Starting with Bold for Delphi / Bold for C++

Part 3: Object Constraint Language (OCL)

Anthony Richardson - anthony@viewpointsa.com
11 August 2002, Revision 1.0

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

Getting Started

Introduction

'Starting with Bold for Delphi' is a series of articles designed 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***

Introduction to the Object Constraint Language (OCL)

Overview

The Object Constraint Language (OCL) is a formal language used to express constraints and navigation within Unified Modeling Language (UML) models. Developed in 1997 by IBM it is now managed by the Object Modeling Group (OMG) and is included as part of the UML specification.

OCL is required because a class model, by itself, is incomplete as means of specifying requirements of the classes at runtime. In the past the extra definition applied to UML diagrams was in the format of natural language. However, as this is open to ambiguous interpretation and not executable as code, it failed to address it primary requirement of simplifying and clarifying the class model.

OCL provides a simple language that can be used to express boundaries for class instances at runtime (constraints) as well as provide a clear means to navigate the model.

OCL and Constraints

OCL is used in the expression of constraints. A quick example of a constraint in OCL is shown in Figure 1.

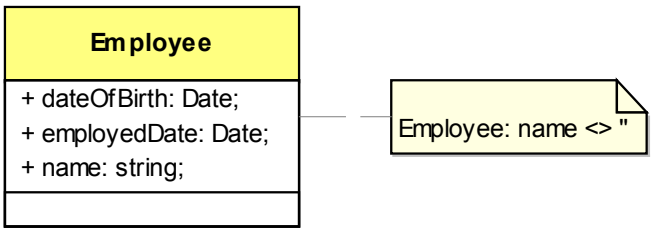


Figure 1: Basic OCL constraint

The class Employee has the constraint that all employees must have a name. This is expressed in OCL as `name <> ''`. Constraints will be covered in more detail in a future article.

OCL and Navigation

OCL is also used to define navigation. Figure 2 shows a derived relationship expressed in OCL.

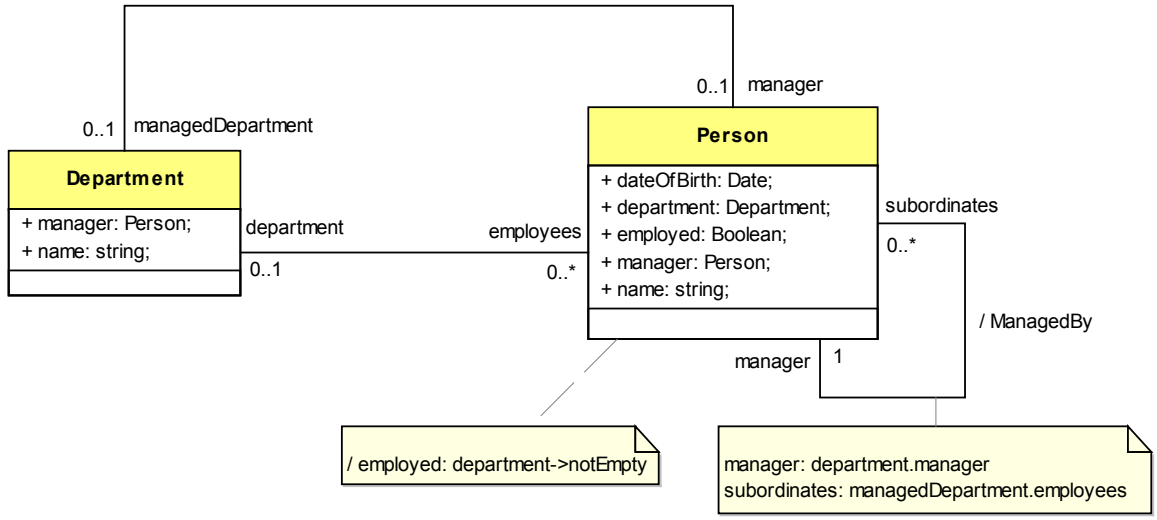


Figure 2: OCL in Derived Relationships for navigation

In Figure 2 the derived role *manager* on the class *Person* is expressed in OCL as `department.manager`. Expressed in English you might say “The manager of a person is the person who manages the department in which that person works”. The role the other way, *subordinates* , uses the OCL expression `managedDepartment.employees`.

If we couldn’t use a language like OCL in our models to describe derived relationships we would have to do one of two things:

A) Store the explicit relationship with the employee, thus creating a duplication of information in the database.
B) Don’t show the relationship and provide the information using code within the application.

In both circumstances we may lose an important business rule, “The manager of a person is the person who manages the department in which that person works”. This rule would have to be enforced in the client application, possibly in many locations.

OCL and Derived Attributes

OCL can also be used to return the value of an attribute. Returned values can be:

A) A basic data type. (string, integer, boolean, etc…).
B) A single instance of a class (an object).
C) A collection of class instances.
D) A collection of basic datatypes.

In Figure 2 the derived attribute *employed* is calculated using the OCL expression **`department->notEmpty`**. This result is a boolean value, true if the person works for a department, false if they don’t.

OCL Explorer

Example Model

The easiest way to learn OCL in Bold for Delphi is to build an application and experiment with it. For this purpose we are going to build an application that allows us to execute OCL against a small contact management model. Figure 3 is the model for our example application, if you have already completed previous articles in this series, there should be no new concepts introduced here.

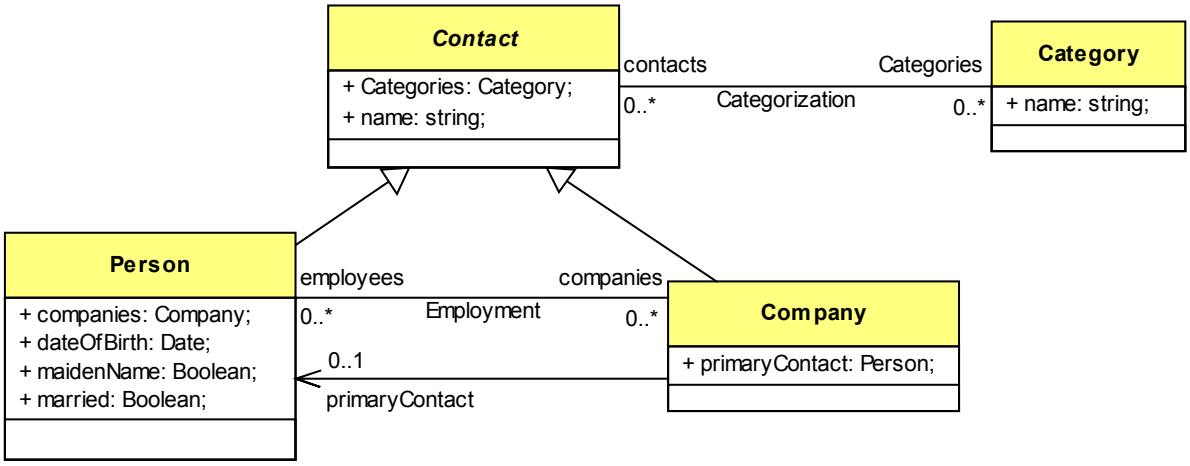


Figure 3: OCL Explorer Model

The source code for the program that implements this model is in Appendix B: Application Source Code.

The OCL explorer, Figure 4, contains a split screen. On the left you have “list one” and on the right “list two”. Each is capable of having an OCL expression set and the results will be shown in the list box and grid under the OCL display.

The OCL Explorer also has the Bold debugger available from the Tools menu. Using the debugger you can modify the data in the object space, log OCL expressions, watch memory usage, etc..

Also available from the Tools menu is the OCL Syntax Summary. This form shows a list of the OCL operations and operators included with Bold for Delphi. Because this list is generated at runtime from the runtime type information in Bold for Delphi any third-party OCL extensions should appear in this list as well. (Thanks to Jonas Hogstrom for providing the code)

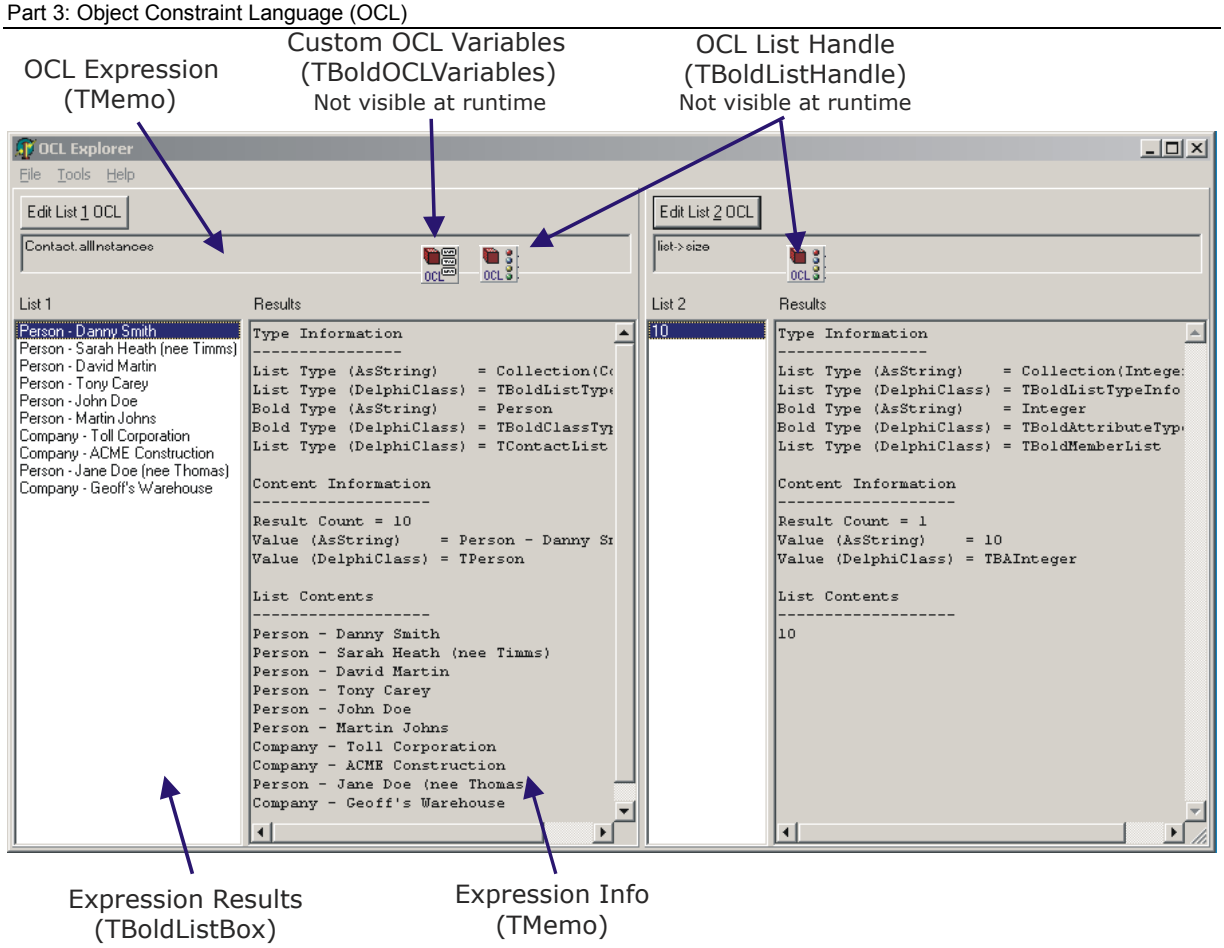


Figure 4: OCL Explorer, Main Form

When you use the OCL Editor, via the Edit Button, the expression is passed to the ListHandle which evaluates the OCL expression against the object space. The results are displayed in the ListBox and Memo.

When the application first starts the OCL Expression is ‘’ (Blank). You may wonder what can you return from a blank OCL expression? Well, the answer is metadata. If the root handle is set to one of the system handles the context is therefore the system. The system by default returns a list of all the classes in the model.

In the Info memo we have the following information:

```
Result Count = 10
Value (AsString)     = Person - Danny Smith
Value (DelphiClass) = TPerson
```

So what does this mean?

The result count is the number of items in the resulting list. Remember, because we are using a TBoldListHandle we will always get a list as a result.

The “Value (AsString)” is the string representation of the current element in the list handle. This updates depending on what item is currently selected in the list.

The “Value (DelphiClass)” is the Delphi class of the current element in the list handle. This updates depending on what item is currently selected in the list.

By pressing the Edit button the standard Bold for Delphi edit dialog is displayed.

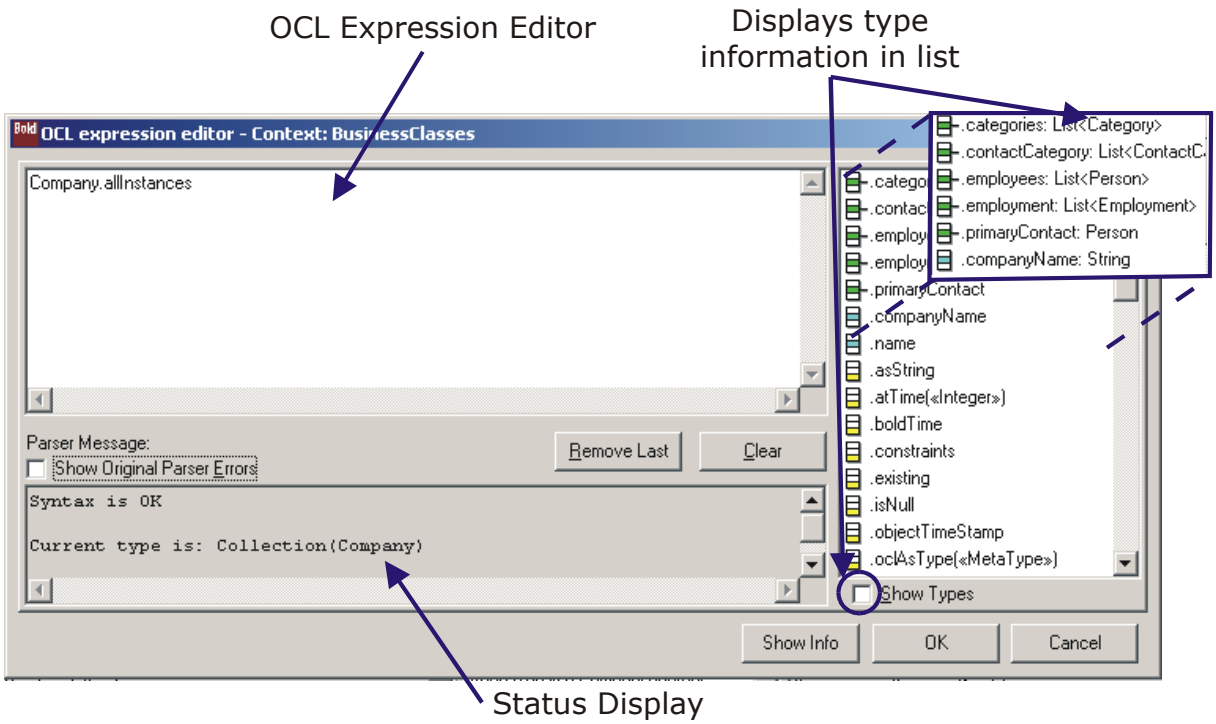


Figure 5: OCL Expression Editor

The OCL Expression editor varies the available operations in the right-hand list depending on the resulting type of the current expression. In the example Figure 5, the result is a Collection of Company objects. Therefore the right-hand list only displays options that a suitable for Company objects or for Collections in general.

How the OCL Explorer works

The following diagram shows the configuration of the Bold related components on the Main Form. The key points are:

- List 2 is cascaded off of List1. This means the default context for any expression in List 2 is the selected element in List 1.
- List 2 is linked to our user defined Variable Definitions. This means any variables declared are available for use by List 2.
- The Variable Definition defines a variable named ‘list’. This variable refers to the results of List 1 as a collection.

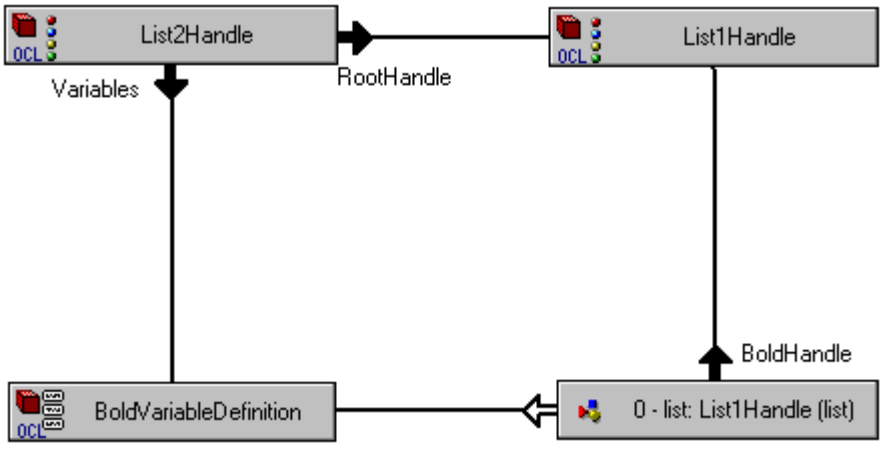


Figure 6: OCL Explorer Configuration

Using the OCL Explorer with examples in this article

Throughout this article examples are given of OCL statements. These examples are formatted in two ways.

Single Expression

The single expression example would be entered into list 1, to avoid possible errors it is best to ensure the OCL for list 2 is clear.

Example of single expression formatting.

Expression	`Company.allInstances`

Results	`Company - Toll Corporation`
	`Company - ACME Construction`
	`Company - Geoff's Warehouse`

Multiple Expressions

When 2 expressions are provided. The first expression should be entered into list 1 and the second expression into list 2.

Example of multiple expression formatting.

Expression 1	`Company.allInstances`

Results	`Company - Toll Corporation`
	`Company - ACME Construction`
	`Company - Geoff's Warehouse`

Expression 2	`self->forAll(company | company.name <> '')`
Results	`Y`

Object Constraint Language Basics

Notation

The Object Constraint Language shares some common notations with Object Pascal, but there are some differences that you need to be aware of.

