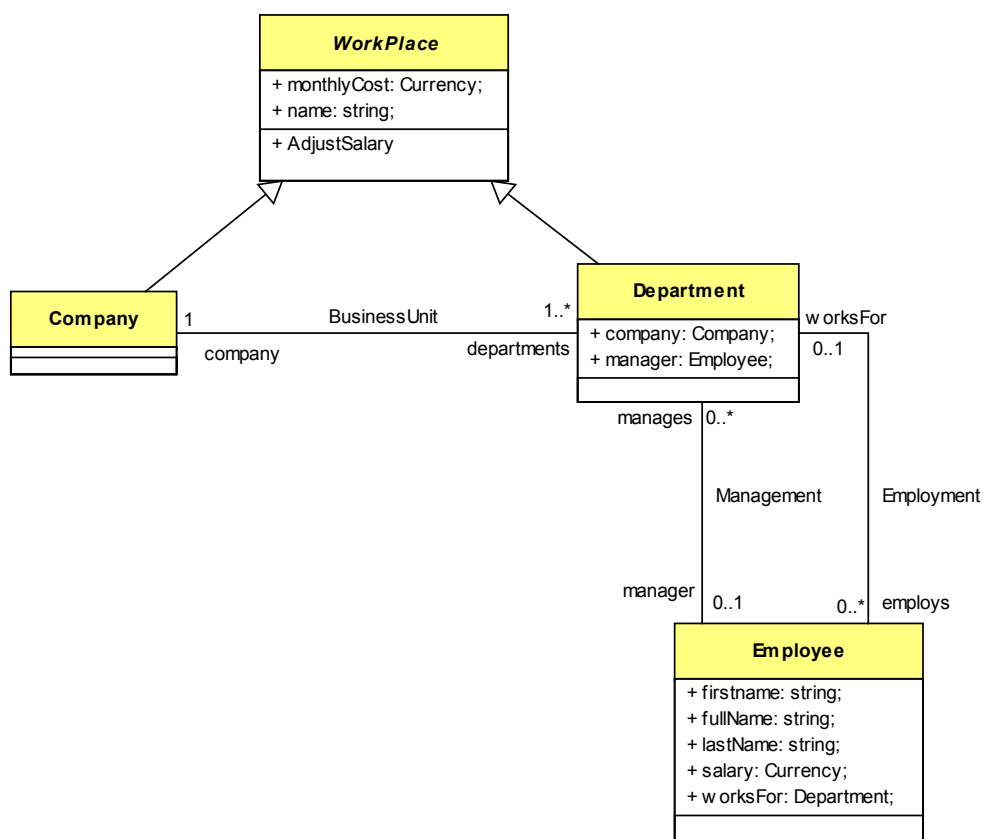


Figure 17: Adding an Operation

The *adjustSalary* operation doesn't need to do anything in the base *Workplace* class. In both the *Company* and *Department* classes it will be overridden to supply different behavior for each class.

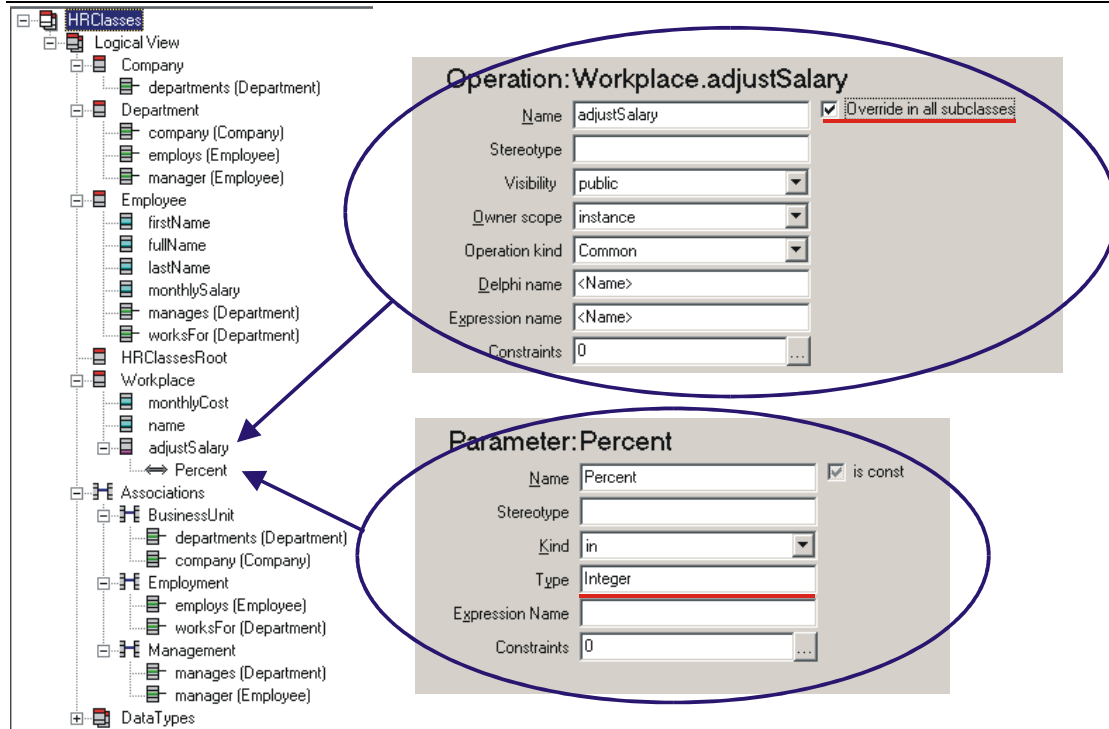


Figure 18: Adding the adjustSalary Operation

After adding the operation you will need to use *Generate Code* from the *Tools* menu. The following method stubs are added to HRClasses.inc.

```
procedure TWorkplace.adjustSalary(Percent: Integer);
begin
end;
```

```
procedure TCompany.adjustSalary(Percent: Integer);
begin
  inherited;
end;
```

```
procedure TDepartment.adjustSalary(Percent: Integer);
begin
  inherited;
end;
```

No behavior is required in the *Workplace* class so we will only add code to the *Company* and *Department* implementations.

```
procedure TCompany.adjustSalary(Percent: Integer);
var counter: Integer;
begin
  inherited;
  // Loop thru all departments and call
  // adjustSalary for each one
  for Counter := 0 to departments.Count -1 do
    departments[counter].adjustSalary(Percent);
end;
```

```
procedure TDepartment.adjustSalary(Percent: Integer);  
var counter: Integer;  
    SalaryChange: Currency;  
begin  
    inherited;  
    // Loop thru all employee's and adjust their  
    // salary  
    for Counter := 0 to employs.Count -1 do  
    begin  
        SalaryChange := employs[counter].monthlySalary * (Percent / 100);  
        employs[counter].monthlySalary := employs[counter].monthlySalary  
            + SalaryChange;  
    end;  
end;
```

To show the effect of this new operation we will add a pop-up menu to both grids and allow the user to adjust salaries.

Add a new TpopupMenu to the form and call it SalaryPopupMenu.
Set both the CompanyGrid and the DepartmentGrid to use this new poupmenu by setting the PopupMenu parameter of each grid.

Add a private field to the form's class declaration:

```
type  
    TForm1 = class (TForm)  
        ...  
    private  
        { Private declarations }  
        FCurrentWorkplace: TWorkplace;  
    public  
        { Public declarations }  
    end;
```

Add the following code to the OnContextPopup event of the DepartmentGrid:

```
procedure TForm1.DepartmentGridContextPopup(Sender: TObject;  
    MousePos: TPoint; var Handled: Boolean);  
begin  
    // Initialize menu item and form private field  
    AdjustSalary1.Enabled := False;  
    FCurrentWorkplace := nil;  
  
    // Check that the object is a Workplace object and  
    // assigned it to our local variable  
    if DepartmentGrid.CurrentBoldElement is TWorkplace then  
    begin  
        AdjustSalary1.Enabled := True;  
        FCurrentWorkplace :=  
            TWorkplace(DepartmentGrid.CurrentBoldElement);  
    end;  
end;
```

Add the following code to the OnContextPopup event of the CompanyGrid:

```
procedure TForm1.CompanyGridContextPopup(Sender: TObject; MousePos: TPoint;  
var Handled: Boolean);  
begin  
    // Initialize menu item and form private field  
    AdjustSalary1.Enabled := False;  
    FCurrentWorkplace := nil;  
    // Check that the object is a Workplace object and  
    // assigned it to our local variable  
    if CompanyGrid.CurrentBoldElement is TWorkplace then  
        begin  
            AdjustSalary1.Enabled := True;  
            FCurrentWorkplace :=  
                TWorkplace(CompanyGrid.CurrentBoldElement);  
        end;  
end;
```

Both of these events are very similar. They check that a valid *Workplace* is selected in the grid and assign it to the local form's private field *FcurrentWorkplace*. The popup menu can then use this.

Add a menu item to the popup menu with the caption 'Adjust Salary'. In the menu items OnClick event add the following code:

```
procedure TForm1.AdjustSalary1Click(Sender: TObject);  
var PercentStr: String;  
    PercentInt: Integer;  
begin  
    // Ensure that the local form field is assigned  
    // an object  
    if assigned(FCurrentWorkplace) then  
        begin  
            // Display a pop-up requesting the percent to  
            // adjust the salaries by  
            PercentStr := '10';  
            if InputQuery('Salary Adjustment', 'Enter adjustment percentage',  
                PercentStr) then  
                begin  
                    try  
                        // call the adjustSalaries method of with the supplied percent  
                        PercentInt := StrToInt(PercentStr);  
                        FCurrentWorkplace.adjustSalary(PercentInt);  
                    except  
                        on EConvertError do  
                            MessageDlg('Invalid Percent entered!', mtError, [mbOK], 0);  
                    end;  
                end;  
        end;  
end;
```

The above code works for both grids because we use the base class of the objects making the code type compatible. This is an enormous benefit in reducing redundant code and helps ensure code re-use. The method simply prompts for an adjustment percentage and calls the *adjustSalary* method of the object. If the object is a *Company* then the *adjustSalary* method of that class is used, if the object is a *Department* then that version of the method is used. Fortunately that fact doesn't need to be known by the popup code above, Object Pascal and Bold for Delphi takes care of this for us.

Derived Relationships

With the power of derived, reverse derived attributes and operations it is easy to see how extending a Bold for Delphi application to truly integrate business rules into the model is easy. Bold for Delphi can extend this even further by deriving relationships 'on the fly'. The advantage of this is, as previously discussed, is reduced complexity, centralized business rules, ease of maintenance. This also reduces redundancy of data, sometime when implementing a database the rules of normalization are broken to accommodate reports or programming requirements. Data that can be easily derived from existing information is instead duplicated in one or more tables. This can result in problems as the application evolves or problems during normal operation ensuring the data is kept synchronized.

By using derived relationships, the benefit of having the information readily available and without redundant data can help ease the application development and maintenance. Figure 19 shows two new relationships in our model, *Workforce* and *Managed By*.

