

# Créez votre première application avec Enterprise Core Objects (ECO)<sup>™</sup>

---

Avec C#Builder ou Delphi pour .NET

Un livre blanc de Borland

*Par Anthony Richardson*

Janvier 2004

---

**Borland<sup>®</sup>**

## Table des matières

Avec C#Builder ou Delphi pour .NET.....	1
Un livre blanc de Borland.....	1
Créez votre première application ECO.....	3
Introduction.....	3
Qu'est-ce que la modélisation?.....	3
L'applications exemple.....	4
Construire l'application.....	5
Ajouter une interface utilisateur.....	11
Exécuter l'application.....	13
Données persistantes.....	14
Fiches automatiques.....	16
Conclusion.....	17

## Créez votre première application ECO

Cet article présente les étapes de la construction d'une application simple en utilisant la technologie Borland® Enterprise Core Objects (ECO)<sup>™</sup> de Borland® Delphi<sup>™</sup> édition Architecte. L'équivalent pour C#Builder édition Architecte se trouve à l'adresse <http://www.borland.com/csharpbuilder/architect/eco/tutorial/>

Le projet du tutorial se trouve à l'adresse [ftp://ftpc.borland.com/pub/delphi\\_net/D8\\_ECOTutorial1\\_source.zip](ftp://ftpc.borland.com/pub/delphi_net/D8_ECOTutorial1_source.zip)

### Introduction

Enterprise Core Objects (ECO)<sup>™</sup> est un framework de développement et d'exécution pour un développement piloté par modèles. Le processus traditionnel de transformation des exigences projet en une vue informatique compréhensible peut impacter la conception des bases de données, des applications et des interfaces ; chacune représentant la logique métier à sa façon. ECO applique la flexibilité de l'orienté objet aux couches métier et de persistance de votre application, rendant par la même la conception et le développement plus simple.

### Qu'est-ce que la modélisation?

Nous avons mentionné ci-dessus le développement piloté par modèles, mais qu'est-ce qu'un modèle? Un modèle est une description ou une vue de quelque chose. Un modèle logiciel est à la représentation du domaine du problème dans lequel le logiciel s'exécutera ce qu'une maquette ou l'aéromodélisme est à l'aviation. Il est possible de générer différents modèles pour un projet logiciel donné. Les méthodologies d'ingénierie logicielle populaire décrivent les modèles et les activités afférentes à différentes étapes du cycle de développement logiciel. De plus, les modèles simplifient la communication entre les développeurs, les utilisateurs et d'autres parties prenantes. En créant de multiples types de modèles vous pouvez décrire un système logiciel complexe au travers d'une série de vues progressivement plus détaillées du problème.

La difficulté dans cette utilisation des modèles, est que, lorsque vous avez suffisamment affiné un modèle au point de faire disparaître toute ambiguïté et de le rendre exécutable, il n'est dès lors plus reconnaissable comme description du problème original. De même lorsque nous convertissons progressivement une description texte d'un problème en code source et schémas de bases de données, seuls les développeurs (et le compilateur) peuvent encore la comprendre. De fait, à chaque modification du code, nous devons opérer cette transformation d'une compréhension métier vers une compréhension « compilateur ». Et lorsque nous regardons le code écrit par d'autres, nous devons le convertir dans l'autre sens, du code source vers la logique métier.

ECO fournit une façon de supprimer ce besoin d'appliquer manuellement nombre de ces transformations. ECO permet à un développeur de créer un modèle du problème, représentant mieux la logique métier, et toujours d'une façon non ambiguë et exécutable. Ce que l'on présente généralement comme une élévation du niveau d'abstraction. En faisant cela, l'aptitude au changement et à la compréhension des logiciels complexes est rendue plus simple et moins onéreuse.

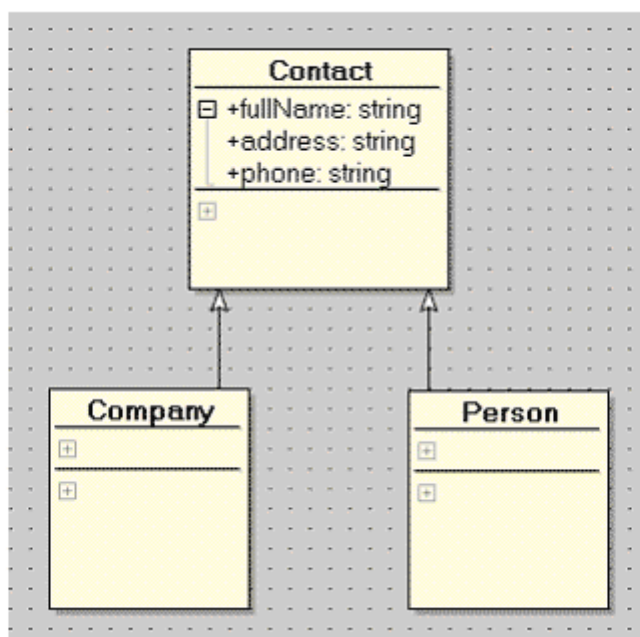
Lorsque nous créons des modèles dans ECO, les classes modélisées sont souvent désignées comme Business Classes ou Domain Classes. A l'exécution, nous les désignons comme Business Objects ou Domain Objects. La raison pour laquelle nous utilisons ces noms est que les classes représentent directement les entités dans le domaine du problème métier. Par exemple, dans une application d'inscription à l'université nous pouvons modéliser les classes métier Student, Teacher et Subject.