'Dot' notation

In Delphi Object Pascal the dot notation is used as a navigation aid for methods and properties of objects. This same concept is used in OCL. For example in Delphi Object Pascal the code `Memo1.Lines.Count` would return the count of the number strings in the string list owned by Memo1. In this example the dot notation is used to navigate from an instance of TMemo, called Memo1, to its Lines property. Because the Lines property returns an instance of the class TstringList, we can further navigate to the property Count of the TstringList.

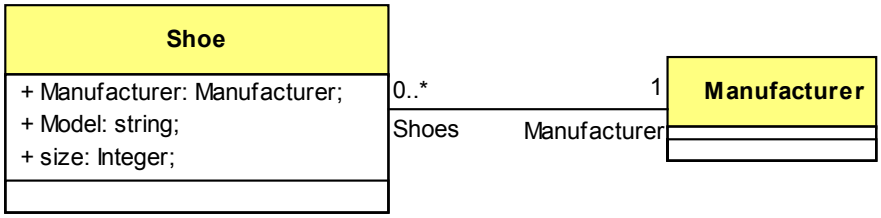
An important difference between the Delphi Object Pascal and the Object Constraint Language, is that in OCL you can **not** call operations (methods) of an object. The reason for this limitation is that OCL is side-effect free language. That is any evaluation of an OCL expression will not result in a change to the model or its information.

Collection Operator (->)

The Object Constraint Language provides implicit support for collections of objects and attributes. The operator for accessing collection operations is the combined characters – and >. The -> operator also can make an implicit conversion of a single instance into a collection containing that single instance.

As an example the OCL expression in a retail footwear store application could be `Shoe.allInstances`. This would return a collection containing all Shoes in our system. To access the attributes and operations that are explicitly available to collections we need to use the `->` notation. For example `Shoe.allInstances->size` will return the size property of the Show collection, this is the number of Shoes in the collection.

You may wonder why we need to use a special operator? Why not use the 'Dot' notation? The reason is to prevent scope conflict. An example of this condition would exist in the model:



In OCL we can directly access the attributes and roles of objects in a collection. If one of these attributes or roles is named the same as a collection operation the expression becomes ambiguous. Using a pure 'dot' notation the expression `Shoe.allInstances.size` becomes ambiguous; do you want to return a collection of all Shoe sizes? Or do you want to know how many Shoes you have?

By explicitly requiring the use of a collection operator `->` the ambiguity is removed.

> **TIP**
> Any time the collection operator is used `->`, Bold will treat the results of the expression of the left of the `->` as a collection. If the left side did not normally result in a collection, it is converted to a collection containing the one element.

Carriage Returns
OCL ignores carriage returns in expressions; the two following expressions are syntactically equivalent in the Bold for Delphi OCL parser:

```
Person.AllInstances->size
```

And

```
Person.
AllInstances
->size
```

If Then Else
OCL provides for expression branching using an If, Then Else structure. This is similar to if then else flow control in Delphi Object Pascal, but has slightly different rules.

```
If <boolean expression> then
  <expression>
else
  <expression>
endif
```

Where *<Boolean Expression>* is any OCL expression that evaluates to a boolean result (True or False) and <Expression> is any valid OCL expression.

The else part of the if-then-else structure is mandatory as the expression must return a value. It is also important that each section should return similar types, for example it is not correct if the ‘then’ sections returns a string and the ‘else’ section returns an object.

Try the following expression into the OCL explorer:

Expression 1	if Contact.AllInstances->Size > 10 then
	  'You have 10 or more contacts!'
	else
	  'You have less than 10 contacts!'
	endif

Results	You less than 10 contacts

Comments

The OCL specification states that using `--` (double hyphen) will denote a comment for anything following the `--` until the end of the line. However Bold for Delphi does not support this notation and trying to enter OCL comments will result in an error.

Special Characters

apostrophe ‘

The single apostrophe (‘) character is used in OCL to mark string literals. If the string needs to contain an apostrophe, you need to escape it using a backslash (\).

For example if you needed to enter the string literal `Geoff’s Warehouse` into an OCL expression you would have to enter `Geoff\’s Warehouse.`

Expression 1	`Contact.allInstances->select(`
	`  name = 'Geoff\'s Warehouse'`
	`)`

Results	`Company - Geoff's Warehouse`

pipe |

The pipe (|) character is used to identify the separation between the declaration of an iterator and the expression when used in various collection operations. Iterators are covered in more detail later.

colon :

The colon (:) is used with iterators for specifying type. Iterators are covered in more detail later.

hash #

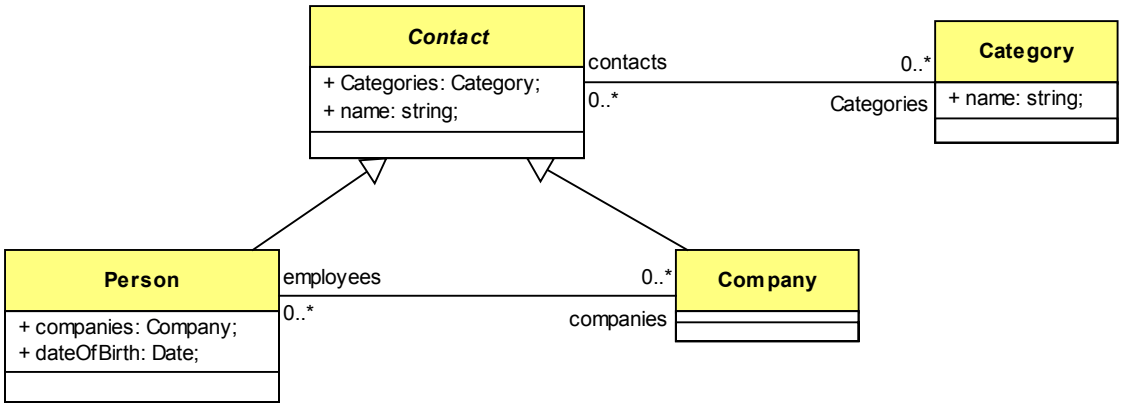
The hash (#) character is used to start date literals and time literals. Using dates and times is covered later.

Context

The UML specification uses the following notation to show the context in which an OCL expression is evaluated.

```
<Context>
<OCL Expression>
```

For example:



If we have the OCL expression `name.toUpper` it is unclear which of the two classes we want the name for, both Contact and Category have an attribute called name!

All OCL expressions in Bold for Delphi are evaluated against a pre-defined context. This context is dependant in where the OCL expression is used. Within a Model the OCL expression is evaluated within the context of the Class the expression is written in. In TBoldXXXhandles the context is set using the RootHandle property of the handle.

When using a Bold Handle (TBoldlistHandle, TboldExpressionHandle, BoldHandle property of Bold-aware controls etc…), you need to set the RootHandle property. Valid root handles are any other handle in your project (except any handle that would cause a circular dependency). Generally you will always have a `TboldSystemHandle` available to access your entire object space (as used in the OCL explorer), however nothing prevents you from cascading handles together to provide cascading updates to handles contents. This was the method demonstrated in Article 2 and is also used in the OCL Explorer.

Self Variable

The self variable is used to reference the context of the OCL expression. For example, in the UML Model editor, whilst adding an OCL expression to class, self would refer to an instance of that class. In a cascaded Bold Handle configuration, self refers to the result of the preceding Bold Handle (assigned to the RootHandle property).

The OCL Explorer application included with this article uses cascaded list handles. The `List2Handle` is linked from the `List1Handle`. The value result of any expression evaluation in list 1 can be access in list 2 using `self`.

OCL Result Types

OCL Expressions in Bold for Delphi can return various 'types' of values. The types can be roughly summarized in the following categories:

Meta Data

Information about the model. This includes classes & OCL Type information. Using the various OCL operations for Meta-data you can query you model to return all supertypes and subclasses of any class. Perform type checking and cast an object to different types.

Objects

Instances of classes.

Attribute Types

Strings, Integers, Floats, Boolean, Blobs and all the other types for attributes defined in your model. The OCL implementation in Bold for Delphi not only returns the attributes in your model, but is capable of performing various operations to return derivative values.

Collections

Lists of anything in the above categories. There are three types of collections: Bag, Sequence and Set. All three types are covered in more detail later.

Using the Object Constraint Language

Arithmetic

Some of the simplest OCL expressions are basic arithmetic. Try the following expressions in the OCL Explorer.

Expression	`2+8`
Results	`10`
Information	`Value (DelphiClass) = TBAInteger`

The result is hardly surprising, but interesting, what is a TBAInteger? Bold for Delphi has to return an object from the value property of a handle. Like TIntegerField in traditional Delphi applications, Bold has created objects that encapsulate all the types that can be returned from an OCL Expression.

I won't go further into these classes, as they are a topic of future articles.

Try the following expressions:

Expression	`4/2`
Results	`2`
Information	`Value (DelphiClass) = TBAFloat`

Expression	`(2+3)/2`
Results	`2.5`
Information	`Value (DelphiClass) = TBAFloat`

The following operators are available:

()	Parenthesis, important for controlling evaluation order.
.abs	Return the absolute value
.round	Round the numeric value to the nearest integer.
.floor	Return the integer portion of a numeric.
/ *	Division and multiplication.
+ -	Addition and subtraction.

The best part of arithmetic with OCL, is it can be freely used with numeric properties of objects, or any OCL operation that returns a numeric value.

Expression	`34.7.floor`
Results	34

Expression	`1+ Person.allInstances->size * 8`
Results	

Boolean Operators

The Bold for Delphi OCL implementation supports the common boolean evaluation operations.

The following is a list of boolean operators in Bold for Delphi's OCL implementation.

Name	Result	Context	Parameter
=	Boolean	<any>	<any>
<>	Boolean	<any>	<any>
<	Boolean	<any>	<any>
>	Boolean	<any>	<any>
<=	Boolean	<any>	<any>
>=	Boolean	<any>	<any>
or	Boolean	Boolean	Boolean
and	Boolean	Boolean	Boolean
not	Boolean	Boolean	
xor	Boolean	Boolean	Boolean
implies	Boolean	Boolean	Boolean

Most of these should be straight forward, however the `implies` operator will most likely be unfamiliar to Delphi developers.

implies

The `implies` operator is like a one way `and` operation. When used in OCL the implies operator say "if the result of my left side is true, then the result of my right side should be true to". Unlike the and operator there is no requirement for the evaluation to work in reverse.

The following truth table shows the different behavior between `and` and `implies`. If the left side of an `implies` operation is false, then the result will always be true

	A=True B=True	**A=True B=False**	**A=False B=True**	**A=False B=False**
A and B	T	F	F	F
B and A	T	F	F	F
A implies B	T	F	T	T
B implies A	T	T	F	T

The following expression performs a check on all persons to see if they comply to the business rule that "all persons who have a maiden name must be married".

Expression	`Person.allInstances->reject(`
	`  (maidenName <> '') implies married`
	`)`

Results	`Person - Sarah Heath (nee Timms)`

Although this sort of business rule would not hold up very well in today's society, it can be seen that the `and` operator would have, inappropriately enforced the requirement for all married people to have maiden names.

Accessing Classes and Objects

The primary use of OCL in Bold for Delphi is to return objects. That is instances of classes within your object space. Lets look at a basic OCL expression and deconstruct it.

allInstances

The `allInstances` operation returns a collection of all objects in the objects space for a class in the model.

Expression	`Person.allInstances`

Results	`Person - Danny Smith`
	`Person - Sarah Heath (nee Timms)`
	`Person - David Martin`
	`Person - Tony Carey`
	`Person - John Doe`
	`Person - Martin Johns`
	`Person - Jane Doe (nee Thomas)`

Information	`Value (AsString)  = Person - Danny Smith` `Value (DelphiClass) = TPerson`

The expression returns all the people, both those already loaded in memory and in our persistence layer (database). The expression ‘Person’ returns a class reference, in this case a reference to the Person class. The expression ‘allInstances’ returns a collection of all instances of the class it is associated with.

You will also have noticed that the Value property returns the TPerson instance of “Danny Smith”, this is the currently selected item in the list. This value will change if a different item is selected.

allLoadedObjects

The `allLoadedObjects` is similar to the `allInstances` operation except only objects that have already been loaded into memory will be returned.

allInstancesAtTime

The `allInstancesAtTime` operation returns a collection of all objects in the objects space at the time passed in as the parameter. This will only work if you have purchased the optional ‘Object Versioning Extension’ from BoldSoft. This extension is a powerful add-on to Bold for Delphi that allows you to navigate the object space over time.

emptyList

The emptyList operation can be used to access an empty list of any class type.

Expression	`Contact.emptyList`
Results	(Empty List of Contacts)

Accessing Attributes

If you are only interested in a particular attribute of a class, you can choose to return just the attribute.

Expression	`Person.allInstances.dateOfBirth`
Results	`9/12/1972` (Danny Smith) `18/02/1968` (Sarah Heath) `2/12/1956` (David Martin) `12/05/1959` (Tony Carey) `6/01/1968` (John Doe) `30/12/1966` (Martin James) `12/09/1973` (Jane Doe)

In the above example a collection of dates is returned. These are of course a collection of the date's of birth for all persons.

Working with Dates and Times

Date and Time Literals

Dates in Bold for Delphi are represented using the format `#YYYY-MM-DD`. This is independent of the regional settings on the PC and is from the ISO-standard for notation of dates/times. You must include all digits with leading zeros.

Valid OCL Dates:

```
#1968-01-01
#1968-12-14
```

Invalid OCL Dates:

`#1968-15-02`	(Must be Year-Month-Day, 15 is an invalid month)
`1958-12-03`	(Must include the hash # prefix)
`#2001-10-1`	(Must include leading zero's)

An example expression using a Date. This returns a list of all people born after 1st January 1965.

Expression	`Person.AllInstances->Select(`
	`  dateOfBirth >#1965-01-01`
	`)`

Results	`Person - Danny Smith`
	`Person - Sarah Heath (nee Timms)`
	`Person - John Doe`
	`Person - Martin Johns`
	`Person - Jane Doe (nee Thomas)`

A similar format can be used for handling times. The formats `#HH:MM:SS` and `#HH:MM` are both supported.

Valid OCL Times:

```
#16:30
#01:45:30
```

Invalid OCL Times:

`#24:00:00`	(Hours must be between 0 and 23)
`#05:60:04`	(Minutes and seconds must be between 0 and 59)
`#12:2`	(Must include leading zero)

The Bold for Delphi OCL implementation currently does not support a notation for expressing a date and a time together.

inDateRange and inTimeRange

The `inDateRange` operation is used to check if a date or time is in between two values. The following expression returns all people born between 1960 and 1969.

Expression	`Person.allInstances->select(`
	`  dateOfBirth.inDateRange(`
	`  #1960-01-01.dateTimeAsFloat,`
	`  #1969-12-30.dateTimeAsFloat )`
	`)`

Results	`Person - Sarah Heath (nee Timms)`
	`Person - John Doe`
	`Person - Martin Johns`

Interestingly the parameters to both of these operations are expected to be numeric values, this is easily achieved by using the `dateTimeAsFloat` operation.

formatDateTime

The `formatDateTime` operation takes a standard Delphi datetime format string. Although the `asString` operation will convert a datetime value to a string, it uses the regional settings. With `formatDateTime` you have the flexibility to specify exactly how you want to convert you date or time.

Expression 1	`Person.allInstances`
Results	(returns a list of all persons)
Expression 2	`self.dateOfBirth.formatDateTime('d mmmm yyyy')`
Results	`9 December 1972` (will change depending which element is selected in list 1)

strToDate and strToDateTime

The are basic conversion operations similar to their Delphi counterparts. If the string doesn't conform to the requirements for the conversion an exception is raised.

sumTime

When working with a collection you can use the `sumTime` operation to add all the times together.

Working with Nulls

Bold for Delphi provide support for dealing with null values in OCL. Any attribute can be checked for 'nullness' (OK, I made that word up) using the operation `isNull`. This returns true if the attribute is currently null and false if it isn't.

> **TIP**
> If you want to specify an attribute as being null for its initial value (In the model editor), you can use the key word <NULL>. This however is not part of the actual OCL implementation.

Working with Strings

OCL provides extra support for Strings over the other basic types. Any of the following operations can be used on any OCL statement that returns a string.

concat(«string»)

This is another method for concatenating strings together. Similar to using the + operator. The follow expression return the same result:

```
Person.allInstances.name.concat(' some text')
```

And

```
Person.allInstances.name + ' some text'
```

length

The OCL specification specifies the `size` operation to return the length of a string. Bold for Delphi does not use `size` and instead uses `length`.

This example returns a collection of the lengths of each persons name.

Expression	`Person.allInstances.name.length`
Results	11 11 12 11 8 12 8

pad(«integer»,«string») and postPad(«Integer»,«String»)

The `pad` operation allows us to pad a string to be at least «integer» characters in length. The «string» characters are used to fill in any padding that occurs. In the following example you will see that the `pad` operation does not trim the length of a string if it is greater than the specified quantity. The padding is applied to the start of the string when using the `pad` operation.

Expression	`Person.allInstances.name.pad(10,'*')`
Results	Danny Smith Sarah Heath David Martin Tony Martin **John Doe Martin Johns **Jane Doe

The `postPad` operation is the same as the `pad` operation, except the padding occurs at the end of the string.

Expression	`Person.allInstances.name.postPad(10,'*')`

Results	`Danny Smith`
	`Sarah Heath`
	`David Martin`
	`Tony Carey`
	`John Doe**`
	`Martin Johns`
	`Jane Doe**`

regExpMatch(«String»)

The `regExpMatch` operation will return True or False depending if the string matches the regular expression in «`String`».

The following expression will return true if a contacts name starts with T or S.