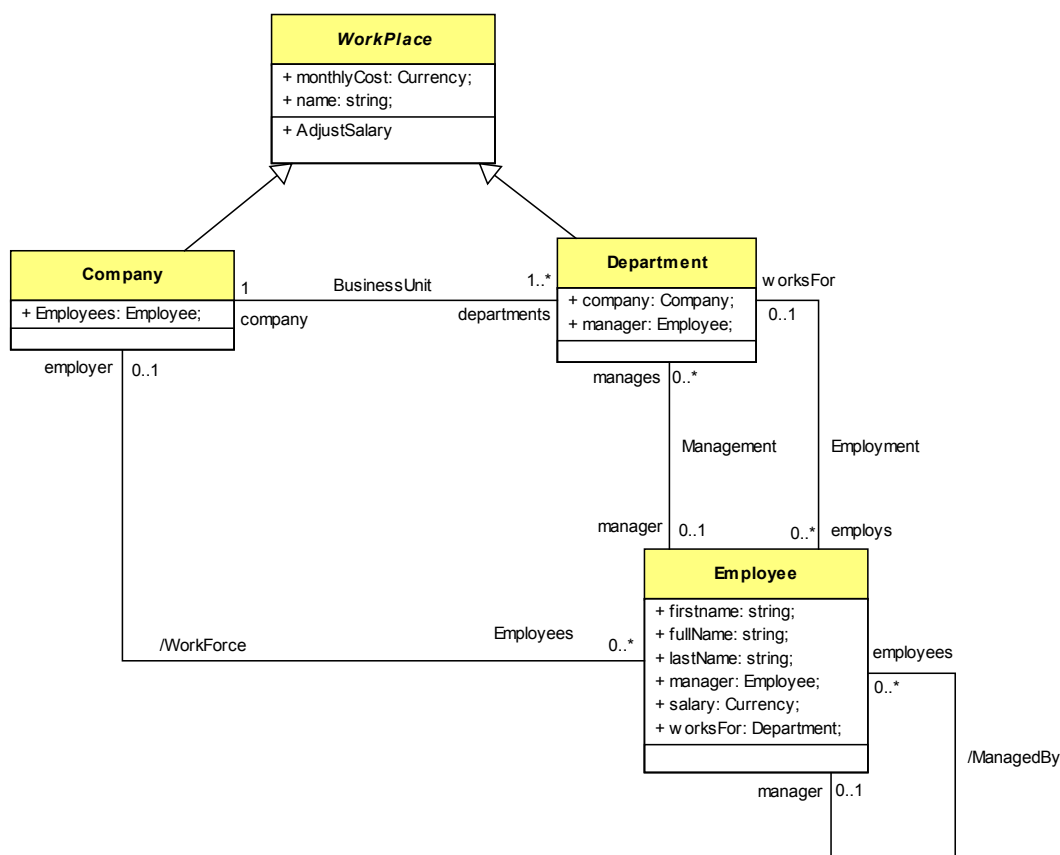


Figure 19: Derived Relationships

The *Managed By* relationship represents the manager for that employee; this is simply whoever is managing the department the employee works for. The *Workforce* relationship represents the employer of the employee; this is the company that owns the department that the employee works for.

Adding the Managed By relationship

Add the *Managed By* relationship as per Figure 20:

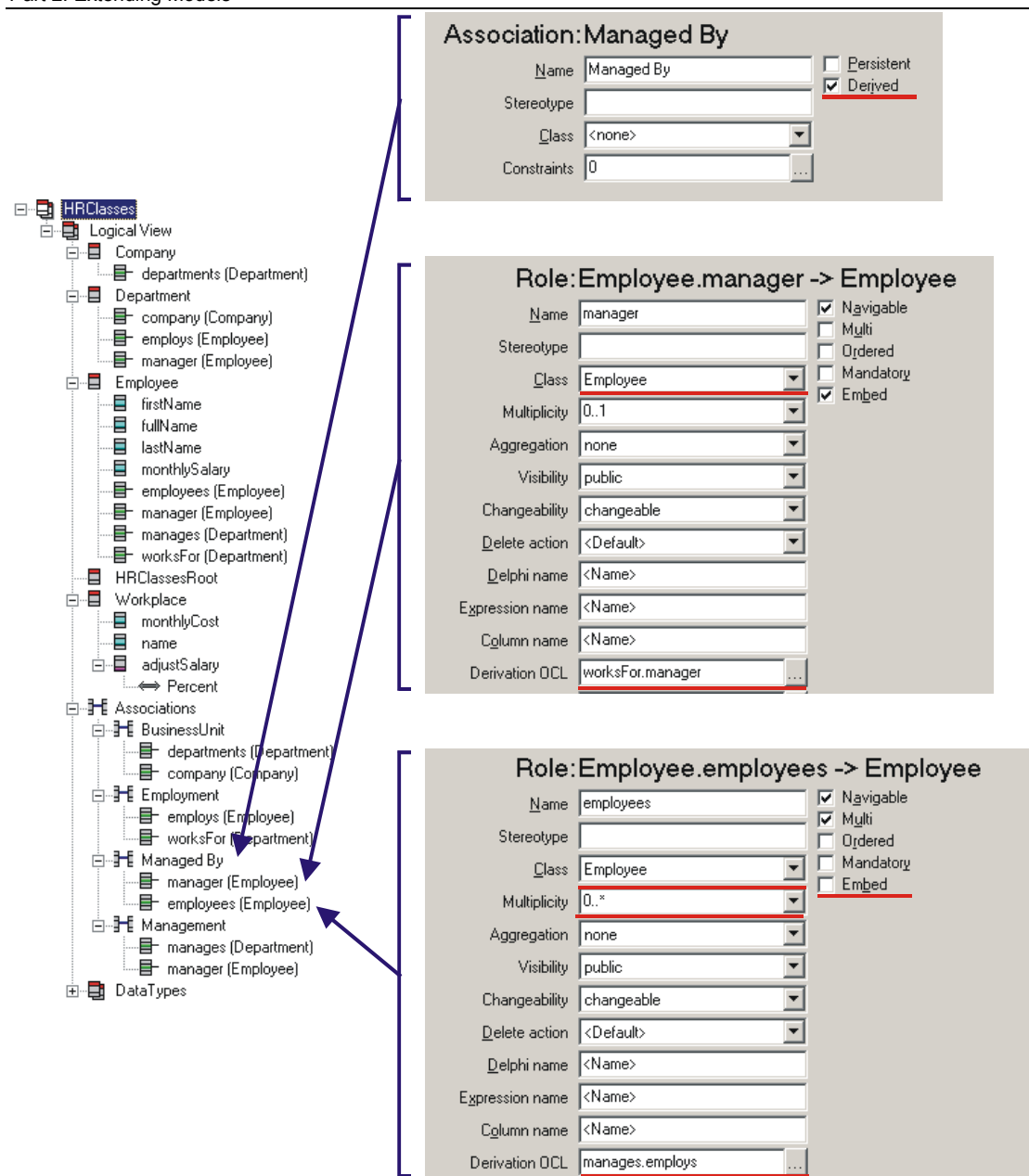


Figure 20: Derived Relationship – Managed By

The OCL expression `worksFor.manager` will return the employee who manages the department the current employee works for. The OCL expression `manages.employs` will return a list of employees. This list is the combination of all employees that work for all the departments that the current employee manages.

This behavior can be observed at runtime using the Bold for Delphi default object forms. Open an employee and drag a few departments into the *manages* tab. The *employees*' tab now lists all employees for the departments managed by that employee.

Adding the Workforce relationship

Add the *Workforce* relationship as per Figure 21:

The image displays the Bold IDE interface. On the left is the 'HRClasses' logical view tree, showing a hierarchy of classes and associations. Three blue arrows point from specific elements in the tree to three configuration panels on the right:

- Association: Workforce**: This panel shows the configuration for the 'Workforce' association. The 'Name' is 'Workforce', 'Stereotype' is empty, 'Class' is '<none>', and 'Constraints' is '0'. The 'Derived' checkbox is checked.
- Role: Employee.employer -> Company**: This panel shows the configuration for the 'employer' role. The 'Name' is 'employer', 'Stereotype' is empty, 'Class' is 'Company', and 'Multiplicity' is '0..1'. The 'Navigable', 'Embed', and 'Derivation OCL' ('worksFor.company') fields are highlighted with red boxes.
- Role: Company.employees -> Employee**: This panel shows the configuration for the 'employees' role. The 'Name' is 'employees', 'Stereotype' is empty, 'Class' is 'Employee', and 'Multiplicity' is '0..*'. The 'Navigable', 'Multi', and 'Derivation OCL' ('departments.employees') fields are highlighted with red boxes.

Figure 21: Derived Relationship – Workforce

The behavior can easily be shown by using the default Bold for Delphi object forms at runtime. The new *employee* tab for the *Company* class shows all employees for all departments. The *Employer* field for the *Employee* class now shows the employer.

One-way Relationships

We will now take a look at a one-way relationship, although not too different from what we have already done, it will give us an opportunity to explore some more of the options available with relationships.

In the UML model in Figure 22 the relationship *Top Salaries* has been added. This provided an association from *Workplace* to *Employee* identifying *highlyPaidEmployees*. To help facilitate the use of this new relationship the attribute *highSalaryThreshold* has been added to the *Department* class.

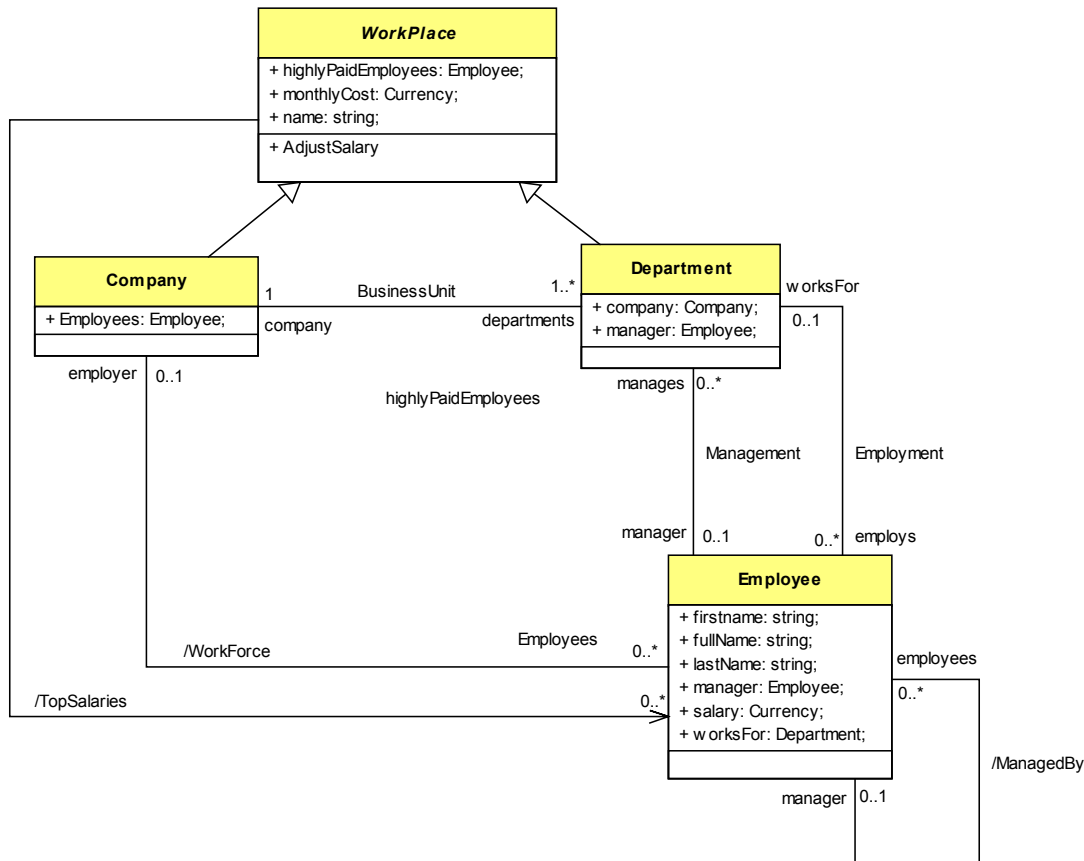


Figure 22: One-way Relationship

This is implemented in the Bold UML Model Editor as follows:

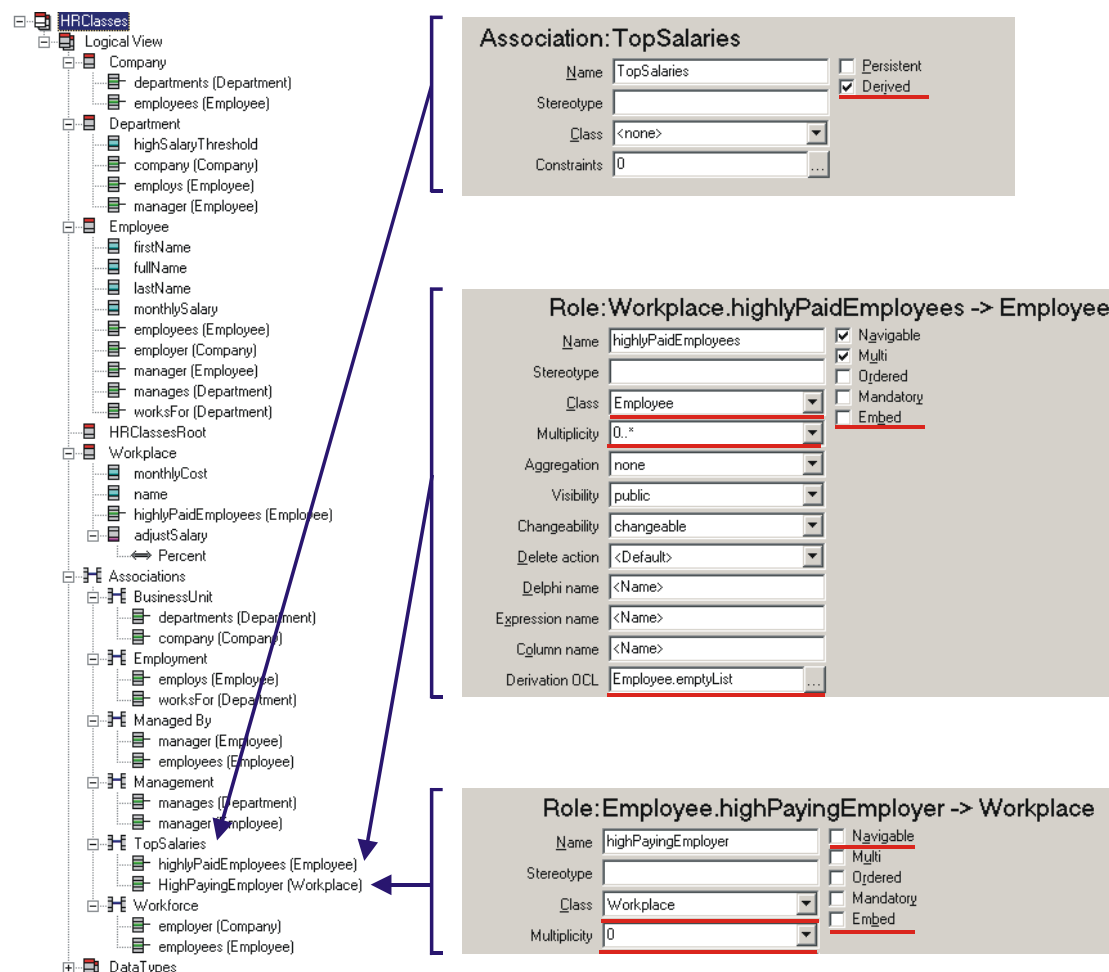


Figure 23: Implementing Top Salaries

The interesting items when implementing this association are:

- 1) The role *highlyPaidEmployees* uses the OCL expression `Employee.emptyList`. This expression results in an empty list of type `Employee` objects. The reason we use this expression is because the *Class* of the association is `Employee`. Even though `Workplace` is an abstract class, all descendant classes will inherit this association. It is important that it returns a valid result if called. If the OCL expression were blank Bold for Delphi would create a method stub to allow you to create your own result in code. Because we will be overriding the result using OCL in descendants the expression above was required.
- 2) The role *highPayingEmployer* is not required in a business sense, indeed deriving a result really adds no value to the model at all. Having the role here is important because assigning the *Class* property to `Workplace` gives us the context for the other role. The trick to removing this otherwise useless role from appearing in the `Employee` class and generated code is to mark it as not *Navigable*. This was done by un-checking the *Navigable* checkbox.

At the moment the association offers no value, as it will always return an empty list. By overriding the OCL expression in descendant classes we can generate a list of *Employees*.

Add the following attribute to the *Department* class:



Figure 24: Attribute – highSalaryThreshold

This attribute will be used to determine which employees in a department are considered as having a high salary. This means we need to now override the implementation of *highlyPaidEmployees* in both the *Company* and *Department* classes.

This is done by adding another entry to the *Derivation Expression* property of the two classes.

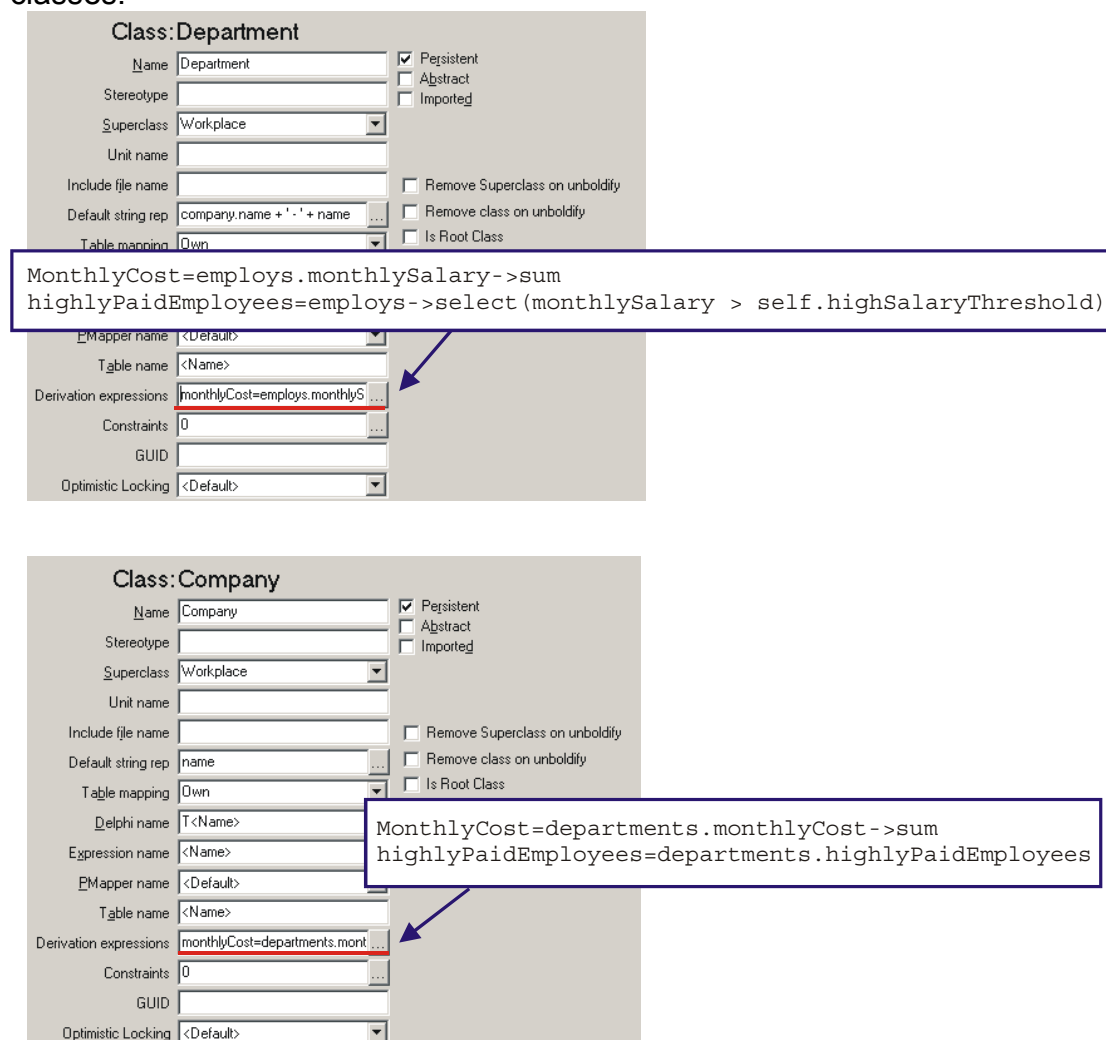


Figure 25: Derivation Expressions

The OCL expression used for the *Department* class uses the *highSalaryThreshold* attribute to determine which employees are considered as being highly paid. The *Company* class simply aggregates all the highly paid employees of its departments to provide a consolidated list.

The behavior can be investigated using the Bold for Delphi default object forms at runtime, however let's add a few more controls to give us a result on our main form. Figure 26 shows the enhanced main form at design time.

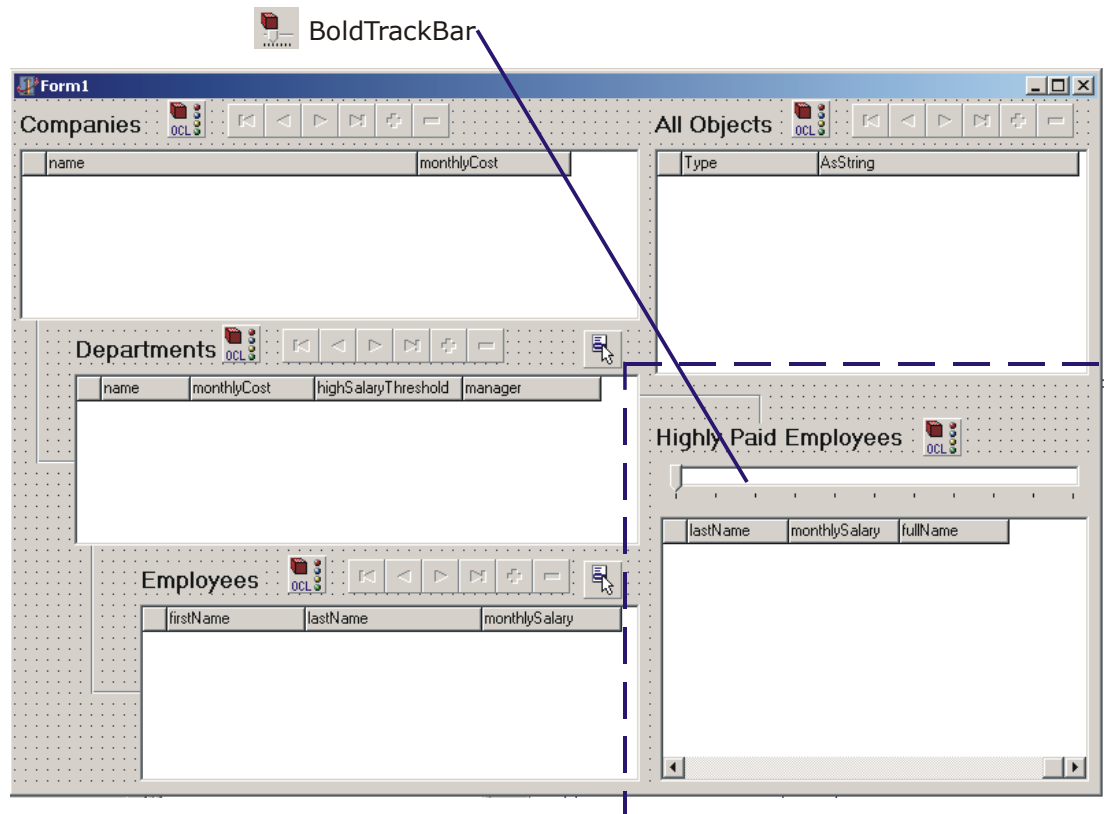


Figure 26: Enhancing the GUI

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

BoldListHandle

Name: HighlyPaidEmployeesList
 RootHandle: DepartmentList
 Expression: highlyPaidEmployees
 Enabled: True

BoldTrackBar

Name: SalaryThresholdTrackBar
 BoldHandle: DepartmentList
 BoldProperties.Expression: highSalaryThreshold
 BoldProperties.ApplyPolicy: bapChange
 Min: 0
 Max: 5000
 Frequency: 500
 Enabled: True

BoldGrid

Name: HighlyPaidEmployeesGrid

BoldHandle:

`HighlyPaidEmployeesList`

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

Run the application. Choose a department and add several employees with a spread of monthly salaries between 0 and 5000. Now slide the trackbar, Bold for Delphi dynamically responds with the correct result in the grid.