## L'applications exemple

Ce tutoriel prendra exemple sur la construction d'une application de gestion de carnet d'adresses qui sauvegarde ses données au format XML. Les besoins de base pour notre application sont :

- Stocker les informations de contact pour les individus.
- Stocker les informations de contact pour les entreprises.

Nous allons implémenter le modèle UML™ (Unified Modeling Language™) suivant pour notre application de carnet d'adresses:



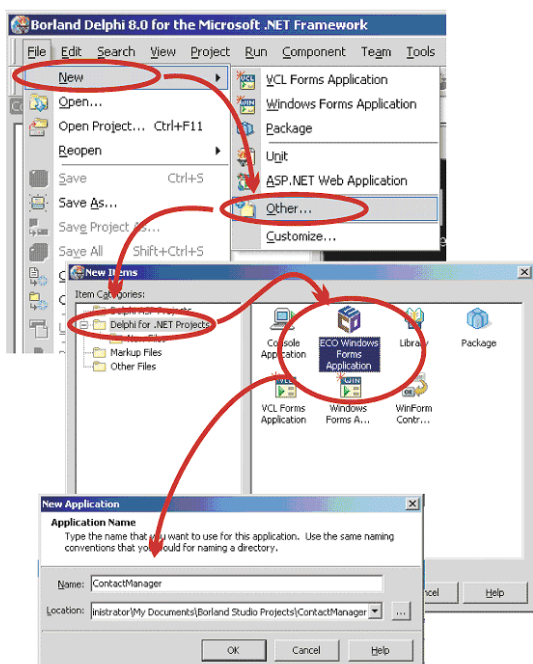
Si vous n'êtes pas familier avec UML, le diagramme ci-dessus montre trois classes, représentées par les rectangles. La classe 'Contact' bien que non marquée comme telle dans le diagramme, est une classe abstraite. Les flèches indiquent que les classes 'Company' et 'Person' descendent de la classe 'Contact'. Le modèle montre les règles métier suivantes:

- Un Contact a un nom, une adresse et un numéro de téléphone.
- Un Contact est soit une personne (Personne) soit une entreprise (Company).

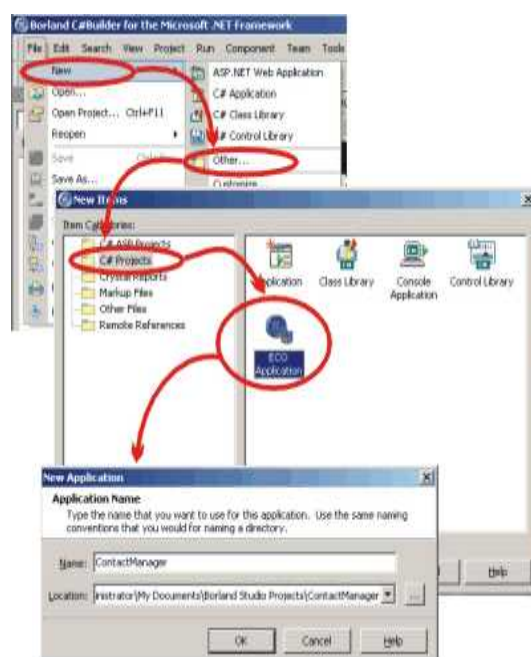
Pour cet exemple, nous pourrions implémenter ce modèle en utilisant une classe et un simple booléen pour indiquer que le contact est une personne.

## Construire l'application

- Sélectionnez "Autre" depuis le menu "Fichier|Nouveau".
- Depuis la boîte de dialogue "Nouveaux éléments", sélectionnez "Projets Delphi pour .Net" (ou "Projets C#"); puis "Application Windows Forms ECO".
- Dans la boîte de dialogue Nouvelle Application, saisissez le nom de l'application : "ContactManager"; puis cliquez sur "OK".