Expression	`Contact.allInstances.name.regExpMatch('^[TS]')`
Results	`N` (Danny Smith) `Y` (Sarah Heath) `N` (David Martin) `Y` (Tony Carey) `N` (John Doe) `N` (Martin Johns) `Y` (Toll Corporation) `N` (ACME Construction) `N` (Jane Doe) `N` (Geoff's Warehouse)

The operation `regExpMatch` is most useful when combined with filtering operations on collections (covered later). The following example filters the collection of all contacts to only the few that start with T or S:

Expression	`Contact.allInstances->select(`
	`  name.regExpMatch('^[TS]')`
	`)`

Results	`Person - Sarah Heath (nee Timms)`
	`Person - Tony Carey`
	`Company - Toll Corporation`

sqlLike(«String») and sqlLikeCaseInsesitive(«String»)

Working similar to the `regExpMatch` operation, these two operations take a string in format of the ANSI SQL Like command. Basically this allows the use of the % (percent) character to act as a wild card.

In the following example any contact with a name containing ‘ra’ will be returned.

Expression	`Contact.allInstances->select(`
	`  name.sqlLike('%ra%')`
	`)`

Results	`Person - Sarah Heath (nee Timms)` `Company - Toll Corporation`

The `sqlLike` operation performs a case-sensitive match. To ignore case you need to use the `sqlLikeCaseInsesitive` operation.

Expression	`Contact.allInstances->select(`
	`  name.sqlLikeCaseInsensitive('s%')`
	`)`

Results	`Person - Sarah Heath (nee Timms)`

strToDate, strToDateTime, strToInt and strToTime
The are basic conversion operations similar to their Delphi counterparts. If the string doesn't conform to the requirements for the conversion an exception is raised.

subString(«Integer»,«Integer»)
Returns the subset of the original string consisting of the characters between the two positions passed in as parameters. If the terminating position past the end of the original string, the subset ends at the original strings last character.

Expression	`Contact.allInstances.name.subString(4,12)`

Results	`ny Smith`
	`ah Heath`
	`id Martin`
	`y Carey`
	`n Doe`
	`tin Johns`
	`l Corpora`
	`E Constru`
	`e Doe`
	`ff's Ware`

toLower and toUpper
These operations perform basic case conversion on the string, either returning a result that is all upper-case or all lower-case.

Collections

In many of the above OCL examples a list of some sort has been the result. In OCL these lists are referred to as collections. Collections can have various operations performed on them, both from the OCL standard and BoldSoft extensions. Collections will prove to be one of the most powerful tools you will use.

Types of Collections

Bag

A Bag is simply a collection of elements. Duplicates are retained and the order of elements is not important. In the following example a list of categories is returned for every contact. Notice that the same elements are repeated many times. A bag doesn't care for order or duplicates it returns every element regardless.

Expression	`Contact.allInstances.categories`

Results	`Category - Business`
	`Category - Personal`
	`Category - Business`
	`Category - Personal`
	`Category - Business`
	`Category - Personal`
	`Category - Business`
	`Category - Business`
	`Category - Business`

Sequence

A Sequence will maintain the order that the elements have in the collection. This is achieved by the model option 'ordered' within a role.

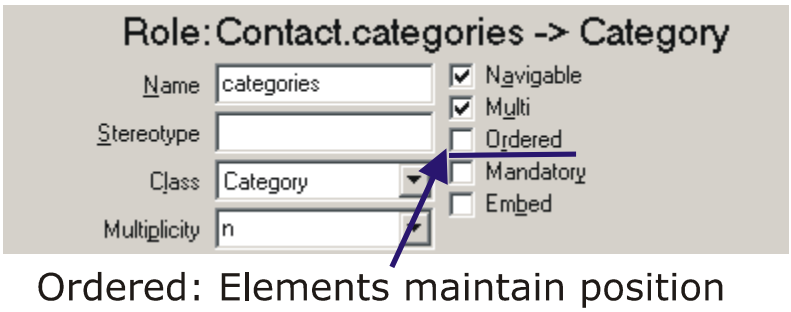


Figure 7: Role's "ordered" option

If a role is marked as ordered, then the position of the elements with the collection will be maintained, even between sessions. This is achieved by recording the order into the database.
In Figure 7 the option has not been set, this means the order that elements are assigned is not guaranteed. It may change each time the OCL is evaluated.

Set

A set is a collection that only contains unique values. Using the above example, and converting it to a set, the results reflect the unique values only.

Expression	`Contact.allInstances.category->asSet`
Results	`Category - Business` `Category - Personal`

Collect Operation

OCL has a very powerful operation called collect. This allows us to aggregate list together easily. The following example demonstrates this.

Expression	`Company.allInstances->collect(employees)`

Results	`Person - Danny Smith`
	`Person - David Martin`
	`Person - Tony Carey`
	`Person - David Martin`
	`Person - Jane Doe (nee Thomas)`

We take the collection of `Company.allInstances` and for each company in the collection we `collect` the `employees` and combine them together. Effectively giving us a collection of all employees for all companies.

Collects can be used more than one level deep. Take the following collect operation:

Expression	`Company.allInstances->collect(employees)` `->collect(dateOfBirth)`

Results	`9/12/1972`
	`2/12/1956`
	`12/05/1959`
	`2/12/1956`
	`12/09/1973`

In the above OCL we are concatenating two collect operations. Giving us all the birth dates, for all the employees of every company. However the OCL expression quickly gets long, and this is very common operation.

Fortunately OCL has a short-cut notation for the collect operation. Simply navigate to the property using the standard ‘dot’ notation. OCL assumes that navigation to a property on a collection is a collect operation. The previous OCL can be simplified to the following.

Expression	`Company.allInstances.employees.dateOfBirth`

Results	`9/12/1972`
	`2/12/1956`
	`12/05/1959`
	`2/12/1956`
	`12/09/1973`

The question may be asked. “If the dot notation can be used to replace collect(), why even have collect()?’.

The answer is the dot notation, as used with collect, can only replace the basic use of concatenation collections when they are attributes or roles of a class. If the collecting is not happening with the attributes or roles you need to use collect. Have a look at the following example.

Expression	`Company.allInstances->collect(employees->size)`
Results	`2` (Toll Corporation) `2` (ACME Construction) `1` (Geoff's Warehouse)

The expression returns a collection containing the number of employees for each company. If we used the dot notation the result would be very different,

Expression	Company.allInstances.employees->size
Results	5

Not what we were after at all. Five is the number employees across all companies.

By using collect() we can collect the results of a ‘sub’ expression for each element in the original collection.

Iterators (Loop Variables)

When working with collection operations it is sometime necessary to refer to collections element as part of the operations expression. This can be usedby defining a loop variable, also called an iterator. (The OCL specification uses the iterator term).

The following expression returns a collection of companies that have blank names.

```
Company.allInstances->forAll(name = '')
```

If we rework the expression to use an iterator it would look like this:

```
Company.allInstances->forAll(c | c.name = '')
```

The pipe character is used to identify an iterator. In the example above ‘c’ now becomes a reference to the element in the collection. On the right side of the pipe character the iterator (c) can be used.

In most cases the use of an iterator is not required. The context of the expression in the operation is explicitly set to the current element of the collection. However in situations where you need to explicitly refer to the element in an expression the iterator will be necessary. This normally occurs when you want to perform some sort of boolean comparison or arithmetic operation.

For example: If we wanted a list of all people who are a primaryContact we could use the following expression

Expression	`Person.allInstances->select(`
	`  p | companies.primaryContact->includes( p )`
	`)`

Results	`Person - Danny Smith`
	`Person - David Martin`
	`Person - Jane Doe (nee Thomas)`

In the above example because we want to use the context in the select statement inside the nested statement, we need to use an iterator. Before you start sending in the emails, I do realise that the model can be navigated directly from Company to retrieve the same result, and the expression is simpler:

Expression	`Company.allInstances.primaryContact`

Results	`Person - Danny Smith`
	`Person - David Martin`
	`Person - Jane Doe (nee Thomas)`

However, that doesn’t demonstrate iterators. I assure you that given a model even mildly more complex than the one we are using, you will find iterators a useful mechanism.

Another way in which using an iterator can add value to you OCL expression is to aid in documentation. When the entire model is being evaluated in an expression the context is always clear. However, when using cascaded handles it can sometime be

difficult to know the context of an expression just by looking at it. Enter the following expressions into the OCL Explorer:

Expression 1	`Company.allInstances`

Results	`Company - Toll Corporation`
	`Company - ACME Construction`
	`Company - Geoff's Warehouse`

Expression 2	`self->forAll(company | company.name <> '')`
Results	`Y`

In the above example we are simply checking in expression 2 that all the companies have a name. Although the expression `self->forAll(name <> '')` would have sufficed to return the correct result, by adding the iterator the expression provides some documentation without you having to track down expression 1 in your application.

The OCL specification allows for multiple iterators to be defined, however this is not supported in Bold for Delphi.

The OCL Specification also states that a colon (:) can be used to specify the type of an iterator. However, this is **not** supported by Bold for Delphi. You can specify the type but it will be ignored. It is not necessary to specify the type of an iterator as it is implicit based on the type of the element in the collection.

Sorting

orderBy(«any») and orderDescending(«any»)

`orderBy` takes an OCL expression as a parameter and then evaluates it for each entry in the collection. The results are then used to order the collection.

Expression	`Contact.allInstances->orderby(name)`

Results	`Company - ACME Construction`
	`Person - Danny Smith`
	`Person - David Martin`
	`Company - Geoff's Warehouse`
	`Person - Jane Doe (nee Thomas)`
	`Person - John Doe`
	`Person - Martin Johns`
	`Person - Sarah Heath (nee Timms)`
	`Company - Toll Corporation`
	`Person - Tony Carey`

Not surprisingly the `OrderDescending` operation does the same, in descending order.

Expression	`Contact.allInstances->orderdescending(name)`

Results	`Person - Tony Carey`
	`Company - Toll Corporation`
	`Person - Sarah Heath (nee Timms)`
	`Person - Martin Johns`
	`Person - John Doe`
	`Person - Jane Doe (nee Thomas)`
	`Company - Geoff's Warehouse`
	`Person - David Martin`
	`Person - Danny Smith`
	`Company - ACME Construction`

However, because the `orderBy` operation accepts more complex expression, we could also do the following. In this example we are ordering using the persons maiden name if they have one.

Expression	`Person.allInstances->orderBy(`
	`  if maidenName = '' then`
	`  lastName + firstName`
	`  else`
	`  maidenName + firstName`
	`  endif`
	`)`

Results	`Person - Tony Carey`
	`Person - John Doe`
	`Person - Martin Johns`
	`Person - David Martin`
	`Person - Danny Smith`
	`Person - Jane Doe (nee Thomas)` <<ordered using Thomas
	`Person - Sarah Heath (nee Timms)` <<ordered using Timms

When using the `orderBy` and `orderDesceding` operations you need to understand how Bold for Delphi sorts the different possible types. Now with the basic types (String,Integer etc..) the sorting is straight forward. However, the sorting is actually performed internally using the `CompareToAs` method of the base Bold for Delphi class `TBoldElement`.

So what does that mean? It means it's possible for you to define a custom way of sorting your domain objects by override the framework method `CompareToAs` in the Bold UML Model editor.

Numeric Collection Operations

sum, minValue and maxValue

Returns the sum of all numeric values in a collection of numeric values.

The following example returns a collection containing the number of employees for each company.

Expression	`Company.allInstances->collect(employees->size)`
Results	`2` (Toll Corporation) `2` (ACME Construction) `1` (Geoff's Warehouse)

By applying the `sum` operation we can retrieve the total number of employees for all companies.

Expression	`Company.allInstances->collect(employees->size)->sum`
Results	5

If we used the `minValue` operation the result would be the lowest value in the collection. If we used the `maxValue` operation the result would be the highest value in the collection.

count(«any»)

This operation will return how many times an element appears in the collection. The element you are looking for is passed in as the argument and must conform to the type of the list. So if we have a list of Contact's we can pass in a Person or Company as the argument because they conform to the type Contact.

In the following example List 1 will contain a list of all employees for all companies. In List 2 the expression applies the `count` operation on the result of list one using the parameter of self, which in the context of List 2 is the currently selected item in list 1. So as you select each employee down the list, the second list will show you how many times that employee appears in the list.

Expression 1	`Company.allInstances.employees`

Results	`Person - Danny Smith`
	`Person - David Martin`
	`Person - Tony Carey`
	`Person - David Martin`
	`Person - Jane Doe (nee Thomas)`

Expression 2	`list->count(self)`
Results	(Depends on which Person in list 1 you select)

Navigating Collections

first and last

The `first` operation returns the first element in the collection; the `last` operation returns the last element in the collection.

Expression	`Contact.allInstances->first`
Results	`Person - Danny Smith`

Of course because we can cascade operations in OCL, we can sort before calling first or last (or any other operation).

Expression	`Contact.allInstances->orderby(name)->last`
Results	`Person - Tony Carey`

at(«Integer»)

The `at` operation returns the element at the location «Integer». The following expression returns the second element in the collection.

Expression	`Person.allInstances->at(2)`
Results	`Person - Sarah Heath (nee Timms)`

Collection Boolean Operations

includes(«any»)

The `includes` operation returns true if the element passed in as the parameter exists in the collection, false if it doesn't.

The following example uses the `includes` operations to check if the person selected in list one is an employee of the first company in our object space.

Expression 1	`Person.allInstances`
Results	(Returns all persons)
Expression 2	`Company.allInstances->first.employees->` `  includes(self)`
Results	`Y or N` (Changes with the currently selected element in list 1)

forAll(«Boolean»)

The `forAll` operation applies the boolean expression passed to it to all elements in the collection. If every element evaluates the expression to true, the result is true. If any of the elements evaluates the expression as false, the entire expression returns false.

The following expression returns true if all people are married, false if even a single one of them is not.

Expression	`Person.allInstances->forAll(p | p.married)`
Results	`N` (No)

includesAll(«list<any Arg>»)

Compares contents of to collections. If every element of the collection appears in the passed in collection the result is True, otherwise false. The collection passed can have other elements, so lists are not necessarily equal, but at least every element of the collection does exist in the passed in collection.

The following example checks that all employees are indeed listed as 'busineess' contacts.

Expression 1	`Category.allInstances->select(`
	`  name = 'business'`
	`).contacts`

Results	`Person - Tony Carey`
	`Person - Danny Smith`
	`Person - David Martin`
	`Company - ACME Construction`
	`Company - Toll Corporation`
	`Person - Jane Doe (nee Thomas)`

Expression 2	`Company.allInstances.employees->` `  includesAll(list)`
Results	`Y` (yes)

isEmpty and notEmpty

These operations simply return whether a collection is empty (contains no elements) or not.

The following expression returns true if any categories named ‘business’ exists in the object space.

Expression	`Category.allInstances->select(`
	`  name = 'business'`
	`)->notEmpty`

Results	`Y` (Yes)

exists(«Boolean»)

The `exists` operation checks if an element exists in the collection. If it does it returns true if not false.

Lets re-write the above `notEmpty` example to use `exists`.

Expression	`Category.allInstances->exists(`
	`  name = 'business'`
	`)`

Results	`Y` (Yes)

Filtering and Querying

A lot of what we do with collections will result in transforming a collection (or multiple collections) into a new collection, containing specific elements for our needs. Bold for Delphi provides many ways to manipulate collections and retrieve just the elements you need.

select(«Boolean») and reject(«Boolean»)

The `select` and `reject` operations take any valid OCL boolean expression and evaluates it for each element in the collection. For the `select` operation each element that evaluates true will be included in the resulting collection. For the `reject` operation each element that evaluates true will be excluded from the resulting collection.

The following expression selects all people born after 1 January 1968.

Expression	`Person.allInstances->select(`
	`  dateOfBirth > #1968-01-01`
	`)`

Results	`Person - Danny Smith`
	`Person - Sarah Heath (nee Timms)`
	`Person - John Doe`
	`Person - Jane Doe (nee Thomas)`

The following expression rejects all companies that only have one employee

Expression	`Company.allInstances->reject(`
	`  employees->size = 1`
	`)`

Results	`Company - ACME Construction`

difference(«list<any Arg>»)

The `difference` operation takes a collection of anything as its parameter. This collection is compared with the collection you are performing the difference on and any items in the original collection that don't appear in the second collection are returned as the result.

The following example does a difference between the different categories to determine which people are members of the 'Personal' category and not members of the 'Business' category.

Expression	`Category.allInstances->select(`
	`  name = 'Business'`
	`).contacts`

Results	`Person - Tony Carey`
	`Person - Danny Smith`
	`Person - David Martin`
	`Company - ACME Construction`
	`Company - Toll Corporation`
	`Person - Jane Doe (nee Thomas)`

Expression	`Category.allInstances->select(`
	`  name = 'Personal'`
	`).contacts->difference(list)`

Results	`Person - John Doe` `Person - Sarah Heath (nee Timms)`

symmetricDifference(«list<any Arg>»)

The `symmetricDifference` operation returns the elements of both collections that don't appear in both collections together (ie. Elements that appear in exactly one of the collections).

The following example show the symmetric difference between the two categories. Since 'David Martin' is the only contact to exist in both categories, he is excluded from the final result.

Expression	`Category.allInstances->select(`
	`  name = 'Business'`
	`).contacts`

Results	`Person - Tony Carey`
	`Person - Danny Smith`
	`Person - David Martin`
	`Company - ACME Construction`
	`Company - Toll Corporation`
	`Person - Jane Doe (nee Thomas)`

Expression	`Category.allInstances->select(`
	`  name = 'Personal'`
	`).contacts->symmetricDifference(list)`

Results	`Person - John Doe`
	`Person - Sarah Heath (nee Timms)`
	`Person - Tony Carey`
	`Person - Danny Smith`
	`Company - ACME Construction`
	`Company - Toll Corporation`
	`Person - Jane Doe (nee Thomas)`

intersection(«list<any Arg>»)

The `intersection` operation return the elements of both collections that appear in both collections together.