If you open the default editor for the *Company* and select the *highlyPaidEmployees* tab, you will see the highly paid employees for all departments. The beauty of Bold for Delphi is this list also dynamically updates as you move the slider. The synchronization methods of Bold for Delphi are extremely powerful and make creating loosely coupled forms easy.

Constraints: A parting note

An important aspect of Bold for Delphi is constraints. These are used to validate objects. Constraints can be specifically entered or implicit based on a relationship role. For example the role *Company* as part of the *BusinessUnit* association has a multiplicity of 1 specified. This means that every department must have a relationship with a company. However, as you can quickly check by running the application (prior to adding the cascade delete) that it is certainly possible to do so.

It would be too limiting or impractical for Bold for Delphi to try and enforce this constraint automatically as almost certainly the resulting behavior won't suit your particular scenario. Bold for Delphi however provides the ability to test the constraints, even as part of the delete verification process for your objects.

This is mentioned only because you may have been aware that certain aspects of the model were not being enforced in this way. However the Bold for Delphi constraint mechanism is powerful and fully assessable at runtime via code and Bold-aware components.

The subject of constraints will be addressed in a future article.

Summary

A lot of information has been presented in this article. The power of moving information about the application from the GUI and source into the model allows for a more precise definition of business rules. The centralized containment of business rule is a good way to ensure they are implemented in the application.

Designing an application with the knowledge everything modeled can actually be implemented, without worrying about technical details such as DB schema, linked lists, updating queries and moving data from the GUI into the logic layer allows focus to stay on the domain problem at hand. Keeping focus on the domain problems, rather than implementation problems ensures energy is spent on the right issues.

Bold for Delphi offers the most powerful assurance ever produced that your application is actually based on the model, and that the model will always accurately reflect the true implementation.

Appendix A: Source Code

Instructions

In this section you will find the source code for the project used in this article. All files can be created using a 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 be placed in the same directory, then simply load the DPR file into Delphi and compile.

HRManager.dpr

Start listing HRManager.dpr

```
program HRManager;  
  
uses  
  Forms,  
  MainForm in 'MainForm.pas' {Form1},  
  BusinessLayer in 'BusinessLayer.pas' {BusinessModule: TDataModule},  
  HRClasses in 'HRClasses.pas';  
  
{ $R *.RES }  
  
begin  
  Application.Initialize;  
  Application.CreateForm(TForm1, Form1);  
  Application.CreateForm(TBusinessModule, BusinessModule);  
  Application.Run;  
end.
```

End Listing HRManager.dpr

MainForm.pas

Start listing MainForm.pas

```
unit MainForm;

interface

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

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    CompanyGrid: TBoldGrid;
    CompanyNavigator: TBoldNavigator;
    DepartmentGrid: TBoldGrid;
    EmployeeGrid: TBoldGrid;
    DepartmentNavigator: TBoldNavigator;
    EmployeeNavigator: TBoldNavigator;
    CompanyList: TBoldListHandle;
    DepartmentList: TBoldListHandle;
    EmployeeList: TBoldListHandle;
    Bevel1: TBevel;
    Bevel2: TBevel;
    Bevel3: TBevel;
    Bevel4: TBevel;
    Label4: TLabel;
    ObjectList: TBoldListHandle;
    ObjectNavigator: TBoldNavigator;
    ObjectGrid: TBoldGrid;
    EmployeePopupMenu: TPopupMenu;
    AddNewEmployee1: TMenuItem;
    DeleteEmployee1: TMenuItem;
    MakeManager1: TMenuItem;
    SalaryPopupMenu: TPopupMenu;
    AdjustSalary1: TMenuItem;
    HighlyPaidEmployeesGrid: TBoldGrid;
    HighlyPaidEmployeesList: TBoldListHandle;
    SalaryThresholdTrackBar: TBoldTrackBar;
    Label5: TLabel;
    Bevel5: TBevel;
    Bevel6: TBevel;
    procedure EmployeeGridContextPopup(Sender: TObject; MousePos: TPoint;
      var Handled: Boolean);
    procedure MakeManager1Click(Sender: TObject);
    procedure AddNewEmployee1Click(Sender: TObject);
    procedure DeleteEmployee1Click(Sender: TObject);
    procedure AdjustSalary1Click(Sender: TObject);
    procedure DepartmentGridContextPopup(Sender: TObject; MousePos: TPoint;
      var Handled: Boolean);
    procedure CompanyGridContextPopup(Sender: TObject; MousePos: TPoint;
      var Handled: Boolean);
  private
    { Private declarations }
    FCurrentWorkplace: TWorkplace;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
  {$R *.DFM}

  procedure TForm1.EmployeeGridContextPopup(Sender: TObject;
    MousePos: TPoint; var Handled: Boolean);
  var
    EmployeeAssigned: Boolean;
  begin
    // Enable/Disable Menu items depending if an Employee is
    // selected.
```

Starting with Bold for Delphi/Bold for C++
Part 2: Extending Models

```
EmployeeAssigned := EmployeeGrid.CurrentBoldElement is TEmployee;
MakeManager1.Enabled := EmployeeAssigned;
DeleteEmployee1.Enabled := EmployeeAssigned;

// Enable/Disable Menu items depending if a department is selected.
AddNewEmployee1.Enabled := DepartmentGrid.CurrentBoldElement is TDepartment;
end;

procedure TForm1.MakeManager1Click(Sender: TObject);
var CurrentEmployee: TEmployee;
begin
// Check that the object is an Employee object and
// assigned it to our local variable
if EmployeeGrid.CurrentBoldElement is TEmployee then
begin
CurrentEmployee := TEmployee(EmployeeGrid.CurrentBoldElement);

// Check if the employee works for a department and set the
// employee as the manager
if assigned(CurrentEmployee.worksFor) then
CurrentEmployee.worksFor.manager := CurrentEmployee;
end;
end;

procedure TForm1.AddNewEmployee1Click(Sender: TObject);
var NewEmployee: TEmployee;
CurrentDepartment: TDepartment;
begin
// Because the Context Pop-up code has already checked
// that the current element is a TDepartment we can
// typecast to TDepartment safely
CurrentDepartment := TDepartment(DepartmentGrid.CurrentBoldElement);

// Create a new Employee object
NewEmployee := TEmployee.Create(CurrentDepartment.BoldSystem);
// Set the department of the new employee to the current
// department
NewEmployee.worksFor := CurrentDepartment;
end;

procedure TForm1.DeleteEmployee1Click(Sender: TObject);
var CurrentEmployee: TEmployee;
begin
// Check that the object is an Employee object and
// Delete it
if EmployeeGrid.CurrentBoldElement is TEmployee then
TEmployee(EmployeeGrid.CurrentBoldElement).Delete;
end;

procedure TForm1.AdjustSalary1Click(Sender: TObject);
var PercentStr: String;
PercentInt: Integer;
begin
// Ensure that the local form field is assigned
// an object
if assigned(FCurrentWorkplace) then
begin
// Display a pop-up requesting the percent to
// adjust the salaries by
PercentStr := '10';
if InputQuery('Salary Adjustment', 'Enter adjustment percentage', PercentStr) then
begin
try
// call the adjustSalaries method of
// with the supplied percent
PercentInt := StrToInt(PercentStr);
FCurrentWorkplace.adjustSalary(PercentInt);
except
on EConvertError do
MessageDlg('Invalid Percent entered!', mtError, [mbOK], 0);
end;
end;
end;
end;

procedure TForm1.DepartmentGridContextPopup(Sender: TObject;
MousePos: TPoint; var Handled: Boolean);
begin
// Initialize menu item and form private field
AdjustSalary1.Enabled := False;
```

```
    FCurrentWorkplace := nil;

    // Check that the object is a Workplace object and
    // assigned it to our local variable
    if DepartmentGrid.CurrentBoldElement is TWorkplace then
    begin
        AdjustSalary1.Enabled := True;
        FCurrentWorkplace := TWorkplace(DepartmentGrid.CurrentBoldElement);
    end;
end;

procedure TForm1.CompanyGridContextPopup(Sender: TObject; MousePos: TPoint;
    var Handled: Boolean);
begin
    // Initialize menu item and form private field
    AdjustSalary1.Enabled := False;
    FCurrentWorkplace := nil;

    // Check that the object is a Workplace object and
    // assigned it to our local variable
    if CompanyGrid.CurrentBoldElement is TWorkplace then
    begin
        AdjustSalary1.Enabled := True;
        FCurrentWorkplace := TWorkplace(CompanyGrid.CurrentBoldElement);
    end;
end;

end.
```