Expression	`Category.allInstances->select(`
	`  name = 'Business'`
	`).contacts`

Results	`Person - Tony Carey`
	`Person - Danny Smith`
	`Person - David Martin`
	`Company - ACME Construction`
	`Company - Toll Corporation`
	`Person - Jane Doe (nee Thomas)`

Expression	`Category.allInstances->select(`
	`  name = 'Personal'`
	`).contacts->intersection(list)`

Results	`Person - David Martin`

union(«list<any Arg>»)

The `union` operation returns a collection that contains all the elements from both collections

Expression	`Category.allInstances->select(`
	`  name = 'Business'`
	`).contacts`

Results	`Person - Tony Carey`
	`Person - Danny Smith`
	`Person - David Martin`
	`Company - ACME Construction`
	`Company - Toll Corporation`
	`Person - Jane Doe (nee Thomas)`

Expression	`Category.allInstances->select(`
	`  name = 'Personal'`
	`).contacts->union(list)`

Results	`Person - John Doe`
	`Person - David Martin`
	`Person - Sarah Heath (nee Timms)`
	`Person - Tony Carey`
	`Person - Danny Smith`
	`Person - David Martin`
	`Company - ACME Construction`
	`Company - Toll Corporation`
	`Person - Jane Doe (nee Thomas)`

You will notice that because 'David Martin' is a member of both categories, he now appears twice in the resulting list. To ensure no duplicates you could apply the `asSet` operation.

excluding(«any class»)

The `excluding` operation returns a subset of the original collection, excluding the element passed as the parameter.

The following example produces a collection in list two, containing all the elements of list one, minus the currently selected element.

Expression 1	`Person.allInstances`
Results	(List of all persons in the object space)
Expression 2	`list->excluding(self)`
Results	(List of all persons in object space, minus the person currently selected in list 1)

including(«any class»)

The `including` operation returns a superset of the original collection containing the original collection and the element passed in as the parameter.

The following example appends the first person in the object space with a collection of all the companies in the object space.

Expression 1	`Company.allInstances`

Results	`Company - Toll Corporation`
	`Company - ACME Construction`
	`Company - Geoff's Warehouse`

Expression 2	`list->including(`
	`  Person.allInstances->first`
	`)`

Results	`Company - Toll Corporation`
	`Company - ACME Construction`
	`Company - Geoff's Warehouse`
	`Person - Danny Smith`

subsequence(«Integer»,«Integer»)

The `subsequence` operation works with index of the collection elements to return a subset of the original collection. The parameters to `subSequence` are the start and end indexes of the elements you want to include in the result.

The following example returns elements two through to six if the collection in list one.

Expression 1	`Person.allInstances`

Results	`Person - Danny Smith`
	`Person - Sarah Heath (nee Timms)`
	`Person - David Martin`
	`Person - Tony Carey`
	`Person - John Doe`
	`Person - Martin Johns`
	`Person - Jane Doe (nee Thomas)`

Expression 2	`list->subSequence(2,6)`

Results	`Person - Sarah Heath (nee Timms)`
	`Person - David Martin`
	`Person - Tony Carey`
	`Person - John Doe`
	`Person - Martin Johns`

Working with Type information

OCL provides operations for accessing and working with the type information of our model.

allInstances, allLoadedObjects, allInstancesAtTime and emptyList

These were all covered earlier in the article. They are included here because they all operation on OCL type information.

oclType and typeName

The `oclType` operation returns the type of an object. The operation `typeName` returns the type as string.

Expression 1	`ContactRoot.allInstances`
Results	(Collection of all objects in the object space)
Information 2	`self.oclType`
Results	(The type of the currently selected element in list one)

filterOnType(«MetaType»)

The `filterOnType` operation returns a subset of a collection filtered to only contain objects of the type passed in as the parameter.

The following example fills list two with all the Category objects in list one.

Expression 1	`ContactRoot.allInstances`
Results	(Collection of all objects in the object space)
Information 2	`list->filterOnType(Category)`
Results	`Category - Business` `Category - Personal`

oclIsTypeOf(«MetaType»)

The `oclIsTypeOf` operation checks if the instances is of the type passed in as the parameter. If it is of that type it returns true otherwise false. The type must be an exact match for the actual type of the instance. Checking if a Person is an `oclTypeOf` Contact would return false.

Expression 1	`ContactRoot.allInstances`
Results	(Collection of all objects in the object space)
Information 2	`self.oclIsTypeOf(Person)`
Results	`N or Y` (Changes depending on the type of the element selected in list one. Y if a Person is selected, otherwise N)

oclIsKindOf(«MetaType»)

The `oclIsKindOf` operation checks if the instances is of the type passed in as the parameter. If it is of that type it returns true otherwise false. The type must be the actual type or any superclass of the instance. Checking if a Person is an `oclKindOf` Contact would return true.

Expression 1	`ContactRoot.allInstances`
Results	(Collection of all objects in the object space)
Information 2	`self.oclIsKindOf(Contact)`
Results	`N or Y` (Changes depending on the type of the element selected in list one. Y if any Person or Company is selected, otherwise N)

oclAsType(«MetaType»)

The `oclAsType` operation typecasts the instance to any other type in the model. If the cast is invalid (because it is not a valid supertype of the instance) an OCL Invalid Cast error will be raised.

Expression 1	`ContactRoot.allInstances`
Results	(Collection of all objects in the object space)
Information 2	`self.oclAsType(Contact)`
Results	(If the selected element in list one is a type of Contact, the instance is returned. If not an error is raised)

safeCast(«MetaType»)

The `safecast` operation is the same as `oclAsType` except no error will be raised. If the type doesn't correctly cast, a nil reference is returned.

Expression 1	`ContactRoot.allInstances`
Results	(Collection of all objects in the object space)
Information 2	`self.oclAsType(Contact)`
Results	(If the selected element in list one is a type of Contact, the instance is returned. If not nil is returned)

superTypes

The `superTypes` operation returns the immediate supertypes of the metatype. Because Delphi and Bold for Delphi do not support multiple inheritance this will always be a collection of one element.

Expression 1	`Company.superTypes`
Results	`Contact`

allSuperTypes

To retrieve all supertypes of a class use the `allSuperTypes` operation. This will walk the class hierarchy to return all super types.

Expression 1	`Company.allSuperTypes`
Results	`Contact` `ContactRoot`

allSubClasses

Unlike Delphi, the rich runtime model information in Bold for Delphi, allows us to query a class for all subclasses.

The following example lists all classes descending from the class Contact.

Expression	`Contact.allSubClasses`

Results	`Contact`
	`Company`
	`Person`

attributes

The `attributes` operation returns a list of all attributes for a class. This will include all inherited attributes.

Expression	`Person.attributes`

Results	`name`
	`dateOfBirth`
	`married`
	`maidenName`
	`firstName`
	`lastName`

associationEnds

The `associationEnds` operation returns all roles for a class. This will include all inherited roles.

Expression	`Person.associationEnds`

Results	`categories`
	`contactCategory`
	`companies`
	`employment`
	`associationEnd`

constraints

The `constraints` operation will all constraints for a class. This will include all inherited constraints.

Using OCL with Bold for Delphi

OCL in Models

There are various locations you can use OCL in your model.

Classes

There are three locations you can use OCL in your class:

Default String Rep

The Default String Representation determines how an object is visualized at runtime. An example is if you display the results of the OCL expression `Contact.allInstances` in a list, the default representation will be used to render the objects.

Constraints

The editor for constraints allows you to enter multiple OCL expressions. These can be queried at runtime to determine if the object meets the constraints. There is a lot more that needs to be said about using constraint, but that is a topic for a future article.

Derived Expressions

This field is used to specify OCL expressions for inherited derived expressions. In the base class the expressions entered as part of the attribute. However, since the attribute is not repeated in the descendant classes this field is the container for all derived expression.

Derived Expressions are entered in the format:

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

In article 2: Extending Models we used the derived expressions field to change the behavior the monthlyCost attribute in our WorkPlace, Company and Department classes.

Attributes

Constraints

Like Classes constraints can be specified for attributes.

Derivation OCL

If your attribute is tagged as derived you can enter a Derivation OCL expression. Whenever the contents of the attribute is queried at runtime, the OCL expression is evaluated and the results become the value for the attribute.

Role

Constraints

Like Classes constraints can be specified for roles.

Derivation OCL

If the association owning the role is tagged as derived you can enter a Derivation OCL expression. Whenever the contents of the role is queried at runtime, the OCL expression is evaluated and the results become the value for the attribute.

Association

Constraints

Like Classes constraints can be specified for associations.

OCL in Components

When using OCL with Bold Components there are 2 basic types, Handles and GUI controls. In general you place an OCL expression in a Handle when you want to share the results amongst other components (like we did in the OCL Explorer). When the OCL is specific to a GUI control you can place the OCL directly in the GUI control.

Handles

The two most common handles used will be the TboldListHandle and TboldExpressionHandle. They are very similar in operation, the main difference being that TboldListHandle is designed to work with lists, where as the TboldExpressionHandle works with single elements.

The most important properties to set for either of these is the Expression (in OCL ofcourse) and the RootHandle. The RootHandle is what provides the context for the expression. This may be any other handle.

GUI Controls

The key part of all Bold-aware controls is the property BoldProperties and RootHandle. The RootHandle needs to be set to any available handle, this provides the context for the BoldProperties property

The BoldPropeties property is a TboldStringFollowerController and it contains the information for retrieving the value to be displayed and some of the properties required to render the values. The OCL is entered in the Expression property.

Appendix A: OCL Quick Reference

The following table is list of the OCL operators and operations in Bold for Delphi.

Name – The operation or operator.
Result – The type of the result.
Context – The context required for this operation/operator to work.
Parameter 1 – If the operation/operator requires at least one parameter, this is the type of the parameter.
Parameter 2 – If the operation/operator requires two parameters, this is the type of the parameter.

Name	Result	Context	Parameter 1	Parameter 2
-	Least common supertype of source element and parameter 1	Numeric	Numeric	
*	Least common supertype of source element and parameter 1	Numeric	Numeric	
/	Float	Numeric	Numeric	
+	Least common supertype of source element and parameter 1	<any>	<any>	
<	Boolean	<any>	<any>	
<=	Boolean	<any>	<any>	
<>	Boolean	<any>	<any>	
=	Boolean	<any>	<any>	
>	Boolean	<any>	<any>	
>=	Boolean	<any>	<any>	
abs	Same as source element	Numeric		
allInstances	Object list	MetaType		
allInstancesAtTime	Object list	MetaType	Integer	
allLoadedObjects	Object list	MetaType		
allSubClasses	Collection(MetaType)	MetaType		
allSuperTypes	Collection(MetaType)	MetaType		
and	Boolean	Boolean	Boolean	
append	Least common supertype of source element and parameter 1	Collection()	<any class>	
asBag	Same as source element	Collection()		
asSequence	Same as source element	Collection()		
asSet	Same as source element	Collection()		
associationEnds	Collection(String)	MetaType		
asString	String	<any>		
at	Same as list elements of first argument	Collection()	Integer	
atTime	Same as source element	<any class>	Integer	
attributes	Collection(String)	MetaType		
average	Float	Collection(Numeric)		
boldTime	Integer	<any class>		
collect	List containing elements of the same type as parameter 1	Collection()	<any>	
concat	String	String	String	
constraints	Collection(Constraint)	<any>		
count	Integer	Collection()	<any>	
dateTimeAsFloat	Float	Moment		

Name	Result	Context	Parameter 1	Parameter 2
difference	Same as source element	Collection()	Collection()	
div	Integer	Integer	Integer	
emptyList	Object list	MetaType		
excluding	Same as source element	Collection()	<any class>	
existing	Boolean	<any class>		
exists	Boolean	Collection()	Boolean	
filterOnType	List containing elements of the same type as refered by parameter 1	Collection()	MetaType	
first	Same as list elements of first argument	Collection()		
floor	Integer	Numeric		
forAll	Boolean	Collection()	Boolean	
formatDateTime	String	Moment	String	
formatNumeric	String	Numeric	String	
if	Least common supertype of parameter 1 and 2	Boolean	<any>	<any>
implies	Boolean	Boolean	Boolean	
includes	Boolean	Collection()	<any>	
includesAll	Boolean	Collection()	Collection()	
including	Least common supertype of source element and parameter 1	Collection()	<any class>	
inDateRange	Boolean	Moment	Numeric	Numeric
intersection	Least common supertype of source element and parameter 1	Collection()	Collection()	
inTimeRange	Boolean	Moment	Numeric	Numeric
isEmpty	Boolean	Collection()		
isNull	Boolean	<any>		
last	Same as list elements of first argument	Collection()		
length	Integer	String		
max	Least common supertype of source element and parameter 1	Numeric	Numeric	
maxValue	Same as list elements of first argument	Collection(Numeric)		
min	Least common supertype of source element and parameter 1	Numeric	Numeric	
minValue	Same as list elements of first argument	Collection(Numeric)		
mod	Integer	Integer	Integer	
not	Boolean	Boolean		
notEmpty	Boolean	Collection()		
objectTimeStamp	Integer	<any class>		
oclAsType	Type of parameter 1	<any>	MetaType	
oclIsKindOf	Boolean	<any>	MetaType	
oclIsTypeOf	Boolean	<any>	MetaType	
oclType	MetaType	<any>		
or	Boolean	Boolean	Boolean	
orderby	Same as source element	Collection()	<any>	
orderdescending	Same as source element	Collection()	<any>	

Name	Result	Context	Parameter 1	Parameter 2
pad	String	String	Integer	String
postPad	String	String	Integer	String
prepend	Least common supertype of source element and parameter 1	Collection()	<any class>	
regExpMatch	Boolean	String	String	
reject	Same as source element	Collection()	Boolean	
round	Integer	Numeric		
safeCast	Type of parameter 1	<any>	MetaType	
select	Same as source element	Collection()	Boolean	
size	Integer	Collection()		
sqlLike	Boolean	String	String	
sqlLikeCaseInsensitive	Boolean	String	String	
stringRepresentation	String	<any>	Integer	
strToDate	Date	String		
strToDateTime	DateTime	String		
strToInt	Integer	String		
strToTime	Time	String		
subSequence	Same as source element	Collection()	Integer	Integer
subString	String	String	Integer	Integer
sum	Same as list elements of first argument	Collection(Numeric)		
sumTime	DateTime	Collection(Moment)		
superTypes	Collection(MetaType)	MetaType		
symmetricDifference	Least common supertype of source element and parameter 1	Collection()	Collection()	
taggedValue	String	<any class>	String	
timeStampToTime	DateTime	Integer		
timeToTimeStamp	Integer	DateTime		
toLower	String	String		
toUpper	String	String		
typename	String	MetaType		
unary-	Same as source element	Numeric		
union	Least common supertype of source element and parameter 1	Collection()	Collection()	
xor	Boolean	Boolean	Boolean	

Appendix B: Application 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 an standard ASCII text editor like notepad.exe. Cut and paste the contents of each file and save the file using the correct name and extension.

The format of the listing is:

Start listing Filename

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

End listing Filename

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

OCLexplorer.dpr

Start Listing for OCLexplorer.dpr

```
program OCLexplorer;

{%File 'ContactClasses_Interface.inc'}
{%File 'ContactClasses.inc'}

uses
  Forms,
  MainFormUnit in 'MainFormUnit.pas' {OclExplorerForm},
  ContactsModel in 'ContactsModel.pas' {ContactDM: TDataModule},
  ContactClasses in 'ContactClasses.pas',
  BoldOclSymbolLister in 'BoldOclSymbolLister.pas' {OCLSyntaxForm};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TContactDM, ContactDM);
  Application.CreateForm(TOclExplorerForm, OclExplorerForm);
  Application.Run;
end.
```