End Listing MainForm.pas

MainForm.dfm

Start listing MainForm.dfm

```
object Form1: TForm1
  Left = 421
  Top = 203
  Width = 799
  Height = 531
  Caption = 'Form1'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  PixelsPerInch = 96
  TextHeight = 13
  object Bevel3: TBevel
    Left = 56
    Top = 424
    Width = 41
    Height = 9
    Shape = bsBottomLine
  end
  object Bevel2: TBevel
    Left = 56
    Top = 320
    Width = 9
    Height = 113
    Shape = bsLeftLine
  end
  object Bevel4: TBevel
    Left = 16
    Top = 256
    Width = 41
    Height = 9
    Shape = bsBottomLine
  end
  object Bevel1: TBevel
    Left = 16
    Top = 160
    Width = 9
    Height = 105
    Shape = bsLeftLine
  end
  object Label1: TLabel
    Left = 4
    Top = 8
    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 Label2: TLabel
    Left = 44
    Top = 172
    Width = 105
    Height = 20
    Caption = 'Departments'
    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 = 92
    Top = 340
    Width = 88
    Height = 20
    Caption = 'Employees'
```

```
Font.Charset = DEFAULT_CHARSET
Font.Color = clWindowText
Font.Height = -16
Font.Name = 'MS Sans Serif'
Font.Style = [fsBold]
ParentFont = False
end
object Label4: TLabel
  Left = 468
  Top = 8
  Width = 87
  Height = 20
  Caption = 'All Objects'
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -16
  Font.Name = 'MS Sans Serif'
  Font.Style = [fsBold]
  ParentFont = False
end
object Bevel5: TBevel
  Left = 456
  Top = 208
  Width = 89
  Height = 9
  Shape = bsBottomLine
end
object Bevel6: TBevel
  Left = 544
  Top = 216
  Width = 9
  Height = 25
  Shape = bsLeftLine
end
object Label5: TLabel
  Left = 468
  Top = 236
  Width = 182
  Height = 20
  Caption = 'Highly Paid Employees'
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -16
  Font.Name = 'MS Sans Serif'
  Font.Style = [fsBold]
  ParentFont = False
end
object CompanyGrid: TBoldGrid
  Left = 4
  Top = 36
  Width = 453
  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 = 'monthlyCost'
      Font.Charset = DEFAULT_CHARSET
      Font.Color = clWindowText
      Font.Height = -11
      Font.Name = 'MS Sans Serif'
```

```
    Font.Style = []
end>
DefaultRowHeight = 17
EnableColAdjust = False
PopupMenu = SalaryPopupMenu
TabOrder = 0
TitleFont.Charset = DEFAULT_CHARSET
TitleFont.Color = clWindowText
TitleFont.Height = -11
TitleFont.Name = 'MS Sans Serif'
TitleFont.Style = []
OnContextPopup = CompanyGridContextPopup
ColWidths = (
    17
    270
    111)
end
object CompanyNavigator: TBoldNavigator
    Left = 156
    Top = 3
    Width = 162
    Height = 25
    BoldHandle = CompanyList
    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 "%1:s"?'
    UnlinkQuestion = 'Unlink "%1:s" from "%2:s"?'
    RemoveQuestion = 'Remove "%1:s" from the list?'
end
object DepartmentGrid: TBoldGrid
    Left = 44
    Top = 200
    Width = 413
    Height = 125
    AddNewAtEnd = False
    BoldAutoColumns = False
    BoldShowConstraints = False
    BoldHandle = DepartmentList
    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 = 'monthlyCost'
            Font.Charset = DEFAULT_CHARSET
            Font.Color = clWindowText
            Font.Height = -11
            Font.Name = 'MS Sans Serif'
            Font.Style = []
        end
        item
            BoldProperties.Expression = 'highSalaryThreshold'
            Font.Charset = DEFAULT_CHARSET
            Font.Color = clWindowText
            Font.Height = -11
            Font.Name = 'MS Sans Serif'
            Font.Style = []
        end
    end
end
```

```
        item
            BoldProperties.Expression = 'manager'
            Font.Charset = DEFAULT_CHARSET
            Font.Color = clWindowText
            Font.Height = -11
            Font.Name = 'MS Sans Serif'
            Font.Style = []
        end>
DefaultRowHeight = 17
EnableColAdjust = False
PopupMenu = SalaryPopupMenu
TabOrder = 2
TitleFont.Charset = DEFAULT_CHARSET
TitleFont.Color = clWindowText
TitleFont.Height = -11
TitleFont.Name = 'MS Sans Serif'
TitleFont.Style = []
OnContextPopup = DepartmentGridContextPopup
ColWidths = (
    17
    64
    90
    107
    100)
end
object EmployeeGrid: TBoldGrid
    Left = 92
    Top = 368
    Width = 365
    Height = 129
    AddNewAtEnd = False
    BoldAutoColumns = False
    BoldShowConstraints = False
    BoldHandle = EmployeeList
    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 = 'firstName'
            Font.Charset = DEFAULT_CHARSET
            Font.Color = clWindowText
            Font.Height = -11
            Font.Name = 'MS Sans Serif'
            Font.Style = []
        end
        item
            BoldProperties.Expression = 'lastName'
            Font.Charset = DEFAULT_CHARSET
            Font.Color = clWindowText
            Font.Height = -11
            Font.Name = 'MS Sans Serif'
            Font.Style = []
        end
        item
            BoldProperties.Expression = 'monthlySalary'
            Font.Charset = DEFAULT_CHARSET
            Font.Color = clWindowText
            Font.Height = -11
            Font.Name = 'MS Sans Serif'
            Font.Style = []
        end
    end>
DefaultRowHeight = 17
EnableColAdjust = False
PopupMenu = EmployeePopupMenu
TabOrder = 3
TitleFont.Charset = DEFAULT_CHARSET
TitleFont.Color = clWindowText
TitleFont.Height = -11
TitleFont.Name = 'MS Sans Serif'
TitleFont.Style = []
OnContextPopup = EmployeeGridContextPopup
ColWidths = (
    17
```

```
    100
    128
    101)
end
object DepartmentNavigator: TBoldNavigator
    Left = 196
    Top = 167
    Width = 162
    Height = 25
    BoldHandle = DepartmentList
    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 "%1:s"?'
    UnlinkQuestion = 'Unlink "%1:s" from "%2:s"?'
    RemoveQuestion = 'Remove "%1:s" from the list?'
end
object EmployeeNavigator: TBoldNavigator
    Left = 244
    Top = 335
    Width = 162
    Height = 25
    BoldHandle = EmployeeList
    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 "%1:s"?'
    UnlinkQuestion = 'Unlink "%1:s" from "%2:s"?'
    RemoveQuestion = 'Remove "%1:s" from the list?'
end
object ObjectNavigator: TBoldNavigator
    Left = 612
    Top = 3
    Width = 162
    Height = 25
    BoldHandle = ObjectList
    Flat = True
    TabOrder = 6
    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 "%1:s"?'
    UnlinkQuestion = 'Unlink "%1:s" from "%2:s"?'
    RemoveQuestion = 'Remove "%1:s" from the list?'
end
object ObjectGrid: TBoldGrid
    Left = 468
    Top = 36
    Width = 317
    Height = 165
    AddNewAtEnd = False
    BoldAutoColumns = False
    BoldShowConstraints = False
    BoldHandle = ObjectList
    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 = 'self.oclType'
        Font.Charset = DEFAULT_CHARSET
        Font.Color = clWindowText
        Font.Height = -11
        Font.Name = 'MS Sans Serif'
        Font.Style = []
        Title.Caption = 'Type'
    end
    item
        Font.Charset = DEFAULT_CHARSET
        Font.Color = clWindowText
        Font.Height = -11
        Font.Name = 'MS Sans Serif'
        Font.Style = []
        Title.Caption = 'AsString'
    end>
DefaultRowHeight = 17
EnableColAdjust = False
TabOrder = 7
TitleFont.Charset = DEFAULT_CHARSET
TitleFont.Color = clWindowText
TitleFont.Height = -11
TitleFont.Name = 'MS Sans Serif'
TitleFont.Style = []
ColWidths = (
    17
    99
    189)
end
object HighlyPaidEmployeesGrid: TBoldGrid
    Left = 472
    Top = 304
    Width = 313
    Height = 193
    AddNewAtEnd = False
    BoldAutoColumns = False
    BoldShowConstraints = False
    BoldHandle = HighlyPaidEmployeesList
    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 = 'firstName'
            Font.Charset = DEFAULT_CHARSET
            Font.Color = clWindowText
            Font.Height = -11
            Font.Name = 'MS Sans Serif'
            Font.Style = []
        end
        item
            BoldProperties.Expression = 'lastName'
            Font.Charset = DEFAULT_CHARSET
            Font.Color = clWindowText
            Font.Height = -11
            Font.Name = 'MS Sans Serif'
            Font.Style = []
        end
        item
            BoldProperties.Expression = 'monthlySalary'
            Font.Charset = DEFAULT_CHARSET
            Font.Color = clWindowText
            Font.Height = -11
            Font.Name = 'MS Sans Serif'
            Font.Style = []
        end
        item
            BoldProperties.Expression = 'fullName'
            Font.Charset = DEFAULT_CHARSET
            Font.Color = clWindowText
```

```
        Font.Height = -11
        Font.Name = 'MS Sans Serif'
        Font.Style = []
    end>
    DefaultRowHeight = 17
    EnableColAdjust = False
    TabOrder = 8
    TitleFont.Charset = DEFAULT_CHARSET
    TitleFont.Color = clWindowText
    TitleFont.Height = -11
    TitleFont.Name = 'MS Sans Serif'
    TitleFont.Style = []
    ColWidths = (
        17
        68
        73
        80
        79)
end
object SalaryThresholdTrackBar: TBoldTrackBar
    Left = 472
    Top = 264
    Width = 313
    Height = 37
    Max = 5000
    Orientation = trHorizontal
    Frequency = 500
    SelEnd = 0
    SelStart = 0
    TabOrder = 9
    TickMarks = tmBottomRight
    TickStyle = tsAuto
    BoldHandle = DepartmentList
    BoldProperties.Expression = 'highSalaryThreshold'
    BoldProperties.ApplyPolicy = bapChange
    ReadOnly = False
end
object CompanyList: TBoldListHandle
    RootHandle = BusinessModule.BoldSystemHandle1
    Expression = 'Company.allInstances'
    Left = 112
end
object DepartmentList: TBoldListHandle
    RootHandle = CompanyList
    Expression = 'departments'
    Left = 152
    Top = 164
end
object EmployeeList: TBoldListHandle
    RootHandle = DepartmentList
    Expression = 'employs'
    Left = 200
    Top = 332
end
object ObjectList: TBoldListHandle
    RootHandle = BusinessModule.BoldSystemHandle1
    Expression = 'HRClassesRoot.allInstances'
    Left = 568
end
object EmployeePopupMenu: TPopupMenu
    Left = 416
    Top = 336
    object MakeManager1: TMenuItem
        Caption = 'Make Manager'
        OnClick = MakeManager1Click
    end
    object AddNewEmployee1: TMenuItem
        Caption = 'Add New Employee'
        OnClick = AddNewEmployee1Click
    end
    object DeleteEmployee1: TMenuItem
        Caption = 'Delete Employee'
        OnClick = DeleteEmployee1Click
    end
end
object SalaryPopupMenu: TPopupMenu
    Left = 416
    Top = 168
    object AdjustSalary1: TMenuItem
        Caption = 'Adjust Salary'
```

```
        OnClick = AdjustSalary1Click
    end
end
object HighlyPaidEmployeesList: TBoldListHandle
    RootHandle = DepartmentList
    Expression = 'highlyPaidEmployees'
    Left = 664
    Top = 232
end
end
```

End Listing MainForm.dfm

BusinessLayer.pas

Start listing BusinessLayer.pas

```
unit BusinessLayer;

interface

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

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

var
  BusinessModule: TBusinessModule;

implementation

{$R *.DFM}

procedure TBusinessModule.DataModuleDestroy(Sender: TObject);
begin
  BoldSystemHandle1.System.UpdateDatabase;
end;

end.
```