End Listing OCLexplorer.dpr

MainFormUnit.pas

Start Listing for MainFormUnit.pas

```
unit MainFormUnit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ContactsModel, StdCtrls, ComCtrls, BoldListBox, BoldSubscription,
  BoldHandles, BoldRootedHandles, BoldAbstractListHandle, BoldCursorHandle,
  BoldListHandle, BoldEdit, ExtCtrls, BoldExpressionHandle,
  BoldVariableDefinition, BoldOclVariables, ActnList, Menus;

type
  TOclExplorerForm = class(TForm)
    Panel1: TPanel;
    Splitter1: TSplitter;
    Splitter2: TSplitter;
    Panel2: TPanel;
    Panel3: TPanel;
    List2EditOCLButton: TButton;
    List2OCLMemo: TMemo;
    Panel4: TPanel;
    List2ResultListBox: TBoldListBox;
    Panel5: TPanel;
    Label1: TLabel;
    Panel6: TPanel;
    List2ResultMemo: TMemo;
    Panel7: TPanel;
    Label2: TLabel;
    List1Handle: TBoldListHandle;
    Panel8: TPanel;
    Splitter3: TSplitter;
    Splitter4: TSplitter;
    Panel9: TPanel;
    Panel10: TPanel;
    List1EditOCLButton: TButton;
    List1OCLMemo: TMemo;
    Panel11: TPanel;
    List1ResultListBox: TBoldListBox;
    Panel12: TPanel;
    Label3: TLabel;
    Panel13: TPanel;
    List1ResultMemo: TMemo;
    Panel14: TPanel;
    Label4: TLabel;
    List2Handle: TBoldListHandle;
    Splitter5: TSplitter;
    BoldVariableDefinition: TBoldOclVariables;
    MainMenu1: TMainMenu;
    FileMenu: TMenuItem;
    ActionList1: TActionList;
    CloseApplicationAction: TAction;
    AboutAction: TAction;
    Close1: TMenuItem;
    ToolsMenu: TMenuItem;
    EditOCL1: TMenuItem;
    HelpMenu: TMenuItem;
    About1: TMenuItem;
    ShowDebuggerAction: TAction;
    EditList1OCL1: TMenuItem;
    EditList2OCL1: TMenuItem;
    N1: TMenuItem;
    EditList1Action: TAction;
    EditList2Action: TAction;
    ShowOCLSyntaxSummary: TAction;
    ShowOCLSummary1: TMenuItem;
    procedure List2EditOCL(Sender: TObject);
    procedure OCLUpdated(Sender: TObject);
    procedure List1EditOCL(Sender: TObject);
    procedure CloseApplicationActionExecute(Sender: TObject);
    procedure ShowDebuggerActionExecute(Sender: TObject);
    procedure AboutActionExecute(Sender: TObject);
    procedure BoldEditListActionPostExecute(Sender: TObject);
    procedure ShowOCLSyntaxSummaryExecute(Sender: TObject);
  private
    { Private declarations }
```

```
    Procedure UpdateStatus;
    procedure EditOCL(ListHandle: TBoldListHandle);
    procedure ExportBoldListToStringList(BoldList: TBoldListHandle; StringList:
        TStrings);
  public
    { Public declarations }
  end;

var
  OclExplorerForm: TOclExplorerForm;

implementation
uses BoldOclPropEditor, BoldSystemDebuggerForm, BoldOclSymbolLister;
{$R *.DFM}

procedure TOclExplorerForm.EditOCL(ListHandle: TBoldListHandle);
begin
  with TBoldOclPropEditForm.Create(Self) do
  try
    Context := ListHandle.RootHandle.BoldType;
    OclExpr := ListHandle.Expression;

    // Make sure that any variables we have added are visible to the OCL editor
    if assigned(ListHandle.Variables) then
      Variables := ListHandle.Variables.VariableList;

    if ShowModal = mrOK then
      ListHandle.Expression := OCLExpr;
  finally
    Free;
  end;
end;

procedure TOclExplorerForm.List2EditOCL(Sender: TObject);
begin
  EditOCL(List2Handle);
  UpdateStatus;
end;

procedure TOclExplorerForm.OCLUpdated(Sender: TObject);
begin
  UpdateStatus;
end;

procedure TOclExplorerForm.UpdateStatus;
var Counter: Integer;
begin
  List1OCLMemo.Text := AdjustLineBreaks(List1Handle.Expression);
  ExportBoldListToStringList(List1Handle, List1ResultMemo.Lines);

  List2OCLMemo.Text := AdjustLineBreaks(List2Handle.Expression);
  ExportBoldListToStringList(List2Handle, List2ResultMemo.Lines);
end;

procedure TOclExplorerForm.List1EditOCL(Sender: TObject);
begin
  EditOCL(List1Handle);
  UpdateStatus;
end;

procedure TOclExplorerForm.ExportBoldListToStringList(BoldList: TBoldListHandle;
    StringList: TStrings);
var Counter: Integer;
begin
  StringList.BeginUpdate;
  try
    StringList.Clear;
    StringList.Add('Type Information');
    StringList.Add('----------------');
    StringList.Add('List Type (AsString)    = ' + BoldList.ListType.AsString);
    StringList.Add('List Type (DelphiClass) = ' + BoldList.ListType.ClassName);
    StringList.Add('Bold Type (AsString)    = ' + BoldList.BoldType.AsString);
    StringList.Add('Bold Type (DelphiClass) = ' + BoldList.BoldType.ClassName);
    if Assigned(BoldList.List) then
      StringList.Add('List Type (DelphiClass) = ' + BoldList.List.ClassName)
    else
      StringList.Add('List Type (DelphiClass) = nil');

    StringList.Add('');
    StringList.Add('Content Information');
```

```
    StringList.Add('-------------------');
    StringList.Add('Result Count = ' + IntToStr(BoldList.Count));
    if Assigned(BoldList.Value) then
    begin
      StringList.Add('Value (AsString)    = ' + BoldList.Value.AsString);
      StringList.Add('Value (DelphiClass) = ' + BoldList.Value.ClassName);
    end
    else
      StringList.Add('Value = nil');

    if assigned(BoldList.List) then
    begin
      StringList.Add('');
      StringList.Add('List Contents');
      StringList.Add('-------------------');
      for counter := 0 to  BoldList.List.Count -1 do
        StringList.Add(BoldList.List[counter].AsString);
    end;
  finally
    StringList.EndUpdate;
  end;
end;


procedure TOclExplorerForm.CloseApplicationActionExecute(Sender: TObject);
begin
  close;
end;

procedure TOclExplorerForm.ShowDebuggerActionExecute(Sender: TObject);
begin
  with TBoldSystemDebuggerFrm.CreateWithSystem(application,
    ContactDM.BoldSystemHandle1.System) do
      show;
end;

procedure TOclExplorerForm.AboutActionExecute(Sender: TObject);
begin
  MessageDlg('For more information on Bold for Delphi goto'+#13+#10
            +'http://www.boldsoft.com'+#13+#10+''+#13+#10
            +'For more information about OCL Explorer goto'+#13+#10
            +'http://www.viewpointsa.com', mtInformation, [mbOK], 0);
end;

procedure TOclExplorerForm.BoldEditListActionPostExecute(Sender: TObject);
begin
  UpdateStatus;
end;

procedure TOclExplorerForm.ShowOCLSyntaxSummaryExecute(Sender: TObject);
begin
  TOCLSyntaxForm.Create(nil);
end;

end.
```

End Listing for MainFormUnit.pas

MainFormUnit.dfm

Start Listing for MainFormUnit.dfm

```
object OclExplorerForm: TOclExplorerForm
  Left = 309
  Top = 250
  Width = 989
  Height = 524
  Caption = 'OCL Explorer'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  Menu = MainMenu1
  OldCreateOrder = False
  Position = poScreenCenter
  OnCreate = OCLUpdated
  PixelsPerInch = 96
  TextHeight = 13
  object Splitter5: TSplitter
    Left = 497
    Top = 0
    Width = 4
    Height = 478
    Cursor = crHSplit
  end
  object Panel1: TPanel
    Left = 501
    Top = 0
    Width = 480
    Height = 478
    Align = alClient
    Caption = 'Panel1'
    TabOrder = 0
    object Splitter1: TSplitter
      Left = 1
      Top = 137
      Width = 478
      Height = 4
      Cursor = crVSplit
      Align = alTop
    end
    object Splitter2: TSplitter
      Left = 202
      Top = 141
      Width = 4
      Height = 336
      Cursor = crHSplit
    end
    object Panel2: TPanel
      Left = 1
      Top = 1
      Width = 478
      Height = 136
      Align = alTop
      BevelOuter = bvNone
      BorderWidth = 5
      TabOrder = 0
      object Panel3: TPanel
        Left = 5
        Top = 5
        Width = 468
        Height = 28
        Align = alTop
        BevelOuter = bvNone
        TabOrder = 0
        object List2EditOCLButton: TButton
          Left = 0
          Top = 0
          Width = 81
          Height = 25
          Action = EditList2Action
          TabOrder = 0
        end
      end
      object List2OCLMemo: TMemo
```

```
        Left = 5
        Top = 33
        Width = 468
        Height = 98
        Align = alClient
        Color = clBtnFace
        ReadOnly = True
        TabOrder = 1
      end
    end
    object Panel4: TPanel
      Left = 1
      Top = 141
      Width = 201
      Height = 336
      Align = alLeft
      BevelOuter = bvNone
      TabOrder = 1
      object List2ResultListBox: TBoldListBox
        Left = 0
        Top = 25
        Width = 201
        Height = 311
        Align = alClient
        Alignment = taLeftJustify
        BoldHandle = List2Handle
        BoldProperties.NilElementMode = neNone
        DragMode = dmAutomatic
        ItemHeight = 16
        TabOrder = 0
        OnClick = OCLUpdated
      end
      object Panel5: TPanel
        Left = 0
        Top = 0
        Width = 201
        Height = 25
        Align = alTop
        BevelOuter = bvNone
        TabOrder = 1
        object Label1: TLabel
          Left = 4
          Top = 7
          Width = 25
          Height = 13
          Caption = 'List 2'
        end
      end
    end
    object Panel6: TPanel
      Left = 206
      Top = 141
      Width = 273
      Height = 336
      Align = alClient
      BevelOuter = bvNone
      TabOrder = 2
      object List2ResultMemo: TMemo
        Left = 0
        Top = 25
        Width = 273
        Height = 311
        Align = alClient
        Color = clBtnFace
        Font.Charset = ANSI_CHARSET
        Font.Color = clWindowText
        Font.Height = -11
        Font.Name = 'Courier New'
        Font.Style = []
        ParentFont = False
        ReadOnly = True
        ScrollBars = ssBoth
        TabOrder = 0
        WordWrap = False
      end
      object Panel7: TPanel
        Left = 0
        Top = 0
        Width = 273
        Height = 25
```

```
        Align = alTop
        BevelOuter = bvNone
        TabOrder = 1
        object Label2: TLabel
          Left = 4
          Top = 7
          Width = 35
          Height = 13
          Caption = 'Results'
        end
      end
    end
  end
  object Panel8: TPanel
    Left = 0
    Top = 0
    Width = 497
    Height = 478
    Align = alLeft
    Caption = 'Panel1'
    TabOrder = 1
    object Splitter3: TSplitter
      Left = 1
      Top = 137
      Width = 495
      Height = 4
      Cursor = crVSplit
      Align = alTop
    end
    object Splitter4: TSplitter
      Left = 202
      Top = 141
      Width = 4
      Height = 336
      Cursor = crHSplit
    end
    object Panel9: TPanel
      Left = 1
      Top = 1
      Width = 495
      Height = 136
      Align = alTop
      BevelOuter = bvNone
      BorderWidth = 5
      TabOrder = 0
      object Panel10: TPanel
        Left = 5
        Top = 5
        Width = 485
        Height = 28
        Align = alTop
        BevelOuter = bvNone
        TabOrder = 0
        object List1EditOCLButton: TButton
          Left = 0
          Top = 0
          Width = 81
          Height = 25
          Action = EditList1Action
          TabOrder = 0
        end
      end
      object List1OCLMemo: TMemo
        Left = 5
        Top = 33
        Width = 485
        Height = 98
        Align = alClient
        Color = clBtnFace
        ReadOnly = True
        TabOrder = 1
      end
    end
    object Panel11: TPanel
      Left = 1
      Top = 141
      Width = 201
      Height = 336
      Align = alLeft
      BevelOuter = bvNone
```

```
      TabOrder = 1
      object List1ResultListBox: TBoldListBox
        Left = 0
        Top = 25
        Width = 201
        Height = 311
        Align = alClient
        Alignment = taLeftJustify
        BoldHandle = List1Handle
        BoldProperties.NilElementMode = neNone
        DragMode = dmAutomatic
        ItemHeight = 16
        TabOrder = 0
        OnClick = OCLUpdated
      end
      object Panel12: TPanel
        Left = 0
        Top = 0
        Width = 201
        Height = 25
        Align = alTop
        BevelOuter = bvNone
        TabOrder = 1
        object Label3: TLabel
          Left = 4
          Top = 7
          Width = 25
          Height = 13
          Caption = 'List 1'
        end
      end
    end
    object Panel13: TPanel
      Left = 206
      Top = 141
      Width = 290
      Height = 336
      Align = alClient
      BevelOuter = bvNone
      TabOrder = 2
      object List1ResultMemo: TMemo
        Left = 0
        Top = 25
        Width = 290
        Height = 311
        Align = alClient
        Color = clBtnFace
        Font.Charset = ANSI_CHARSET
        Font.Color = clWindowText
        Font.Height = -11
        Font.Name = 'Courier New'
        Font.Style = []
        ParentFont = False
        ReadOnly = True
        ScrollBars = ssBoth
        TabOrder = 0
        WordWrap = False
      end
      object Panel14: TPanel
        Left = 0
        Top = 0
        Width = 290
        Height = 25
        Align = alTop
        BevelOuter = bvNone
        TabOrder = 1
        object Label4: TLabel
          Left = 4
          Top = 7
          Width = 35
          Height = 13
          Caption = 'Results'
        end
      end
    end
  end
  object List1Handle: TBoldListHandle
    RootHandle = ContactDM.BoldSystemHandle1
    Left = 12
    Top = 104
```

```
  end
  object List2Handle: TBoldListHandle
    RootHandle = List1Handle
    Variables = BoldVariableDefinition
    Left = 516
    Top = 104
  end
  object BoldVariableDefinition: TBoldOclVariables
    Variables = <
      item
        BoldHandle = List1Handle
        VariableName = 'list'
        UseListElement = True
      end>
    Left = 49
    Top = 105
  end
  object MainMenu1: TMainMenu
    Left = 88
    Top = 104
    object FileMenu: TMenuItem
      Caption = '&File'
      object EditList1OCL1: TMenuItem
        Action = EditList1Action
      end
      object EditList2OCL1: TMenuItem
        Action = EditList2Action
      end
      object N1: TMenuItem
        Caption = '-'
      end
      object Close1: TMenuItem
        Action = CloseApplicationAction
      end
    end
    object ToolsMenu: TMenuItem
      Caption = '&Tools'
      object EditOCL1: TMenuItem
        Action = ShowDebuggerAction
      end
      object ShowOCLSummary1: TMenuItem
        Action = ShowOCLSyntaxSummary
      end
    end
    object HelpMenu: TMenuItem
      Caption = '&Help'
      object About1: TMenuItem
        Action = AboutAction
      end
    end
  end
  object ActionList1: TActionList
    Left = 128
    Top = 104
    object CloseApplicationAction: TAction
      Caption = '&Close'
      OnExecute = CloseApplicationActionExecute
    end
    object AboutAction: TAction
      Caption = '&About'
      OnExecute = AboutActionExecute
    end
    object ShowDebuggerAction: TAction
      Caption = 'Show &Debugger'
      OnExecute = ShowDebuggerActionExecute
    end
    object EditList1Action: TAction
      Caption = 'Edit List &1 OCL'
      OnExecute = List1EditOCL
    end
    object EditList2Action: TAction
      Caption = 'Edit List &2 OCL'
      OnExecute = List2EditOCL
    end
    object ShowOCLSyntaxSummary: TAction
      Caption = 'Show &OCL Syntax Summary'
      OnExecute = ShowOCLSyntaxSummaryExecute
    end
  end
end
```

End Listing for MainFormUnit.dfm

BoldOCLSymbolLister.pas

Start Listing for BoldOCLSymbolLister.pas

```
unit BoldOclSymbolLister;

interface

uses
  BoldOclClasses, BoldOclSymbolImplementations,
  Clipbrd,
  BoldSystemRT, BoldElements,
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, BoldHandles, BoldSystemHandle, BoldSubscription,
  BoldAbstractModel, BoldModel, Grids;

type
  TOCLSyntaxForm = class(TForm)
    BoldModel1: TBoldModel;
    BoldSystemTypeInfoHandle1: TBoldSystemTypeInfoHandle;
    sgSymbols: TStringGrid;
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

implementation

{$R *.dfm}

procedure TOCLSyntaxForm.FormCreate(Sender: TObject);
var
  SymTab: TBoldSymbolDictionary;
  Symbol: TBoldOclSymbol;
  i, j: integer;
  errors: boolean;
  temp: string;

  function GetTypeName(aType: TBoldElementTypeInfo): string;
  begin
    if assigned(aType) then
    begin
      if (aType is TBoldClassTypeInfo) and
        not assigned((aType as TBoldClassTypeInfo).SuperClassTypeInfo) then
        Result := '<any class>'
      else
        Result := aType.ExpressionName
    end
    else
      result := '<any>';
  end;

begin
  SymTab := TBoldSymbolDictionary.Create(
       BoldSystemTypeInfoHandle1.StaticSystemTypeInfo, nil, errors);

  sgSymbols.Cells[0, 0] := 'Name';
  sgSymbols.Cells[1, 0] := 'Result';
  sgSymbols.Cells[2, 0] := 'Context';
  sgSymbols.Cells[3, 0] := 'Parameter 1';
  sgSymbols.Cells[4, 0] := 'Parameter 2';
  sgSymbols.DefaultColWidth := 80;
  sgSymbols.ColWidths[1] := 400;
  width := 4 * 80 + 400 + 40;

  InitializeSymbolTable(symtab);
  sgSymbols.RowCount := SymTab.Count + 1;
  for i := 0 to SymTab.Count - 1 do
  begin
    Symbol := SymTab.Symbols[i];
    sgSymbols.Cells[0, i + 1] := Symbol.SymbolName;
    case Symbol.DeduceMethod of
      tbodNo: temp := GetTypeName(symbol.ResultType);
```

```
      tbodCopyLoopVar: Temp := 'Type of loop variable';
      tbodCopyArg1: Temp := 'Same as source element';
      tbodCopyArg1Elem: Temp := 'Same as list elements of first argument';
      tbodCopyArg2: Temp := 'Same as parameter 1';
      tbodCopyArg3: Temp := 'Same as parameter 2';
      tbodLCC: Temp := 'Least common supertype of source element and parameter 1';
      tbodLCC23: Temp := 'Least common supertype of parameter 1 and 2';
      tbodListofArg2: Temp := 'List containing elements of the same type as parameter 1';
      tbodObjectlist: Temp := 'Object list';
      tbodType: Temp := 'Meta type';
      tbodTypecast: Temp := 'Type of parameter 1';
      tbodArg1AsList: Temp := 'Same as source element';
      tbodListFromArg2: Temp := 'List containing elements of the same type as refered by
parameter 1';
    end;
    sgSymbols.Cells[1, i + 1] := Temp;

    if sgSymbols.ColCount < Symbol.NumberOfArgs + 2 then
      sgSymbols.ColCount := Symbol.NumberOfArgs + 2;

    for j := 0 to Symbol.NumberOfArgs - 1 do
      sgSymbols.Cells[j + 2, i + 1] := GetTypeName(Symbol.FormalArguments[j]);
  end;
  SymTab.Free;
end;

procedure TOCLSyntaxForm.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
  action := caFree;
end;

end.
```