End listing BusinessLayer.pas

BusinessLayer.dfm

Start listing BusinessLayer.dfm

```
object BusinessModule: TBusinessModule
  OldCreateOrder = False
  OnDestroy = DataModuleDestroy
  Left = 468
  Top = 228
  Height = 636
  Width = 662
object BoldModel1: TBoldModel
  UMLModelMode = ummNone
  Boldify.EnforceDefaultUMLCase = False
  Boldify.DefaultNavigableMultiplicity = '0..1'
  Boldify.DefaultNonNavigableMultiplicity = '0..*'
  Left = 68
  Top = 12
  Model = (
    'VERSION 19'
    ' (Model'
    #9' "HRClasses" '
    #9' "HRClassesRoot" '
    #9' "" '
    #9' "" '

    #9' "_Boldify.boldified=True,_BoldInternal.flattened=True,_BoldInte' +
    'rnal.ModelErrors=,Bold.DelphiName=<Name>,Bold.UnitName=HRClasses' +
    ',Bold.RootClass=HRClassesRoot" '
    #9' (Classes'
    #9#9' (Class'
    #9#9#9' "HRClassesRoot" '
    #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' "Workplace" '
    #9#9#9' "HRClassesRoot" '
    #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' "monthlyCost" '
    #9#9#9#9#9' "Currency" '
    #9#9#9#9#9' TRUE'
    #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' "derived=True,Bold.DerivationOCL=0" '
    #9#9#9#9' ) '
    #9#9#9' ) '
    #9#9#9' (Methods'
    #9#9#9#9' (Method'
    #9#9#9#9#9' "adjustSalary" '
    #9#9#9#9#9' "Percent: Integer" '
    #9#9#9#9#9' FALSE'
    #9#9#9#9#9' "" '
    #9#9#9#9#9' "" '

```



```
#9#9#9#9')'  
#9#9#9#9'(Attribute'  
#9#9#9#9'"monthlySalary"'  
#9#9#9#9'"Currency"'  
#9#9#9#9'FALSE'  
#9#9#9#9'""'  
#9#9#9#9'""'  
#9#9#9#9'2'  
#9#9#9#9'""'  
#9#9#9#9'"persistence=persistent"'  
#9#9#9#9')'  
#9#9#9#9'(Attribute'  
#9#9#9#9'"fullName"'  
#9#9#9#9'"String"'  
#9#9#9#9'TRUE'  
#9#9#9#9'""'  
#9#9#9#9'""'  
#9#9#9#9'2'  
#9#9#9#9'""'  
#9#9#9#9'"derived=True,Bold.ReverseDerive=True"'  
#9#9#9#9')'  
#9#9#9#9')'  
#9#9#9#9'(Methods'  
#9#9#9#9')'  
#9#9#9#9')'  
#9#9#9#9')'  
#9#9#9#9'(Associations'  
#9#9#9#9'(Association'  
#9#9#9#9'"Employment"'  
#9#9#9#9'"<NONE>"'  
#9#9#9#9'""'  
#9#9#9#9'""'  
#9#9#9#9'"persistence=persistent,Bold.DelphiName=<Name>"'  
#9#9#9#9'FALSE'  
#9#9#9#9'(Roles'  
#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'"Department"'  
#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,Bold.DeleteAction=Cascade"'  
#9#9#9#9#9'(Qualifiers'  
#9#9#9#9#9')'  
#9#9#9#9#9')'  
#9#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'"Employee"'  
#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#9'(Qualifiers'  
#9#9#9#9#9#9')'  
#9#9#9#9#9#9')'  
#9#9#9#9#9#9')'  
#9#9#9#9#9#9')'  
#9#9#9#9#9#9'(Association'  
#9#9#9#9#9#9'"Management"'  
#9#9#9#9#9#9'"<NONE>"'  
#9#9#9#9#9#9'""'  
#9#9#9#9#9#9'""'  
#9#9#9#9#9#9'"persistence=persistent,Bold.DelphiName=<Name>"'  
#9#9#9#9#9#9'FALSE'  
#9#9#9#9#9#9'(Roles'  
#9#9#9#9#9#9#9'(Role'  
#9#9#9#9#9#9#9'"manages"'  
#9#9#9#9#9#9#9'TRUE'  
#9#9#9#9#9#9#9'FALSE'  
#9#9#9#9#9#9#9'"Employee"'
```

```
#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' (Role'
#9#9#9#9#9' "manager"
#9#9#9#9#9' TRUE'
#9#9#9#9#9' FALSE'
#9#9#9#9#9' "Department"
#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#9' )'
#9#9#9' (Association'
#9#9#9' "BusinessUnit"
#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' "departments"
#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,Bold.DeleteAction=Cascade"
#9#9#9#9#9' (Qualifiers'
#9#9#9#9#9' )'
#9#9#9#9#9' )'
#9#9#9#9#9' (Role'
#9#9#9#9#9' "company"
#9#9#9#9#9' TRUE'
#9#9#9#9#9' FALSE'
#9#9#9#9#9' "Department"
#9#9#9#9#9' "'
#9#9#9#9#9' "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#9' )'
#9#9#9' (Association'
#9#9#9' "Managed By"
#9#9#9' "<NONE>"
#9#9#9' "'
#9#9#9' "'
#9#9#9' "persistence=persistent,derived=True,Bold.DelphiName=<Name>"
#9#9#9' TRUE'
#9#9#9' (Roles'
#9#9#9#9' (Role'
#9#9#9#9#9' "manager"
#9#9#9#9#9' TRUE'
#9#9#9#9#9' FALSE'
#9#9#9#9#9' "Employee"
```

```
#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' "Bold.DerivationOCL=worksFor.manager" '
#9#9#9#9#9' (Qualifiers '
#9#9#9#9#9' ) '
#9#9#9#9' (Role '
#9#9#9#9#9' "employees" '
#9#9#9#9#9' TRUE '
#9#9#9#9#9' FALSE '
#9#9#9#9#9' "Employee" '
#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,Bold.DerivationOCL=manages.employs" '
#9#9#9#9#9' (Qualifiers '
#9#9#9#9#9' ) '
#9#9#9#9' (Association '
#9#9#9' "Workforce" '
#9#9#9' "<NONE>" '
#9#9#9' "" '
#9#9#9' "" '
#9#9#9' "persistence=persistent,derived=True,Bold.DelphiName=<Name>" '
#9#9#9' TRUE '
#9#9#9' (Roles '
#9#9#9#9' (Role '
#9#9#9#9#9' "employer" '
#9#9#9#9#9' TRUE '
#9#9#9#9#9' FALSE '
#9#9#9#9#9' "Employee" '
#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' "Bold.DerivationOCL=worksFor.company" '
#9#9#9#9#9' (Qualifiers '
#9#9#9#9#9' ) '
#9#9#9#9' (Role '
#9#9#9#9#9' "employees" '
#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,Bold.DerivationOCL=departments.employs" '
#9#9#9#9#9' (Qualifiers '
#9#9#9#9#9' ) '
#9#9#9#9' (Association '
#9#9#9' "TopSalaries" '
#9#9#9' "<NONE>" '
#9#9#9' "" '
#9#9#9' "" '
#9#9#9' "persistence=persistent,derived=True,Bold.DelphiName=<Name>" '
#9#9#9' TRUE '
#9#9#9' (Roles '
#9#9#9#9' (Role '
#9#9#9#9#9' "highlyPaidEmployees" '
#9#9#9#9#9' TRUE '
#9#9#9#9#9' FALSE '
#9#9#9#9#9' "Workplace" '