End Listing BoldOCLSymbolLister.pas

BoldOCLSymbolLister.dfm

Start Listing for BoldOCLSymbolLister.dfm

```
object OCLSyntaxForm: TOCLSyntaxForm
  Left = 454
  Top = 310
  Width = 625
  Height = 282
  Caption = 'OCL Syntax Summary'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  Position = poDefault
  Visible = True
  OnClose = FormClose
  OnCreate = FormCreate
  PixelsPerInch = 96
  TextHeight = 13
  object sgSymbols: TStringGrid
    Left = 0
    Top = 0
    Width = 617
    Height = 255
    Align = alClient
    DefaultColWidth = 80
    DefaultRowHeight = 17
    FixedCols = 0
    Options = [goFixedVertLine, goFixedHorzLine, goVertLine, goHorzLine, goRangeSelect,
goColSizing, goRowSelect]
    TabOrder = 0
  end
  object BoldModel1: TBoldModel
    UMLModelMode = ummNone
    Boldify.EnforceDefaultUMLCase = False
    Boldify.DefaultNavigableMultiplicity = '0..1'
    Boldify.DefaultNonNavigableMultiplicity = '0..*'
    Left = 16
    Top = 8
    Model = (
      'VERSION 19'
      '(Model'
      #9'"New_Model"'
      #9'"New_ModelRoot"'
      #9'""'
      #9'""'
      #9'"Bold.DelphiName=<Name>"'
      #9'(Classes'
      #9#9'(Class'
      #9#9#9'"New_ModelRoot"'
      #9#9#9'"<NONE>"'
      #9#9#9'TRUE'
      #9#9#9'FALSE'
      #9#9#9'""'
      #9#9#9'""'
      #9#9#9'""'
      #9#9#9'(Attributes'
      #9#9#9')'
      #9#9#9'(Methods'
      #9#9#9')'
      #9#9')'
      #9')'
      #9'(Associations'
      #9')'
      ')')
  end
  object BoldSystemTypeInfoHandle1: TBoldSystemTypeInfoHandle
    BoldModel = BoldModel1
    Left = 80
    Top = 8
  end
end
```

End Listing for BoldOCLSymbolLister.dfm

ContactModel.pas

Start Listing for ContactModel.pas

```
unit ContactsModel;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  BoldHandle, BoldPersistenceHandle, BoldPersistenceHandleFile,
  BoldPersistenceHandleFileXML, BoldHandles, BoldSystemHandle,
  BoldSubscription, BoldAbstractModel, BoldModel, BoldUMLModelLink,
  BoldUMLRose98Link, BoldPersistenceHandleDB,
  BoldPersistenceHandleIB;

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

var
  ContactDM: TContactDM;

implementation

{$R *.DFM}

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

procedure TContactDM.DataModuleCreate(Sender: TObject);
begin
  BoldSystemHandle1.Active := True;
end;

end.
```

End Listing for ContactModel.pas

ContactModel.dfm

Start Listing for ContactModel.dfm

```
object ContactDM: TContactDM
  OldCreateOrder = False
  OnCreate = DataModuleCreate
  OnDestroy = DataModuleDestroy
  Left = 391
  Top = 286
  Height = 303
  Width = 227
  object BoldModel1: TBoldModel
    UMLModelMode = ummNone
    Boldify.EnforceDefaultUMLCase = False
    Boldify.DefaultNavigableMultiplicity = '0..1'
    Boldify.DefaultNonNavigableMultiplicity = '0..*'
    Left = 76
    Top = 12
    Model = (
      'VERSION 19'
      '(Model'
      #9'"BusinessClasses"'
      #9'"ContactRoot"'
      #9'"""'
      #9'"""'

        #9'"_Boldify.boldified=True,_BoldInternal.flattened=True,_BoldInte' +
        'rnal.toolId=3C4BD831036F,_BoldInternal.ModelErrors=,Bold.DelphiN' +
        'ame=<Name>,Bold.UnitName=ContactClasses,Bold.RootClass=ContactRo' +
        'ot"'
      #9'(Classes'
      #9#9'(Class'
      #9#9#9'"ContactRoot"'
      #9#9#9'"<NONE>"'
      #9#9#9'TRUE'
      #9#9#9'TRUE'
      #9#9#9'""'
      #9#9#9'""'
      #9#9#9'"_BoldInternal.toolId=3C4C11DD02D9,persistence=persistent"'
      #9#9#9'(Attributes'
      #9#9#9')'
      #9#9#9'(Methods'
      #9#9#9')'
      #9#9')'
      #9#9'(Class'
      #9#9#9'"Contact"'
      #9#9#9'"ContactRoot"'
      #9#9#9'TRUE'
      #9#9#9'TRUE'
      #9#9#9'""'
      #9#9#9'""'

        #9#9#9'"_BoldInternal.toolId=3C4BD87801D7,persistence=persistent,\"B' +
        'old.DefaultStringRepresentation=''Contact - '' + name\""'
      #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'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'"_BoldInternal.toolId=3C4BD94B0393,persistence=transient,de' +
        'rived=True,Bold.ReverseDerive=True"'
      #9#9#9#9')'
      #9#9#9')'
      #9#9#9'(Methods'
      #9#9#9')'
      #9#9')'
      #9#9'(Class'
      #9#9#9'"Person"'
      #9#9#9'"Contact"'
      #9#9#9'TRUE'
      #9#9#9'FALSE'
      #9#9#9'""'
      #9#9#9'"""
```

```
  #9#9#9'"_BoldInternal.toolId=3C4BD8D6004B,persistence=persistent,\"B' +
  'old.DefaultStringRepresentation=if maidenName = '''' then\c  ''Pers' +
  'on - '' + name\celse\c  ''Person - '' + name + '' (nee '' + maidenNam' +
  'e + '')''\cendif\""'
#9#9#9'(Attributes'
#9#9#9#9'(Attribute'
#9#9#9#9#9'"dateOfBirth"'
#9#9#9#9#9'"Date"'
#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'"_BoldInternal.toolId=3C4BD9BF034A,persistence=Persistent"'
#9#9#9#9')'
#9#9#9#9'(Attribute'
#9#9#9#9#9'"married"'
#9#9#9#9#9'"Boolean"'
#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'"derived=False,persistence=persistent"'
#9#9#9#9')'
#9#9#9#9'(Attribute'
#9#9#9#9#9'"maidenName"'
#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'"derived=False,persistence=persistent"'
#9#9#9#9')'
#9#9#9#9'(Attribute'
#9#9#9#9#9'"firstName"'
#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'"derived=False,persistence=persistent"'
#9#9#9#9')'
#9#9#9#9'(Attribute'
#9#9#9#9#9'"lastName"'
#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'"derived=False,persistence=persistent"'
#9#9#9#9')'
#9#9#9')'
#9#9#9'(Methods'
#9#9#9')'
#9#9')'
#9#9'(Class'
#9#9#9'"Company"'
#9#9#9'"Contact"'
#9#9#9'TRUE'
#9#9#9'FALSE'
#9#9#9'""'
#9#9#9'""'

  #9#9#9'"_BoldInternal.toolId=3C4BD8DD025E,persistence=persistent,\"B' +
  'old.DefaultStringRepresentation=''Company - '' + name\""'
#9#9#9'(Attributes'
#9#9#9#9'(Attribute'
#9#9#9#9#9'"companyName"'
#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'"derived=False,persistence=persistent"'
```

```
      #9#9#9#9')'
      #9#9#9')'
      #9#9#9'(Methods'
      #9#9#9')'
      #9#9')'
      #9#9'(Class'
      #9#9#9'"Category"'
      #9#9#9'"ContactRoot"'
      #9#9#9'TRUE'
      #9#9#9'FALSE'
      #9#9#9'""'
      #9#9#9'""'

        #9#9#9'"_BoldInternal.toolId=3C4BD8EC036E,persistence=persistent,\"B' +
        'old.DefaultStringRepresentation=''Category - '' + name\""'
      #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'"_BoldInternal.toolId=3C4BD9660048,persistence=Persistent"'
      #9#9#9#9')'
      #9#9#9')'
      #9#9#9'(Methods'
      #9#9#9')'
      #9#9')'
      #9#9'(Class'
      #9#9#9'"ContactCategory"'
      #9#9#9'"ContactRoot"'
      #9#9#9'TRUE'
      #9#9#9'FALSE'
      #9#9#9'""'
      #9#9#9'""'

        #9#9#9'"_Boldify.autoCreated=True,persistence=persistent,\"Bold.Defa' +
        'ultStringRepresentation=contacts.name.asString + '' is a '' + cate' +
        'gories.name.asString + '' contact''\""'
      #9#9#9'(Attributes'
      #9#9#9')'
      #9#9#9'(Methods'
      #9#9#9')'
      #9#9')'
      #9#9'(Class'
      #9#9#9'"Employment"'
      #9#9#9'"ContactRoot"'
      #9#9#9'TRUE'
      #9#9#9'FALSE'
      #9#9#9'""'
      #9#9#9'""'

        #9#9#9'"_Boldify.autoCreated=True,persistence=persistent,\"Bold.Defa' +
        'ultStringRepresentation=employees.name.asString + '' works for th' +
        'e '' + companies.name.asString + '' company''\""'
      #9#9#9'(Attributes'
      #9#9#9')'
      #9#9#9'(Methods'
      #9#9#9')'
      #9#9')'
      #9')'
      #9'(Associations'
      #9#9'(Association'
      #9#9#9'"ContactCategory"'
      #9#9#9'"ContactCategory"'
      #9#9#9'""'
      #9#9#9'""'

        #9#9#9'"persistence=Persistent,_BoldInternal.toolId=3C4BD90C0306,Bol' +
        'd.DelphiName=<Name>"'
      #9#9#9'FALSE'
      #9#9#9'(Roles'
      #9#9#9#9'(Role'
      #9#9#9#9#9'"categories"'
      #9#9#9#9#9'TRUE'
      #9#9#9#9#9'FALSE'
      #9#9#9#9#9'"Contact"'
      #9#9#9#9#9'""'
```

```
      #9#9#9#9#9'"n"'
      #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'"_BoldInternal.toolId=3C4BD90D03D0,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'"contacts"'
      #9#9#9#9#9'TRUE'
      #9#9#9#9#9'FALSE'
      #9#9#9#9#9'"Category"'
      #9#9#9#9#9'"""'
      #9#9#9#9#9'"n"'
      #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'"_BoldInternal.toolId=3C4BD90D03D2,Bold.Embed=False"'
      #9#9#9#9#9'(Qualifiers'
      #9#9#9#9#9')'
      #9#9#9#9')'
      #9#9#9')'
      #9#9')'
      #9#9'(Association'
      #9#9#9'"Employment"'
      #9#9#9'"Employment"'
      #9#9#9'"""'
      #9#9#9'"""'

        #9#9#9'"persistence=Persistent,_BoldInternal.toolId=3C4BDFA9031E,Bol' +
        'd.DelphiName=<Name>"'
      #9#9#9'FALSE'
      #9#9#9'(Roles'
      #9#9#9#9'(Role'
      #9#9#9#9#9'"companies"'
      #9#9#9#9#9'TRUE'
      #9#9#9#9#9'FALSE'
      #9#9#9#9#9'"Person"'
      #9#9#9#9#9'"""'
      #9#9#9#9#9'"n"'
      #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'"_BoldInternal.toolId=3C4BDFAA0230,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'"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'"n"'
      #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'"_BoldInternal.toolId=3C4BDFAA023A,Bold.Embed=False"'
      #9#9#9#9#9'(Qualifiers'
      #9#9#9#9#9')'
      #9#9#9#9')'
      #9#9#9')'
      #9#9')'
      #9#9'(Association'
      #9#9#9'"PrimaryContact"'
      #9#9#9'"<NONE>"'
      #9#9#9'"""'
      #9#9#9'"""'
      #9#9#9'"persistence=persistent,derived=False,Bold.DelphiName=<Name>"'
      #9#9#9'FALSE'
      #9#9#9'(Roles'
      #9#9#9#9'(Role'
      #9#9#9#9#9'"AssociationEnd"'
      #9#9#9#9#9'FALSE'
      #9#9#9#9#9'FALSE'
```

```
      #9#9#9#9#9'"Person"'
      #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'(Role'
      #9#9#9#9#9'"primaryContact"'
      #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'"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')'
      ')')
  end
  object BoldSystemTypeInfoHandle1: TBoldSystemTypeInfoHandle
    BoldModel = BoldModel1
    Left = 76
    Top = 67
  end
  object BoldSystemHandle1: TBoldSystemHandle
    IsDefault = True
    SystemTypeInfoHandle = BoldSystemTypeInfoHandle1
    Active = False
    PersistenceHandle = BoldPersistenceHandleFileXML1
    Left = 76
    Top = 137
  end
  object BoldPersistenceHandleFileXML1: TBoldPersistenceHandleFileXML
    FileName = 'data.xml'
    CacheData = False
    BoldModel = BoldModel1
    Left = 80
    Top = 200
  end
end
```

End Listing for ContactModel.dfm

ContactClasses.pas
Start Listing for ContactClasses.pas

```
(*****************************************)
(*      This file is autogenerated       *)
(*   Any manual changes will be LOST!    *)
(*****************************************)
(* Generated 9/08/2002 6:46:42 PM        *)
(*****************************************)
(* This file should be stored in the     *)
(* same directory as the form/datamodule *)
(* with the corresponding model          *)
(*****************************************)
(* Copyright notice:                     *)
(*                                       *)
(*****************************************)

unit ContactClasses;

{$DEFINE ContactClasses_unitheader}
{$INCLUDE ContactClasses_Interface.inc}

{ Includefile for methodimplementations }

{$INCLUDE ContactClasses.inc}

const
  BoldMemberAssertInvalidObjectType: string = 'Object of singlelink (%s.%s) is of wrong type
(is %s, should be %s)';

{ TContactRoot }

procedure TContactRootList.Add(NewObject: TContactRoot);
begin
  if Assigned(NewObject) then
    AddElement(NewObject);
end;

function TContactRootList.IndexOf(anObject: TContactRoot): Integer;
begin
  result := IndexOfElement(anObject);
end;

function TContactRootList.Includes(anObject: TContactRoot) : Boolean;
begin
  result := IncludesElement(anObject);
end;

function TContactRootList.AddNew: TContactRoot;
begin
  result := TContactRoot(InternalAddNew);
end;

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

function TContactRootList.GetBoldObject(index: Integer): TContactRoot;
begin
  result := TContactRoot(GetElement(index));
end;

procedure TContactRootList.SetBoldObject(index: Integer; NewObject: TContactRoot);
begin;
  SetElement(index, NewObject);
end;

{ TCategory }

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

function TCategory._Getname: String;
begin
  Result := M_name.AsString;
```

```
end;

procedure TCategory._Setname(const NewValue: String);
begin
  M_name.AsString := NewValue;
end;

function TCategory._Getcontacts: TContactList;
begin
  assert(ValidateMember('TCategory', 'contacts', 1, TContactList));
  Result := TContactList(BoldMembers[1]);
end;

function TCategory._GetContactCategory: TContactCategoryList;
begin
  assert(ValidateMember('TCategory', 'ContactCategory', 2, TContactCategoryList));
  Result := TContactCategoryList(BoldMembers[2]);
end;

procedure TCategoryList.Add(NewObject: TCategory);
begin
  if Assigned(NewObject) then
    AddElement(NewObject);
end;

function TCategoryList.IndexOf(anObject: TCategory): Integer;
begin
  result := IndexOfElement(anObject);
end;

function TCategoryList.Includes(anObject: TCategory) : Boolean;
begin
  result := IncludesElement(anObject);
end;

function TCategoryList.AddNew: TCategory;
begin
  result := TCategory(InternalAddNew);
end;

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

function TCategoryList.GetBoldObject(index: Integer): TCategory;
begin
  result := TCategory(GetElement(index));
end;

procedure TCategoryList.SetBoldObject(index: Integer; NewObject: TCategory);
begin;
  SetElement(index, NewObject);
end;

{ TContact }

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

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

procedure TContact._Setname(const NewValue: String);
begin
  M_name.AsString := NewValue;
end;

function TContact._Getcategories: TCategoryList;
begin
  assert(ValidateMember('TContact', 'categories', 1, TCategoryList));
  Result := TCategoryList(BoldMembers[1]);
end;
```

```
function TContact._GetContactCategory: TContactCategoryList;
begin
  assert(ValidateMember('TContact', 'ContactCategory', 2, TContactCategoryList));
  Result := TContactCategoryList(BoldMembers[2]);
end;

procedure TContactList.Add(NewObject: TContact);
begin
  if Assigned(NewObject) then
    AddElement(NewObject);
end;

function TContactList.IndexOf(anObject: TContact): Integer;
begin
  result := IndexOfElement(anObject);
end;

function TContactList.Includes(anObject: TContact) : Boolean;
begin
  result := IncludesElement(anObject);
end;

function TContactList.AddNew: TContact;
begin
  result := TContact(InternalAddNew);
end;

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

function TContactList.GetBoldObject(index: Integer): TContact;
begin
  result := TContact(GetElement(index));
end;

procedure TContactList.SetBoldObject(index: Integer; NewObject: TContact);
begin;
  SetElement(index, NewObject);
end;

function TContact.GetDeriveMethodForMember(Member: TBoldMember): TBoldDeriveAndResubscribe;
begin
  if (Member = M_name) then result := _name_DeriveAndSubscribe else
  result := inherited GetDeriveMethodForMember(Member);
end;

function TContact.GetReverseDeriveMethodForMember(Member: TBoldMember): TBoldReverseDerive;
begin
  result := inherited GetReverseDeriveMethodForMember(Member);
  if not assigned(result) and (Member = M_name) then result := _name_ReverseDerive;
end;

{ TContactCategory }

function TContactCategory._Get_M_contacts: TBoldObjectReference;
begin
  assert(ValidateMember('TContactCategory', 'contacts', 0, TBoldObjectReference));
  Result := TBoldObjectReference(BoldMembers[0]);
end;

function TContactCategory._Getcontacts: TContact;
begin
  assert(not assigned(M_contacts.BoldObject) or (M_contacts.BoldObject is TContact),
SysUtils.format(BoldMemberAssertInvalidObjectType, [ClassName, 'contacts',
M_contacts.BoldObject.ClassName, 'TContact']));
  Result := TContact(M_contacts.BoldObject);
end;

function TContactCategory._Get_M_categories: TBoldObjectReference;
begin
  assert(ValidateMember('TContactCategory', 'categories', 1, TBoldObjectReference));
  Result := TBoldObjectReference(BoldMembers[1]);
end;

function TContactCategory._Getcategories: TCategory;
begin
```

```
  assert(not assigned(M_categories.BoldObject) or (M_categories.BoldObject is TCategory),
SysUtils.format(BoldMemberAssertInvalidObjectType, [ClassName, 'categories',
M_categories.BoldObject.ClassName, 'TCategory']));
  Result := TCategory(M_categories.BoldObject);
end;

procedure TContactCategoryList.Add(NewObject: TContactCategory);
begin
  if Assigned(NewObject) then
    AddElement(NewObject);
end;

function TContactCategoryList.IndexOf(anObject: TContactCategory): Integer;
begin
  result := IndexOfElement(anObject);
end;

function TContactCategoryList.Includes(anObject: TContactCategory) : Boolean;
begin
  result := IncludesElement(anObject);
end;

function TContactCategoryList.AddNew: TContactCategory;
begin
  result := TContactCategory(InternalAddNew);
end;

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

function TContactCategoryList.GetBoldObject(index: Integer): TContactCategory;
begin
  result := TContactCategory(GetElement(index));
end;

procedure TContactCategoryList.SetBoldObject(index: Integer; NewObject: TContactCategory);
begin;
  SetElement(index, NewObject);
end;

{ TEmployment }

function TEmployment._Get_M_employees: TBoldObjectReference;
begin
  assert(ValidateMember('TEmployment', 'employees', 0, TBoldObjectReference));
  Result := TBoldObjectReference(BoldMembers[0]);
end;

function TEmployment._Getemployees: TPerson;
begin
  assert(not assigned(M_employees.BoldObject) or (M_employees.BoldObject is TPerson),
SysUtils.format(BoldMemberAssertInvalidObjectType, [ClassName, 'employees',
M_employees.BoldObject.ClassName, 'TPerson']));
  Result := TPerson(M_employees.BoldObject);
end;

function TEmployment._Get_M_companies: TBoldObjectReference;
begin
  assert(ValidateMember('TEmployment', 'companies', 1, TBoldObjectReference));
  Result := TBoldObjectReference(BoldMembers[1]);
end;

function TEmployment._Getcompanies: TCompany;
begin
  assert(not assigned(M_companies.BoldObject) or (M_companies.BoldObject is TCompany),
SysUtils.format(BoldMemberAssertInvalidObjectType, [ClassName, 'companies',
M_companies.BoldObject.ClassName, 'TCompany']));
  Result := TCompany(M_companies.BoldObject);
end;

procedure TEmploymentList.Add(NewObject: TEmployment);
begin
  if Assigned(NewObject) then
    AddElement(NewObject);
end;

function TEmploymentList.IndexOf(anObject: TEmployment): Integer;
```

```
begin
  result := IndexOfElement(anObject);
end;

function TEmploymentList.Includes(anObject: TEmployment) : Boolean;
begin
  result := IncludesElement(anObject);
end;

function TEmploymentList.AddNew: TEmployment;
begin
  result := TEmployment(InternalAddNew);
end;

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

function TEmploymentList.GetBoldObject(index: Integer): TEmployment;
begin
  result := TEmployment(GetElement(index));
end;

procedure TEmploymentList.SetBoldObject(index: Integer; NewObject: TEmployment);
begin;
  SetElement(index, NewObject);
end;

{ TCompany }

function TCompany._Get_M_companyName: TBAString;
begin
  assert(ValidateMember('TCompany', 'companyName', 3, TBAString));
  Result := TBAString(BoldMembers[3]);
end;

function TCompany._GetcompanyName: String;
begin
  Result := M_companyName.AsString;
end;

procedure TCompany._SetcompanyName(const NewValue: String);
begin
  M_companyName.AsString := NewValue;
end;

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

function TCompany._GetEmployment: TEmploymentList;
begin
  assert(ValidateMember('TCompany', 'Employment', 5, TEmploymentList));
  Result := TEmploymentList(BoldMembers[5]);
end;

function TCompany._Get_M_primaryContact: TBoldObjectReference;
begin
  assert(ValidateMember('TCompany', 'primaryContact', 6, TBoldObjectReference));
  Result := TBoldObjectReference(BoldMembers[6]);
end;

function TCompany._GetprimaryContact: TPerson;
begin
  assert(not assigned(M_primaryContact.BoldObject) or (M_primaryContact.BoldObject is
TPerson), SysUtils.format(BoldMemberAssertInvalidObjectType, [ClassName, 'primaryContact',
M_primaryContact.BoldObject.ClassName, 'TPerson']));
  Result := TPerson(M_primaryContact.BoldObject);
end;

procedure TCompany._SetprimaryContact(const value: TPerson);
begin
  M_primaryContact.BoldObject := value;
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;

{ TPerson }

function TPerson._Get_M_dateOfBirth: TBADate;
begin
  assert(ValidateMember('TPerson', 'dateOfBirth', 3, TBADate));
  Result := TBADate(BoldMembers[3]);
end;

function TPerson._GetdateOfBirth: TDate;
begin
  Result := M_dateOfBirth.AsDate;
end;

procedure TPerson._SetdateOfBirth(const NewValue: TDate);
begin
  M_dateOfBirth.AsDate := NewValue;
end;

function TPerson._Get_M_married: TBABoolean;
begin
  assert(ValidateMember('TPerson', 'married', 4, TBABoolean));
  Result := TBABoolean(BoldMembers[4]);
end;

function TPerson._Getmarried: Boolean;
begin
  Result := M_married.AsBoolean;
end;

procedure TPerson._Setmarried(const NewValue: Boolean);
begin
  M_married.AsBoolean := NewValue;
end;

function TPerson._Get_M_maidenName: TBAString;
begin
  assert(ValidateMember('TPerson', 'maidenName', 5, TBAString));
  Result := TBAString(BoldMembers[5]);
end;

function TPerson._GetmaidenName: String;
begin
  Result := M_maidenName.AsString;
end;
```

```
procedure TPerson._SetmaidenName(const NewValue: String);
begin
  M_maidenName.AsString := NewValue;
end;

function TPerson._Get_M_firstName: TBAString;
begin
  assert(ValidateMember('TPerson', 'firstName', 6, TBAString));
  Result := TBAString(BoldMembers[6]);
end;

function TPerson._GetfirstName: String;
begin
  Result := M_firstName.AsString;
end;

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

function TPerson._Get_M_lastName: TBAString;
begin
  assert(ValidateMember('TPerson', 'lastName', 7, TBAString));
  Result := TBAString(BoldMembers[7]);
end;

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

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

function TPerson._Getcompanies: TCompanyList;
begin
  assert(ValidateMember('TPerson', 'companies', 8, TCompanyList));
  Result := TCompanyList(BoldMembers[8]);
end;

function TPerson._GetEmployment: TEmploymentList;
begin
  assert(ValidateMember('TPerson', 'Employment', 9, TEmploymentList));
  Result := TEmploymentList(BoldMembers[9]);
end;

procedure TPersonList.Add(NewObject: TPerson);
begin
  if Assigned(NewObject) then
    AddElement(NewObject);
end;

function TPersonList.IndexOf(anObject: TPerson): Integer;
begin
  result := IndexOfElement(anObject);
end;

function TPersonList.Includes(anObject: TPerson) : Boolean;
begin
  result := IncludesElement(anObject);
end;

function TPersonList.AddNew: TPerson;
begin
  result := TPerson(InternalAddNew);
end;

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

function TPersonList.GetBoldObject(index: Integer): TPerson;
begin
  result := TPerson(GetElement(index));
```

```
end;

procedure TPersonList.SetBoldObject(index: Integer; NewObject: TPerson);
begin;
  SetElement(index, NewObject);
end;

function GeneratedCodeCRC: String;
begin
  result := '1295627635';
end;

procedure InstallObjectListClasses(BoldObjectListClasses: TBoldGeneratedClassList);
begin
  BoldObjectListClasses.AddObjectEntry('ContactRoot', TContactRootList);
  BoldObjectListClasses.AddObjectEntry('Category', TCategoryList);
  BoldObjectListClasses.AddObjectEntry('Contact', TContactList);
  BoldObjectListClasses.AddObjectEntry('ContactCategory', TContactCategoryList);
  BoldObjectListClasses.AddObjectEntry('Employment', TEmploymentList);
  BoldObjectListClasses.AddObjectEntry('Company', TCompanyList);
  BoldObjectListClasses.AddObjectEntry('Person', TPersonList);
end;

procedure InstallBusinessClasses(BoldObjectClasses: TBoldGeneratedClassList);
begin
  BoldObjectClasses.AddObjectEntry('ContactRoot', TContactRoot);
  BoldObjectClasses.AddObjectEntry('Category', TCategory);
  BoldObjectClasses.AddObjectEntry('Contact', TContact);
  BoldObjectClasses.AddObjectEntry('ContactCategory', TContactCategory);
  BoldObjectClasses.AddObjectEntry('Employment', TEmployment);
  BoldObjectClasses.AddObjectEntry('Company', TCompany);
  BoldObjectClasses.AddObjectEntry('Person', TPerson);
end;

var
  CodeDescriptor: TBoldGeneratedCodeDescriptor;

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

End Listing for ContactClasses.pas

ContactClasses.inc

Start Listing for ContactClasses.inc

```
(*************************************************)
(*                                               *)
(*  Bold for Delphi Stub File                    *)
(*                                               *)
(*  Autogenerated file for method implementations    *)
(*                                               *)
(*************************************************)

//
{$INCLUDE ContactClasses_Interface.inc}

procedure TContact._name_DeriveAndSubscribe(DerivedObject: TObject; Subscriber:
TBoldSubscriber);
begin

end;

procedure TContact._name_ReverseDerive(DerivedObject: TObject);
begin

end;

procedure TCompany._name_DeriveAndSubscribe(DerivedObject: TObject; Subscriber:
TBoldSubscriber);
begin
  Name := CompanyName;
  M_CompanyName.DefaultSubscribe(subscriber);
end;

procedure TCompany._name_ReverseDerive(DerivedObject: TObject);
begin
  CompanyName := Name;
end;

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

procedure TPerson._name_ReverseDerive(DerivedObject: TObject);
var
  aFullName: String;
  p: integer;
begin
  // strip away leading and trailing spaces
  aFullName := trim(Name);
  // 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;
```

End Listing for ContactClasses.inc

ContactClasses_Interface.inc

Starting Listing for ContactClasses_Interface.inc

```
(*****************************************)
(*      This file is autogenerated       *)
(*   Any manual changes will be LOST!    *)
(*****************************************)
(* Generated 9/08/2002 6:46:43 PM        *)
(*****************************************)
(* This file should be stored in the     *)
(* same directory as the form/datamodule *)
(* with the corresponding model          *)
(*****************************************)
(* Copyright notice:                     *)
(*                                       *)
(*****************************************)

{$IFNDEF ContactClasses_Interface.inc}
{$DEFINE ContactClasses_Interface.inc}

{$IFNDEF ContactClasses_unitheader}
unit ContactClasses;
{$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 }

  TContactRoot = class;
  TContactRootList = class;
  TCategory = class;
  TCategoryList = class;
  TContact = class;
  TContactList = class;
  TContactCategory = class;
  TContactCategoryList = class;
  TEmployment = class;
  TEmploymentList = class;
  TCompany = class;
  TCompanyList = class;
  TPerson = class;
  TPersonList = class;

  TContactRoot = class(TBoldObject)
  private
  protected
  public
  end;

  TCategory = class(TContactRoot)
  private
    function _Get_M_name: TBAString;
    function _Getname: String;
    procedure _Setname(const NewValue: String);
    function _Getcontacts: TContactList;
    function _GetContactCategory: TContactCategoryList;
  protected
  public
    property M_name: TBAString read _Get_M_name;
    property M_contacts: TContactList read _Getcontacts;
```

```
    property M_ContactCategory: TContactCategoryList read _GetContactCategory;
    property name: String read _Getname write _Setname;
    property contacts: TContactList read _Getcontacts;
    property ContactCategory: TContactCategoryList read _GetContactCategory;
  end;

  TContact = class(TContactRoot)
  private
    function _Get_M_name: TBAString;
    function _Getname: String;
    procedure _Setname(const NewValue: String);
    function _Getcategories: TCategoryList;
    function _GetContactCategory: TContactCategoryList;
  protected
    procedure _name_DeriveAndSubscribe(DerivedObject: TObject; Subscriber: TBoldSubscriber);
virtual;
    procedure _name_ReverseDerive(DerivedObject: TObject); virtual;
    function GetDeriveMethodForMember(Member: TBoldMember): TBoldDeriveAndResubscribe;
override;
    function GetReverseDeriveMethodForMember(Member: TBoldMember): TBoldReverseDerive;
override;
  public
    property M_name: TBAString read _Get_M_name;
    property M_categories: TCategoryList read _Getcategories;
    property M_ContactCategory: TContactCategoryList read _GetContactCategory;
    property name: String read _Getname write _Setname;
    property categories: TCategoryList read _Getcategories;
    property ContactCategory: TContactCategoryList read _GetContactCategory;
  end;

  TContactCategory = class(TContactRoot)
  private
    function _Getcontacts: TContact;
    function _Get_M_contacts: TBoldObjectReference;
    function _Getcategories: TCategory;
    function _Get_M_categories: TBoldObjectReference;
  protected
  public
    property M_contacts: TBoldObjectReference read _Get_M_contacts;
    property M_categories: TBoldObjectReference read _Get_M_categories;
    property contacts: TContact read _Getcontacts;
    property categories: TCategory read _Getcategories;
  end;

  TEmployment = class(TContactRoot)
  private
    function _Getemployees: TPerson;
    function _Get_M_employees: TBoldObjectReference;
    function _Getcompanies: TCompany;
    function _Get_M_companies: TBoldObjectReference;
  protected
  public
    property M_employees: TBoldObjectReference read _Get_M_employees;
    property M_companies: TBoldObjectReference read _Get_M_companies;
    property employees: TPerson read _Getemployees;
    property companies: TCompany read _Getcompanies;
  end;

  TCompany = class(TContact)
  private
    function _Get_M_companyName: TBAString;
    function _GetcompanyName: String;
    procedure _SetcompanyName(const NewValue: String);
    function _Getemployees: TPersonList;
    function _GetEmployment: TEmploymentList;
    function _GetprimaryContact: TPerson;
    function _Get_M_primaryContact: TBoldObjectReference;
    procedure _SetprimaryContact(const value: TPerson);
  protected
    procedure _name_DeriveAndSubscribe(DerivedObject: TObject; Subscriber: TBoldSubscriber);
override;
    procedure _name_ReverseDerive(DerivedObject: TObject); override;
  public
    property M_companyName: TBAString read _Get_M_companyName;
    property M_employees: TPersonList read _Getemployees;
    property M_Employment: TEmploymentList read _GetEmployment;
    property M_primaryContact: TBoldObjectReference read _Get_M_primaryContact;
    property companyName: String read _GetcompanyName write _SetcompanyName;
    property employees: TPersonList read _Getemployees;
    property Employment: TEmploymentList read _GetEmployment;
```

```
    property primaryContact: TPerson read _GetprimaryContact write _SetprimaryContact;
  end;

  TPerson = class(TContact)
  private
    function _Get_M_dateOfBirth: TBADate;
    function _GetdateOfBirth: TDate;
    procedure _SetdateOfBirth(const NewValue: TDate);
    function _Get_M_married: TBABoolean;
    function _Getmarried: Boolean;
    procedure _Setmarried(const NewValue: Boolean);
    function _Get_M_maidenName: TBAString;
    function _GetmaidenName: String;
    procedure _SetmaidenName(const NewValue: String);
    function _Get_M_firstName: TBAString;
    function _GetfirstName: String;
    procedure _SetfirstName(const NewValue: String);
    function _Get_M_lastName: TBAString;
    function _GetlastName: String;
    procedure _SetlastName(const NewValue: String);
    function _Getcompanies: TCompanyList;
    function _GetEmployment: TEmploymentList;
  protected
    procedure _name_DeriveAndSubscribe(DerivedObject: TObject; Subscriber: TBoldSubscriber);
override;
    procedure _name_ReverseDerive(DerivedObject: TObject); override;
  public
    property M_dateOfBirth: TBADate read _Get_M_dateOfBirth;
    property M_married: TBABoolean read _Get_M_married;
    property M_maidenName: TBAString read _Get_M_maidenName;
    property M_firstName: TBAString read _Get_M_firstName;
    property M_lastName: TBAString read _Get_M_lastName;
    property M_companies: TCompanyList read _Getcompanies;
    property M_Employment: TEmploymentList read _GetEmployment;
    property dateOfBirth: TDate read _GetdateOfBirth write _SetdateOfBirth;
    property married: Boolean read _Getmarried write _Setmarried;
    property maidenName: String read _GetmaidenName write _SetmaidenName;
    property firstName: String read _GetfirstName write _SetfirstName;
    property lastName: String read _GetlastName write _SetlastName;
    property companies: TCompanyList read _Getcompanies;
    property Employment: TEmploymentList read _GetEmployment;
  end;

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

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

  TContactList = class(TContactRootList)
  protected
    function GetBoldObject(index: Integer): TContact;
    procedure SetBoldObject(index: Integer; NewObject: TContact);
  public
    function Includes(anObject: TContact): Boolean;
    function IndexOf(anObject: TContact): Integer;
    procedure Add(NewObject: TContact);
    function AddNew: TContact;
```

```
    procedure Insert(index: Integer; NewObject: TContact);
    property BoldObjects[index: Integer]: TContact read GetBoldObject write SetBoldObject;
default;
  end;

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

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

  TCompanyList = class(TContactList)
  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;

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

function GeneratedCodeCRC: String;

implementation

uses
  // implementation uses
  // implementation dependencies
  // other
  BoldGeneratedCodeDictionary;