```


HRClasses.pas

Start listing HRClasses.pas

```
(*****  
(*      This file is autogenerated      *)  
(*  Any manual changes will be LOST!  *)  
(*****  
(* Generated 2/08/2002 8:14:52 PM      *)  
(*****  
(* This file should be stored in the   *)  
(* same directory as the form/datamodule *)  
(* with the corresponding model       *)  
(*****  
(* Copyright notice:                   *)  
(*                                     *)  
(*****  
  
unit HRClasses;  
  
{ $DEFINE HRClasses_unitheader }  
{ $INCLUDE HRClasses_Interface.inc }  
  
{ Includefile for methodimplementations }  
  
{ $INCLUDE HRClasses.inc }  
  
const  
  BoldMemberAssertInvalidObjectType: string = 'Object of singlelink (%s.%s) is of wrong type  
(is %s, should be %s)';  
  
{ THRClassesRoot }  
  
procedure THRClassesRootList.Add(NewObject: THRClassesRoot);  
begin  
  if Assigned(NewObject) then  
    AddElement(NewObject);  
end;  
  
function THRClassesRootList.IndexOf(anObject: THRClassesRoot): Integer;  
begin  
  result := IndexOfElement(anObject);  
end;  
  
function THRClassesRootList.Includes(anObject: THRClassesRoot) : Boolean;  
begin  
  result := IncludesElement(anObject);  
end;  
  
function THRClassesRootList.AddNew: THRClassesRoot;  
begin  
  result := THRClassesRoot(InternalAddNew);  
end;  
  
procedure THRClassesRootList.Insert(index: Integer; NewObject: THRClassesRoot);  
begin  
  if assigned(NewObject) then  
    InsertElement(index, NewObject);  
end;  
  
function THRClassesRootList.GetBoldObject(index: Integer): THRClassesRoot;  
begin  
  result := THRClassesRoot(GetElement(index));  
end;  
  
procedure THRClassesRootList.SetBoldObject(index: Integer; NewObject: THRClassesRoot);  
begin;  
  SetElement(index, NewObject);  
end;  
  
{ TEmployee }  
  
function TEmployee._Get_M_firstName: TBAStrng;  
begin  
  assert(ValidateMember('TEmployee', 'firstName', 0, TBAStrng));  
  Result := TBAStrng(BoldMembers[0]);  
end;  
  
function TEmployee._GetfirstName: String;
```

```
begin
    Result := M_firstName.AsString;
end;

procedure TEmployee._SetfirstName(NewValue: String);
begin
    M_firstName.AsString := NewValue;
end;

function TEmployee._Get_M_lastName: TBAStrng;
begin
    assert(ValidateMember('TEmployee', 'lastName', 1, TBAStrng));
    Result := TBAStrng(BoldMembers[1]);
end;

function TEmployee._GetlastName: String;
begin
    Result := M_lastName.AsString;
end;

procedure TEmployee._SetlastName(NewValue: String);
begin
    M_lastName.AsString := NewValue;
end;

function TEmployee._Get_M_monthlySalary: TBACurrency;
begin
    assert(ValidateMember('TEmployee', 'monthlySalary', 2, TBACurrency));
    Result := TBACurrency(BoldMembers[2]);
end;

function TEmployee._GetmonthlySalary: Currency;
begin
    Result := M_monthlySalary.AsCurrency;
end;

procedure TEmployee._SetmonthlySalary(NewValue: Currency);
begin
    M_monthlySalary.AsCurrency := NewValue;
end;

function TEmployee._Get_M_fullName: TBAStrng;
begin
    assert(ValidateMember('TEmployee', 'fullName', 3, TBAStrng));
    Result := TBAStrng(BoldMembers[3]);
end;

function TEmployee._GetfullName: String;
begin
    Result := M_fullName.AsString;
end;

procedure TEmployee._SetfullName(NewValue: String);
begin
    M_fullName.AsString := NewValue;
end;

function TEmployee._Get_M_worksFor: TBoldObjectReference;
begin
    assert(ValidateMember('TEmployee', 'worksFor', 4, TBoldObjectReference));
    Result := TBoldObjectReference(BoldMembers[4]);
end;

function TEmployee._GetworksFor: TDepartment;
begin
    assert(not assigned(M_worksFor.BoldObject) or (M_worksFor.BoldObject is TDepartment),
    SysUtils.Format(BoldMemberAssertInvalidObjectType, [ClassName, 'worksFor',
    M_worksFor.BoldObject.ClassName, 'TDepartment']));
    Result := TDepartment(M_worksFor.BoldObject);
end;

procedure TEmployee._SetworksFor(value: TDepartment);
begin
    M_worksFor.BoldObject := value;
end;

function TEmployee._Getmanages: TDepartmentList;
begin
    assert(ValidateMember('TEmployee', 'manages', 5, TDepartmentList));
    Result := TDepartmentList(BoldMembers[5]);
```

```
end;

function TEmployee._Get_M_manager: TBoldObjectReference;
begin
  assert(ValidateMember('TEmployee', 'manager', 6, TBoldObjectReference));
  Result := TBoldObjectReference(BoldMembers[6]);
end;

function TEmployee._Getmanager: TEmployee;
begin
  assert(not assigned(M_manager.BoldObject) or (M_manager.BoldObject is TEmployee),
  SysUtils.format(BoldMemberAssertInvalidObjectType, [ClassName, 'manager',
  M_manager.BoldObject.ClassName, 'TEmployee']));
  Result := TEmployee(M_manager.BoldObject);
end;

procedure TEmployee._Setmanager(value: TEmployee);
begin
  M_manager.BoldObject := value;
end;

function TEmployee._Getemployees: TEmployeeList;
begin
  assert(ValidateMember('TEmployee', 'employees', 7, TEmployeeList));
  Result := TEmployeeList(BoldMembers[7]);
end;

function TEmployee._Get_M_employer: TBoldObjectReference;
begin
  assert(ValidateMember('TEmployee', 'employer', 8, TBoldObjectReference));
  Result := TBoldObjectReference(BoldMembers[8]);
end;

function TEmployee._Getemployer: TCompany;
begin
  assert(not assigned(M_employer.BoldObject) or (M_employer.BoldObject is TCompany),
  SysUtils.format(BoldMemberAssertInvalidObjectType, [ClassName, 'employer',
  M_employer.BoldObject.ClassName, 'TCompany']));
  Result := TCompany(M_employer.BoldObject);
end;

procedure TEmployee._Setemployer(value: TCompany);
begin
  M_employer.BoldObject := value;
end;

procedure TEmployeeList.Add(NewObject: TEmployee);
begin
  if Assigned(NewObject) then
    AddElement(NewObject);
end;

function TEmployeeList.IndexOf(anObject: TEmployee): Integer;
begin
  result := IndexOfElement(anObject);
end;

function TEmployeeList.Includes(anObject: TEmployee) : Boolean;
begin
  result := IncludesElement(anObject);
end;

function TEmployeeList.AddNew: TEmployee;
begin
  result := TEmployee(InternalAddNew);
end;

procedure TEmployeeList.Insert(index: Integer; NewObject: TEmployee);
begin
  if assigned(NewObject) then
    InsertElement(index, NewObject);
end;

function TEmployeeList.GetBoldObject(index: Integer): TEmployee;
begin
  result := TEmployee(GetElement(index));
end;

procedure TEmployeeList.SetBoldObject(index: Integer; NewObject: TEmployee);
begin;
```

```
    SetElement(index, NewObject);
end;

function TEmployee.GetDeriveMethodForMember(Member: TBoldMember): TBoldDeriveAndResubscribe;
begin
    if (Member = M_fullName) then result := _fullName_DeriveAndSubscribe else
        result := inherited GetDeriveMethodForMember(Member);
end;

function TEmployee.GetReverseDeriveMethodForMember(Member: TBoldMember): TBoldReverseDerive;
begin
    result := inherited GetReverseDeriveMethodForMember(Member);
    if not assigned(result) and (Member = M_fullName) then result := _fullName_ReverseDerive;
end;

{ TWorkplace }

function TWorkplace._Get_M_name: TBAStrng;
begin
    assert(ValidateMember('TWorkplace', 'name', 0, TBAStrng));
    Result := TBAStrng(BoldMembers[0]);
end;

function TWorkplace._Getname: String;
begin
    Result := M_name.AsString;
end;

procedure TWorkplace._SetName(NewValue: String);
begin
    M_name.AsString := NewValue;
end;

function TWorkplace._Get_M_monthlyCost: TBACurrency;
begin
    assert(ValidateMember('TWorkplace', 'monthlyCost', 1, TBACurrency));
    Result := TBACurrency(BoldMembers[1]);
end;

function TWorkplace._GetmonthlyCost: Currency;
begin
    Result := M_monthlyCost.AsCurrency;
end;

function TWorkplace._GethighlyPaidEmployees: TEmployeeList;
begin
    assert(ValidateMember('TWorkplace', 'highlyPaidEmployees', 2, TEmployeeList));
    Result := TEmployeeList(BoldMembers[2]);
end;

procedure TWorkplaceList.Add(NewObject: TWorkplace);
begin
    if Assigned(NewObject) then
        AddElement(NewObject);
end;

function TWorkplaceList.IndexOf(anObject: TWorkplace): Integer;
begin
    result := IndexOfElement(anObject);
end;

function TWorkplaceList.Includes(anObject: TWorkplace) : Boolean;
begin
    result := IncludesElement(anObject);
end;

function TWorkplaceList.AddNew: TWorkplace;
begin
    result := TWorkplace(InternalAddNew);
end;

procedure TWorkplaceList.Insert(index: Integer; NewObject: TWorkplace);
begin
    if assigned(NewObject) then
        InsertElement(index, NewObject);
end;

function TWorkplaceList.GetBoldObject(index: Integer): TWorkplace;
begin
    result := TWorkplace(GetElement(index));
end;
```

```
end;

procedure TWorkplaceList.SetBoldObject(index: Integer; NewObject: TWorkplace);
begin
  SetElement(index, NewObject);
end;

{ TCompany }

function TCompany._Getdepartments: TDepartmentList;
begin
  assert(ValidateMember('TCompany', 'departments', 3, TDepartmentList));
  Result := TDepartmentList(BoldMembers[3]);
end;

function TCompany._Getemployees: TEmployeeList;
begin
  assert(ValidateMember('TCompany', 'employees', 4, TEmployeeList));
  Result := TEmployeeList(BoldMembers[4]);
end;

procedure TCompanyList.Add(NewObject: TCompany);
begin
  if Assigned(NewObject) then
    AddElement(NewObject);
end;

function TCompanyList.IndexOf(anObject: TCompany): Integer;
begin
  result := IndexOfElement(anObject);
end;

function TCompanyList.Includes(anObject: TCompany) : Boolean;
begin
  result := IncludesElement(anObject);
end;

function TCompanyList.AddNew: TCompany;
begin
  result := TCompany(InternalAddNew);
end;

procedure TCompanyList.Insert(index: Integer; NewObject: TCompany);
begin
  if assigned(NewObject) then
    InsertElement(index, NewObject);
end;

function TCompanyList.GetBoldObject(index: Integer): TCompany;
begin
  result := TCompany(GetElement(index));
end;

procedure TCompanyList.SetBoldObject(index: Integer; NewObject: TCompany);
begin
  SetElement(index, NewObject);
end;

{ TDepartment }

function TDepartment._Get_M_highSalaryThreshold: TBACurrency;
begin
  assert(ValidateMember('TDepartment', 'highSalaryThreshold', 3, TBACurrency));
  Result := TBACurrency(BoldMembers[3]);
end;

function TDepartment._GethighSalaryThreshold: Currency;
begin
  Result := M_highSalaryThreshold.AsCurrency;
end;

procedure TDepartment._SethighSalaryThreshold(NewValue: Currency);
begin
  M_highSalaryThreshold.AsCurrency := NewValue;
end;

function TDepartment._Getemploys: TEmployeeList;
begin
  assert(ValidateMember('TDepartment', 'employs', 4, TEmployeeList));
  Result := TEmployeeList(BoldMembers[4]);
```

```
end;

function TDepartment._Get_M_manager: TBoldObjectReference;
begin
  assert(ValidateMember('TDepartment', 'manager', 5, TBoldObjectReference));
  Result := TBoldObjectReference(BoldMembers[5]);
end;

function TDepartment._Getmanager: TEmployee;
begin
  assert(not assigned(M_manager.BoldObject) or (M_manager.BoldObject is TEmployee),
  SysUtils.format(BoldMemberAssertInvalidObjectType, [ClassName, 'manager',
  M_manager.BoldObject.ClassName, 'TEmployee']));
  Result := TEmployee(M_manager.BoldObject);
end;

procedure TDepartment._Setmanager(value: TEmployee);
begin
  M_manager.BoldObject := value;
end;

function TDepartment._Get_M_company: TBoldObjectReference;
begin
  assert(ValidateMember('TDepartment', 'company', 6, TBoldObjectReference));
  Result := TBoldObjectReference(BoldMembers[6]);
end;

function TDepartment._Getcompany: TCompany;
begin
  assert(not assigned(M_company.BoldObject) or (M_company.BoldObject is TCompany),
  SysUtils.format(BoldMemberAssertInvalidObjectType, [ClassName, 'company',
  M_company.BoldObject.ClassName, 'TCompany']));
  Result := TCompany(M_company.BoldObject);
end;

procedure TDepartment._Setcompany(value: TCompany);
begin
  M_company.BoldObject := value;
end;

procedure TDepartmentList.Add(NewObject: TDepartment);
begin
  if Assigned(NewObject) then
    AddElement(NewObject);
end;

function TDepartmentList.IndexOf(anObject: TDepartment): Integer;
begin
  result := IndexOfElement(anObject);
end;

function TDepartmentList.Includes(anObject: TDepartment) : Boolean;
begin
  result := IncludesElement(anObject);
end;

function TDepartmentList.AddNew: TDepartment;
begin
  result := TDepartment(InternalAddNew);
end;

procedure TDepartmentList.Insert(index: Integer; NewObject: TDepartment);
begin
  if assigned(NewObject) then
    InsertElement(index, NewObject);
end;

function TDepartmentList.GetBoldObject(index: Integer): TDepartment;
begin
  result := TDepartment(GetElement(index));
end;

procedure TDepartmentList.SetBoldObject(index: Integer; NewObject: TDepartment);
begin
  SetElement(index, NewObject);
end;

function GeneratedCodeCRC: String;
begin
  result := '180244953';
end;
```

```
end;

procedure InstallObjectListClasses(BoldObjectListClasses: TBoldGeneratedClassList);
begin
  BoldObjectListClasses.AddObjectEntry('HRClassesRoot', THRClassesRootList);
  BoldObjectListClasses.AddObjectEntry('Employee', TEmployeeList);
  BoldObjectListClasses.AddObjectEntry('Workplace', TWorkplaceList);
  BoldObjectListClasses.AddObjectEntry('Company', TCompanyList);
  BoldObjectListClasses.AddObjectEntry('Department', TDepartmentList);
end;

procedure InstallBusinessClasses(BoldObjectClasses: TBoldGeneratedClassList);
begin
  BoldObjectClasses.AddObjectEntry('HRClassesRoot', THRClassesRoot);
  BoldObjectClasses.AddObjectEntry('Employee', TEmployee);
  BoldObjectClasses.AddObjectEntry('Workplace', TWorkplace);
  BoldObjectClasses.AddObjectEntry('Company', TCompany);
  BoldObjectClasses.AddObjectEntry('Department', TDepartment);
end;

var
  CodeDescriptor: TBoldGeneratedCodeDescriptor;

initialization
  CodeDescriptor := GeneratedCodes.AddGeneratedCodeDescriptorWithFunc('HRClasses',
  InstallBusinessClasses, InstallObjectListClasses, GeneratedCodeCRC);
finalization
  GeneratedCodes.Remove(CodeDescriptor);
end.
```