{$ENDIF}
```

End Listing for ContactClasses_Interface.inc

Date.xml

Start Listing for Data.xml

<ValueSpace xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance" xml:space="preserve"><Person><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>0</DbValue></id><persistencestate>0</persist
encestate><existencestate>1</existencestate><timestamp>-
1</timestamp><members><categories><persistencestate>0</persistencestate><content><idlist1><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>128256817</DbValue></id></idlist1>
<idlist2><id
xsi:type="BoldDefaultObjectId"><ClassName>Category</ClassName><DbValue>1019612489</DbValue></id></idlist2></conte
nt></categories><dateOfBirth><persistencestate>0</persistencestate><content><Year>1972</Year><Month>12</Month><Day
>9</Day></content></dateOfBirth><married><persistencestate>0</persistencestate><content>1</content></married><maiden
Name><persistencestate>0</persistencestate><content></content></maidenName><firstName><persistencestate>0</persiste
ncestate><content>Danny</content></firstName><lastName><persistencestate>0</persistencestate><content>Smith</content
></lastName><companies><persistencestate>0</persistencestate><content><idlist1><id
xsi:type="BoldDefaultObjectId"><ClassName>Employment</ClassName><DbValue>799375788</DbValue></id></idlist1><idlis
t2><id
xsi:type="BoldDefaultObjectId"><ClassName>Company</ClassName><DbValue>684384293</DbValue></id></idlist2></conte
nt></companies></members></Person><Person><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>67387906</DbValue></id><persistencestate>0
</persistencestate><existencestate>1</existencestate><timestamp>-
1</timestamp><members><categories><persistencestate>0</persistencestate><content><idlist1><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>1498245597</DbValue></id></idlist1
><idlist2><id
xsi:type="BoldDefaultObjectId"><ClassName>Category</ClassName><DbValue>151546482</DbValue></id></idlist2></conten
t></categories><dateOfBirth><persistencestate>0</persistencestate><content><Year>1968</Year><Month>2</Month><Day>1
8</Day></content></dateOfBirth><married><persistencestate>0</persistencestate><content>0</content></married><maidenN
ame><persistencestate>0</persistencestate><content>Timms</content></maidenName><firstName><persistencestate>0</per
sistencestate><content>Sarah</content></firstName><lastName><persistencestate>0</persistencestate><content>Heath</co
ntent></lastName><companies><persistencestate>0</persistencestate><content><idlist1/><idlist2/></content></companies></
members></Person><Person><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>1849087502</DbValue></id><persistencestate
>0</persistencestate><existencestate>1</existencestate><timestamp>-
1</timestamp><members><categories><persistencestate>0</persistencestate><content><idlist1><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>629849591</DbValue></id><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>704214725</DbValue></id></idlist1>
<idlist2><id xsi:type="BoldDefaultObjectId"><ClassName>Category</ClassName><DbValue>1019612489</DbValue></id><id
xsi:type="BoldDefaultObjectId"><ClassName>Category</ClassName><DbValue>151546482</DbValue></id></idlist2></conten
t></categories><dateOfBirth><persistencestate>0</persistencestate><content><Year>1956</Year><Month>12</Month><Day>
2</Day></content></dateOfBirth><married><persistencestate>0</persistencestate><content>0</content></married><maidenN
ame><persistencestate>0</persistencestate><content></content></maidenName><firstName><persistencestate>0</persisten
cestate><content>David</content></firstName><lastName><persistencestate>0</persistencestate><content>Martin</content>
</lastName><companies><persistencestate>0</persistencestate><content><idlist1><id
xsi:type="BoldDefaultObjectId"><ClassName>Employment</ClassName><DbValue>1541856444</DbValue></id><id
xsi:type="BoldDefaultObjectId"><ClassName>Employment</ClassName><DbValue>658507187</DbValue></id></idlist1><idlis
t2><id xsi:type="BoldDefaultObjectId"><ClassName>Company</ClassName><DbValue>684384293</DbValue></id><id
xsi:type="BoldDefaultObjectId"><ClassName>Company</ClassName><DbValue>347453115</DbValue></id></idlist2></conte
nt></companies></members></Person><Person><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>435039309</DbValue></id><persistencestate>
0</persistencestate><existencestate>1</existencestate><timestamp>-
1</timestamp><members><categories><persistencestate>0</persistencestate><content><idlist1><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>1805721148</DbValue></id></idlist1
><idlist2><id
xsi:type="BoldDefaultObjectId"><ClassName>Category</ClassName><DbValue>1019612489</DbValue></id></idlist2></conte
nt></categories><dateOfBirth><persistencestate>0</persistencestate><content><Year>1959</Year><Month>5</Month><Day>
12</Day></content></dateOfBirth><married><persistencestate>0</persistencestate><content>0</content></married><maiden
Name><persistencestate>0</persistencestate><content></content></maidenName><firstName><persistencestate>0</persiste
ncestate><content>Tony</content></firstName><lastName><persistencestate>0</persistencestate><content>Carey</content>
</lastName><companies><persistencestate>0</persistencestate><content><idlist1><id
xsi:type="BoldDefaultObjectId"><ClassName>Employment</ClassName><DbValue>176119771</DbValue></id></idlist1><idlis
t2><id
xsi:type="BoldDefaultObjectId"><ClassName>Company</ClassName><DbValue>347453115</DbValue></id></idlist2></conte
nt></companies></members></Person><Person><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>586093958</DbValue></id><persistencestate>
0</persistencestate><existencestate>1</existencestate><timestamp>-
1</timestamp><members><categories><persistencestate>0</persistencestate><content><idlist1><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>1663580198</DbValue></id></idlist1
><idlist2><id
xsi:type="BoldDefaultObjectId"><ClassName>Category</ClassName><DbValue>151546482</DbValue></id></idlist2></conten
t></categories><dateOfBirth><persistencestate>0</persistencestate><content><Year>1968</Year><Month>1</Month><Day>6
</Day></content></dateOfBirth><married><persistencestate>0</persistencestate><content>1</content></married><maidenNa
me><persistencestate>0</persistencestate><content></content></maidenName><firstName><persistencestate>0</persistenc
estate><content>John</content></firstName><lastName><persistencestate>0</persistencestate><content>Doe</content></las
tName><companies><persistencestate>0</persistencestate><content><idlist1/><idlist2/></content></companies></members>
</Person><Person><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>1442366880</DbValue></id><persistencestate
>0</persistencestate><existencestate>1</existencestate><timestamp>-

1</timestamp><members><categories><persistencestate>0</persistencestate><content><idlist1/><idlist2/></content></categories><dateOfBirth><persistencestate>0</persistencestate><content><Year>1966</Year><Month>12</Month><Day>30</Day></content></dateOfBirth><married><persistencestate>0</persistencestate><content>0</content></married><maidenName><persistencestate>0</persistencestate><content></content></maidenName><firstName><persistencestate>0</persistencestate><content>Martin</content></firstName><lastName><persistencestate>0</persistencestate><content>Johns</content></lastName><companies><persistencestate>0</persistencestate><content><idlist1/><idlist2/></content></companies></members></Person><Company><id
xsi:type="BoldDefaultObjectId"><ClassName>Company</ClassName><DbValue>684384293</DbValue></id><persistencestate>0</persistencestate><existencestate>1</existencestate><timestamp>-1</timestamp><members><categories><persistencestate>0</persistencestate><content><idlist1><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>790073146</DbValue></id></idlist1><idlist2><id
xsi:type="BoldDefaultObjectId"><ClassName>Category</ClassName><DbValue>1019612489</DbValue></id></idlist2></content></categories><companyName><persistencestate>0</persistencestate><content>Toll Corporation</content></companyName><employees><persistencestate>0</persistencestate><content><idlist1><id
xsi:type="BoldDefaultObjectId"><ClassName>Employment</ClassName><DbValue>799375788</DbValue></id><id
xsi:type="BoldDefaultObjectId"><ClassName>Employment</ClassName><DbValue>1541856444</DbValue></id></idlist1><idlist2><id xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>0</DbValue></id><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>1849087502</DbValue></id></idlist2></content></employees><primaryContact><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>0</DbValue></id><OrderNo>0</OrderNo></content></primaryContact></members></Company><Company><id
xsi:type="BoldDefaultObjectId"><ClassName>Company</ClassName><DbValue>347453115</DbValue></id><persistencestate>0</persistencestate><existencestate>1</existencestate><timestamp>-1</timestamp><members><categories><persistencestate>0</persistencestate><content><idlist1><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>1969853911</DbValue></id></idlist1><idlist2><id
xsi:type="BoldDefaultObjectId"><ClassName>Category</ClassName><DbValue>1019612489</DbValue></id></idlist2></content></categories><companyName><persistencestate>0</persistencestate><content>ACME Construction</content></companyName><employees><persistencestate>0</persistencestate><content><idlist1><id
xsi:type="BoldDefaultObjectId"><ClassName>Employment</ClassName><DbValue>176119771</DbValue></id><id
xsi:type="BoldDefaultObjectId"><ClassName>Employment</ClassName><DbValue>658507187</DbValue></id></idlist1><idlist2><id xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>435039309</DbValue></id><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>1849087502</DbValue></id></idlist2></content></employees><primaryContact><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>1849087502</DbValue></id><OrderNo>0</OrderNo></content></primaryContact></members></Company><Employment><id
xsi:type="BoldDefaultObjectId"><ClassName>Employment</ClassName><DbValue>799375788</DbValue></id><persistencestate>0</persistencestate><existencestate>1</existencestate><timestamp>-1</timestamp><members><employees><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>0</DbValue></id><OrderNo>0</OrderNo></content></employees><companies><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Company</ClassName><DbValue>684384293</DbValue></id><OrderNo>0</OrderNo></content></companies></members></Employment><Employment><id
xsi:type="BoldDefaultObjectId"><ClassName>Employment</ClassName><DbValue>176119771</DbValue></id><persistencestate>0</persistencestate><existencestate>1</existencestate><timestamp>-1</timestamp><members><employees><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>435039309</DbValue></id><OrderNo>0</OrderNo></content></employees><companies><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Company</ClassName><DbValue>347453115</DbValue></id><OrderNo>0</OrderNo></content></companies></members></Employment><Category><id
xsi:type="BoldDefaultObjectId"><ClassName>Category</ClassName><DbValue>1019612489</DbValue></id><persistencestate>0</persistencestate><existencestate>1</existencestate><timestamp>-1</timestamp><members><name><persistencestate>0</persistencestate><content>Business</content></name><contacts><persistencestate>0</persistencestate><content><idlist1><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>1805721148</DbValue></id><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>128256817</DbValue></id><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>629849591</DbValue></id><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>1969853911</DbValue></id><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>790073146</DbValue></id><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>707588246</DbValue></id></idlist1><idlist2><id xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>435039309</DbValue></id><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>0</DbValue></id><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>1849087502</DbValue></id><id
xsi:type="BoldDefaultObjectId"><ClassName>Company</ClassName><DbValue>347453115</DbValue></id><id
xsi:type="BoldDefaultObjectId"><ClassName>Company</ClassName><DbValue>684384293</DbValue></id><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>1812843352</DbValue></id></idlist2></content></contacts></members></Category><Category><id
xsi:type="BoldDefaultObjectId"><ClassName>Category</ClassName><DbValue>151546482</DbValue></id><persistencestate>0</persistencestate><existencestate>1</existencestate><timestamp>-1</timestamp><members><name><persistencestate>0</persistencestate><content>Personal</content></name><contacts><persistencestate>0</persistencestate><content><idlist1><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>1663580198</DbValue></id><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>704214725</DbValue></id><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>1498245597</DbValue></id></idlist1><idlist2><id xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>586093958</DbValue></id><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>1849087502</DbValue></id><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>67387906</DbValue></id></idlist2></content></contacts></members></Category><ContactCategory><id

xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>1805721148</DbValue></id><persist
encestate>0</persistencestate><existencestate>1</existencestate><timestamp>-
1</timestamp><members><contacts><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>435039309</DbValue></id><OrderNo>0</Orde
rNo></content></contacts><categories><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Category</ClassName><DbValue>1019612489</DbValue></id><OrderNo>0</O
rderNo></content></categories></members></ContactCategory><ContactCategory><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>128256817</DbValue></id><persiste
ncestate>0</persistencestate><existencestate>1</existencestate><timestamp>-
1</timestamp><members><contacts><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>0</DbValue></id><OrderNo>0</OrderNo></con
tent></contacts><categories><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Category</ClassName><DbValue>1019612489</DbValue></id><OrderNo>0</O
rderNo></content></categories></members></ContactCategory><ContactCategory><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>629849591</DbValue></id><persiste
ncestate>0</persistencestate><existencestate>1</existencestate><timestamp>-
1</timestamp><members><contacts><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>1849087502</DbValue></id><OrderNo>0</Ord
erNo></content></contacts><categories><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Category</ClassName><DbValue>1019612489</DbValue></id><OrderNo>0</O
rderNo></content></categories></members></ContactCategory><ContactCategory><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>1969853911</DbValue></id><persist
encestate>0</persistencestate><existencestate>1</existencestate><timestamp>-
1</timestamp><members><contacts><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Company</ClassName><DbValue>347453115</DbValue></id><OrderNo>0</Or
derNo></content></contacts><categories><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Category</ClassName><DbValue>1019612489</DbValue></id><OrderNo>0</O
rderNo></content></categories></members></ContactCategory><ContactCategory><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>790073146</DbValue></id><persiste
ncestate>0</persistencestate><existencestate>1</existencestate><timestamp>-
1</timestamp><members><contacts><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Company</ClassName><DbValue>684384293</DbValue></id><OrderNo>0</Or
derNo></content></contacts><categories><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Category</ClassName><DbValue>1019612489</DbValue></id><OrderNo>0</O
rderNo></content></categories></members></ContactCategory><ContactCategory><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>1663580198</DbValue></id><persist
encestate>0</persistencestate><existencestate>1</existencestate><timestamp>-
1</timestamp><members><contacts><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>586093958</DbValue></id><OrderNo>0</Orde
rNo></content></contacts><categories><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Category</ClassName><DbValue>151546482</DbValue></id><OrderNo>0</Or
derNo></content></categories></members></ContactCategory><ContactCategory><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>704214725</DbValue></id><persiste
ncestate>0</persistencestate><existencestate>1</existencestate><timestamp>-
1</timestamp><members><contacts><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>1849087502</DbValue></id><OrderNo>0</Ord
erNo></content></contacts><categories><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Category</ClassName><DbValue>151546482</DbValue></id><OrderNo>0</Or
derNo></content></categories></members></ContactCategory><ContactCategory><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>1498245597</DbValue></id><persist
encestate>0</persistencestate><existencestate>1</existencestate><timestamp>-
1</timestamp><members><contacts><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>67387906</DbValue></id><OrderNo>0</Order
No></content></contacts><categories><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Category</ClassName><DbValue>151546482</DbValue></id><OrderNo>0</Or
derNo></content></categories></members></ContactCategory><Person><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>1812843352</DbValue></id><persistencestate
>0</persistencestate><existencestate>1</existencestate><timestamp>-
1</timestamp><members><categories><persistencestate>0</persistencestate><content><idlist1><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>707588246</DbValue></id></idlist1>
<idlist2><id
xsi:type="BoldDefaultObjectId"><ClassName>Category</ClassName><DbValue>1019612489</DbValue></id></idlist2></conte
nt></categories><dateOfBirth><persistencestate>0</persistencestate><content><Year>1973</Year><Month>9</Month><Day>
12</Day></content></dateOfBirth><married><persistencestate>0</persistencestate><content>1</content></married><maiden
Name><persistencestate>0</persistencestate><content>Thomas</content></maidenName><firstName><persistencestate>0</
persistencestate><content>Jane</content></firstName><lastName><persistencestate>0</persistencestate><content>Doe</co
ntent></lastName><companies><persistencestate>0</persistencestate><content><idlist1><id
xsi:type="BoldDefaultObjectId"><ClassName>Employment</ClassName><DbValue>349236356</DbValue></id></idlist1><idlis
t2><id
xsi:type="BoldDefaultObjectId"><ClassName>Company</ClassName><DbValue>914127454</DbValue></id></idlist2></conte
nt></companies></members></Person><Employment><id
xsi:type="BoldDefaultObjectId"><ClassName>Employment</ClassName><DbValue>1541856444</DbValue></id><persistence
state>0</persistencestate><existencestate>1</existencestate><timestamp>-
1</timestamp><members><employees><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>1849087502</DbValue></id><OrderNo>0</Ord
erNo></content></employees><companies><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Company</ClassName><DbValue>684384293</DbValue></id><OrderNo>0</Or
derNo></content></companies></members></Employment><Employment><id
xsi:type="BoldDefaultObjectId"><ClassName>Employment</ClassName><DbValue>658507187</DbValue></id><persistencest

ate>0</persistencestate><existencestate>1</existencestate><timestamp>-
1</timestamp><members><employees><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>1849087502</DbValue></id><OrderNo>0</Ord
erNo></content></employees><companies><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Company</ClassName><DbValue>347453115</DbValue></id><OrderNo>0</Or
derNo></content></companies></members></Employment><Company><id
xsi:type="BoldDefaultObjectId"><ClassName>Company</ClassName><DbValue>914127454</DbValue></id><persistencestat
e>0</persistencestate><existencestate>1</existencestate><timestamp>-
1</timestamp><members><categories><persistencestate>0</persistencestate><content><idlist1/><idlist2/></content></catego
ries><companyName><persistencestate>0</persistencestate><content>Geoff's
Warehouse</content></companyName><employees><persistencestate>0</persistencestate><content><idlist1><id
xsi:type="BoldDefaultObjectId"><ClassName>Employment</ClassName><DbValue>349236356</DbValue></id></idlist1><idlis
t2><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>1812843352</DbValue></id></idlist2></content
></employees><primaryContact><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>1812843352</DbValue></id><OrderNo>0</Ord
erNo></content></primaryContact></members></Company><Employment><id
xsi:type="BoldDefaultObjectId"><ClassName>Employment</ClassName><DbValue>349236356</DbValue></id><persistencest
ate>0</persistencestate><existencestate>1</existencestate><timestamp>-
1</timestamp><members><employees><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>1812843352</DbValue></id><OrderNo>0</Ord
erNo></content></employees><companies><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Company</ClassName><DbValue>914127454</DbValue></id><OrderNo>0</Or
derNo></content></companies></members></Employment><ContactCategory><id
xsi:type="BoldDefaultObjectId"><ClassName>ContactCategory</ClassName><DbValue>707588246</DbValue></id><persiste
ncestate>0</persistencestate><existencestate>1</existencestate><timestamp>-
1</timestamp><members><contacts><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Person</ClassName><DbValue>1812843352</DbValue></id><OrderNo>0</Ord
erNo></content></contacts><categories><persistencestate>0</persistencestate><content><id
xsi:type="BoldDefaultObjectId"><ClassName>Category</ClassName><DbValue>1019612489</DbValue></id><OrderNo>0</O
rderNo></content></categories></members></ContactCategory></ValueSpace>

End Listing for Data.xml