End listing HRClasses.pas

HRClasses.inc

Start listing HRClasses.inc

```
{*****}
{
  Bold for Delphi Stub File
}
{
  Autogenerated file for method implementations
}
{*****}

//
{$INCLUDE HRClasses_Interface.inc}

procedure TEmployee._fullName_DeriveAndSubscribe(DerivedObject: TObject; Subscriber:
TBoldSubscriber);
begin
  // Set the fullname
  M_FullName.AsString := firstName + ' ' + lastName;
  // subscribe to notifications of either the first
  // or last name changing
  M_FirstName.DefaultSubscribe(subscriber);
  M_LastName.DefaultSubscribe(subscriber);
end;

procedure TEmployee._fullName_ReverseDerive(DerivedObject: TObject);
var aFullName: String;
    p: integer;
begin
  // strip away leading and trailing spaces
  aFullName := trim(fullName);
  p := pos( ' ', aFullName );
  // Check if a space was found
  if p <> 0 then
    begin
      // the first name is everything up to the first space
      // the last name is the rest
      firstName := copy( aFullName, 1, p-1 );
      lastName := trim(copy(aFullName, p+1, maxint ));
    end else
    begin
      // No space found, the first name is everything,
      // the last name is set blank
      firstName := aFullName;
      lastName := '';
    end;
end;

procedure TWorkplace.adjustSalary(Percent: Integer);
begin
  // abstract class, implementation not required
end;

procedure TCompany.adjustSalary(Percent: Integer);
var counter: Integer;
begin
  inherited;
  // Loop thru all departments and call
  // adjustSalary for each one
  for Counter := 0 to departments.Count -1 do
    departments[counter].adjustSalary(Percent);
end;

procedure TDepartment.adjustSalary(Percent: Integer);
var counter: Integer;
    SalaryChange: Currency;
begin
  inherited;
  // Loop thru all employee's and adjust their
  // salary
  for Counter := 0 to employs.Count -1 do
    begin
      SalaryChange := employs[counter].monthlySalary * (Percent / 100);
      employs[counter].monthlySalary := employs[counter].monthlySalary + SalaryChange;
    end;
end;
end;
```

End listing HRClasses.inc

HRClasses_Interface.inc

Start listing HRClasses_Interface.inc

```
(*****)  
(*      This file is autogenerated      *)  
(*  Any manual changes will be LOST!  *)  
(*****)  
(* Generated 2/08/2002 8:14:53 PM      *)  
(*****)  
(* This file should be stored in the    *)  
(* same directory as the form/datamodule *)  
(* with the corresponding model        *)  
(*****)  
(* Copyright notice:                   *)  
(*                                     *)  
(*****)  
  
{ $IFDEF HRClasses_Interface.inc }  
{ $DEFINE HRClasses_Interface.inc }  
  
{ $IFDEF HRClasses_unitheader }  
unit HRClasses;  
{ $ENDIF }  
  
interface  
  
uses  
  // interface uses  
  // interface dependencies  
  // attribute classes  
  BoldAttributes,  
  // other  
  Classes,  
  Controls, // for TDate  
  SysUtils,  
  BoldDefs,  
  BoldSubscription,  
  BoldDeriver,  
  BoldElements,  
  BoldDomainElement,  
  BoldSystemRT,  
  BoldSystem;  
  
type  
  { forward declarations of all classes }  
  
  THRClassesRoot = class;  
  THRClassesRootList = class;  
  TEmployee = class;  
  TEmployeeList = class;  
  TWorkplace = class;  
  TWorkplaceList = class;  
  TCompany = class;  
  TCompanyList = class;  
  TDepartment = class;  
  TDepartmentList = class;  
  
  THRClassesRoot = class(TBoldObject)  
  private  
  protected  
  public  
  end;
```

```
TEmployee = class(THRClassesRoot)
private
    function _Get_M_firstName: TBAStrng;
    function _GetfirstName: String;
    procedure _SetfirstName(NewValue: String);
    function _Get_M_lastName: TBAStrng;
    function _GetlastName: String;
    procedure _SetlastName(NewValue: String);
    function _Get_M_monthlySalary: TBACurrency;
    function _GetmonthlySalary: Currency;
    procedure _SetmonthlySalary(NewValue: Currency);
    function _Get_M_fullName: TBAStrng;
    function _GetfullName: String;
    procedure _SetfullName(NewValue: String);
    function _GetworksFor: TDepartment;
    function _Get_M_worksFor: TBoldObjectReference;
    procedure _SetworksFor(value: TDepartment);
    function _Getmanages: TDepartmentList;
    function _Getmanager: TEmployee;
    function _Get_M_manager: TBoldObjectReference;
    procedure _Setmanager(value: TEmployee);
    function _Getemployees: TEmployeeList;
    function _Getemployer: TCompany;
    function _Get_M_employer: TBoldObjectReference;
    procedure _Setemployer(value: TCompany);
protected
    procedure _fullName_DeriveAndSubscribe(DerivedObject: TObject;
Subscriber: TBoldSubscriber); virtual;
    procedure _fullName_ReverseDerive(DerivedObject: TObject); virtual;
    function GetDeriveMethodForMember(Member: TBoldMember):
TBoldDeriveAndResubscribe; override;
    function GetReverseDeriveMethodForMember(Member: TBoldMember):
TBoldReverseDerive; override;
public
    property M_firstName: TBAStrng read _Get_M_firstName;
    property M_lastName: TBAStrng read _Get_M_lastName;
    property M_monthlySalary: TBACurrency read _Get_M_monthlySalary;
    property M_fullName: TBAStrng read _Get_M_fullName;
    property M_worksFor: TBoldObjectReference read _Get_M_worksFor;
    property M_manages: TDepartmentList read _Getmanages;
    property M_manager: TBoldObjectReference read _Get_M_manager;
    property M_employees: TEmployeeList read _Getemployees;
    property M_employer: TBoldObjectReference read _Get_M_employer;
    property firstName: String read _GetfirstName write _SetfirstName;
    property lastName: String read _GetlastName write _SetlastName;
    property monthlySalary: Currency read _GetmonthlySalary write
_SetmonthlySalary;
    property fullName: String read _GetfullName write _SetfullName;
    property worksFor: TDepartment read _GetworksFor write _SetworksFor;
    property manages: TDepartmentList read _Getmanages;
    property manager: TEmployee read _Getmanager write _Setmanager;
    property employees: TEmployeeList read _Getemployees;
    property employer: TCompany read _Getemployer write _Setemployer;
end;

TWorkplace = class(THRClassesRoot)
private
    function _Get_M_name: TBAStrng;
    function _Getname: String;
    procedure _Setname(NewValue: String);
    function _Get_M_monthlyCost: TBACurrency;
    function _GetmonthlyCost: Currency;
    function _GethighlyPaidEmployees: TEmployeeList;
protected
```

```
public
    procedure adjustSalary(Percent: Integer); virtual;
    property M_name: TBAStrng read _Get_M_name;
    property M_monthlyCost: TBACurrency read _Get_M_monthlyCost;
    property M_highlyPaidEmployees: TEmployeeList read
_GethighlyPaidEmployees;
    property name: String read _Getname write _Setname;
    property monthlyCost: Currency read _GetmonthlyCost;
    property highlyPaidEmployees: TEmployeeList read
_GethighlyPaidEmployees;
end;

TCompany = class(TWorkplace)
private
    function _Getdepartments: TDepartmentList;
    function _Getemployees: TEmployeeList;
protected
public
    procedure adjustSalary(Percent: Integer); override;
    property M_departments: TDepartmentList read _Getdepartments;
    property M_employees: TEmployeeList read _Getemployees;
    property departments: TDepartmentList read _Getdepartments;
    property employees: TEmployeeList read _Getemployees;
end;

TDepartment = class(TWorkplace)
private
    function _Get_M_highSalaryThreshold: TBACurrency;
    function _GethighSalaryThreshold: Currency;
    procedure _SethighSalaryThreshold(NewValue: Currency);
    function _Getemploys: TEmployeeList;
    function _Getmanager: TEmployee;
    function _Get_M_manager: TBoldObjectReference;
    procedure _Setmanager(value: TEmployee);
    function _Getcompany: TCompany;
    function _Get_M_company: TBoldObjectReference;
    procedure _Setcompany(value: TCompany);
protected
public
    procedure adjustSalary(Percent: Integer); override;
    property M_highSalaryThreshold: TBACurrency read
_Get_M_highSalaryThreshold;
    property M_employs: TEmployeeList read _Getemploys;
    property M_manager: TBoldObjectReference read _Get_M_manager;
    property M_company: TBoldObjectReference read _Get_M_company;
    property highSalaryThreshold: Currency read _GethighSalaryThreshold
write _SethighSalaryThreshold;
    property employs: TEmployeeList read _Getemploys;
    property manager: TEmployee read _Getmanager write _Setmanager;
    property company: TCompany read _Getcompany write _Setcompany;
end;

THRClassesRootList = class(TBoldObjectList)
protected
    function GetBoldObject(index: Integer): THRClassesRoot;
    procedure SetBoldObject(index: Integer; NewObject: THRClassesRoot);
public
    function Includes(anObject: THRClassesRoot): Boolean;
    function IndexOf(anObject: THRClassesRoot): Integer;
    procedure Add(NewObject: THRClassesRoot);
    function AddNew: THRClassesRoot;
    procedure Insert(index: Integer; NewObject: THRClassesRoot);
    property BoldObjects[index: Integer]: THRClassesRoot read GetBoldObject
write SetBoldObject; default;
```

```
end;

TEmployeeList = class(THRClassesRootList)
protected
    function GetBoldObject(index: Integer): TEmployee;
    procedure SetBoldObject(index: Integer; NewObject: TEmployee);
public
    function Includes(anObject: TEmployee): Boolean;
    function IndexOf(anObject: TEmployee): Integer;
    procedure Add(NewObject: TEmployee);
    function AddNew: TEmployee;
    procedure Insert(index: Integer; NewObject: TEmployee);
    property BoldObjects[index: Integer]: TEmployee read GetBoldObject
write SetBoldObject; default;
end;

TWorkplaceList = class(THRClassesRootList)
protected
    function GetBoldObject(index: Integer): TWorkplace;
    procedure SetBoldObject(index: Integer; NewObject: TWorkplace);
public
    function Includes(anObject: TWorkplace): Boolean;
    function IndexOf(anObject: TWorkplace): Integer;
    procedure Add(NewObject: TWorkplace);
    function AddNew: TWorkplace;
    procedure Insert(index: Integer; NewObject: TWorkplace);
    property BoldObjects[index: Integer]: TWorkplace read GetBoldObject
write SetBoldObject; default;
end;

TCompanyList = class(TWorkplaceList)
protected
    function GetBoldObject(index: Integer): TCompany;
    procedure SetBoldObject(index: Integer; NewObject: TCompany);
public
    function Includes(anObject: TCompany): Boolean;
    function IndexOf(anObject: TCompany): Integer;
    procedure Add(NewObject: TCompany);
    function AddNew: TCompany;
    procedure Insert(index: Integer; NewObject: TCompany);
    property BoldObjects[index: Integer]: TCompany read GetBoldObject write
SetBoldObject; default;
end;

TDepartmentList = class(TWorkplaceList)
protected
    function GetBoldObject(index: Integer): TDepartment;
    procedure SetBoldObject(index: Integer; NewObject: TDepartment);
public
    function Includes(anObject: TDepartment): Boolean;
    function IndexOf(anObject: TDepartment): Integer;
    procedure Add(NewObject: TDepartment);
    function AddNew: TDepartment;
    procedure Insert(index: Integer; NewObject: TDepartment);
    property BoldObjects[index: Integer]: TDepartment read GetBoldObject
write SetBoldObject; default;
end;

function GeneratedCodeCRC: String;

implementation

uses
    // implementation uses
```

```
// implementation dependencies  
// other  
BoldGeneratedCodeDictionary;  
  
{ $ENDIF }
```

End listing HRClasses_Interface.inc