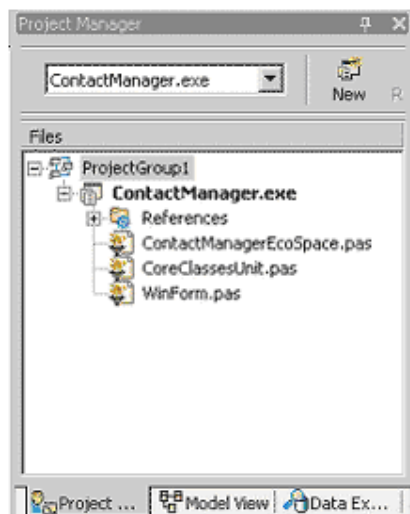


**Delphi**

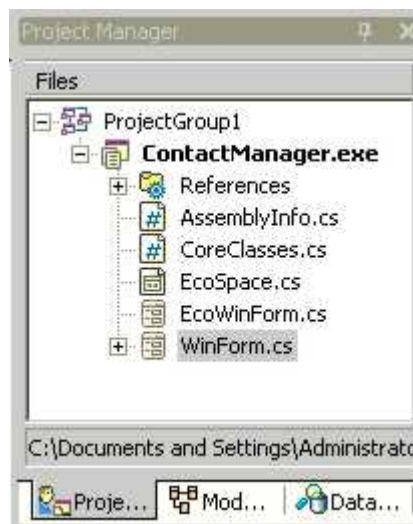


**C#Builder**

L'environnement de développement créé un squelette d'application ECO. Le Gestionnaire de Projet (Vue|Gestionnaire de Projet) affiche les fichiers automatiquement générés. Vous trouverez deux fichiers différents d'une application Delphi normale : ContactManagerEcoSpace.pas et CoreClassesUnit.pas. (Vous en dénombrez trois pour C# : CoreClasses.cs, EcoSpace.cs, EcoWinForm.cs.)



Delphi



C#Builder

### **CoreClassesUnit.pas (ou CoreClassesUnit.cs)**

Ce fichier contient un "package UML" vide. Les classes de notre application ECO y seront créées par défaut. Il est possible de créer de multiples packages, Néanmoins, ce tutorial n'utilisera que ce package par défaut.

### **ContactManagerEcoSpace.pas (ou EcoSpace.cs)**

Il s'agit de "l'espace ECO" (EcoSpace) de votre application. Un espace ECO contient les objets de l'application à l'exécution. Il fournit les liens entre les objets et le mécanisme de persistance (XML ou SGBDR). L'espace Eco peut être interrogé en utilisant le langage OCL (Object Constraint Language) pour retrouver les objets dans l'espace Eco.

Dans le système d'aide, on peut lire :

*"Un espace ECO est un conteneur d'objets. A l'exécution, l'espace ECO contient les véritables instances des classes de votre modèle. Il est commode de considérer l'espace ECO comme une instance du modèle, au même titre qu'un objet est une instance d'une classe. Les objets contenus dans l'espace ECO conservent les propriétés du domaine (attributs et opérations) et les relations définies dans le modèle.*

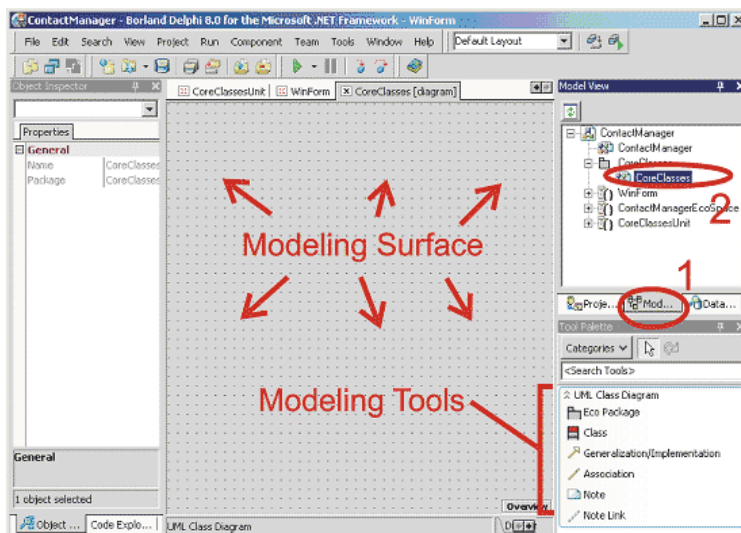
*En tant que conteneur d'objets, un espace ECO est à la fois un cache, et un contexte transactionnel. Dans un espace ECO, un autre composant appelé mappeur de persistance est utilisé comme canal de communication entre la couche de persistance (soit un SGBDR soit un fichier XML), et les instances des classes définies dans le modèle. Lorsque vous configurez un espace ECO en utilisant l'EDI, vous sélectionnez les packages UML de votre modèle que souhaitez rendre persistants dans l'espace ECO."*

## Construire un modèle ECO

ECO utilise un sous-ensemble du langage UML (Unified Modeling Language). Les éléments spécifiques sont les diagrammes de classes et l'OCL (Object Constraint Language).

### Créer le Modèle

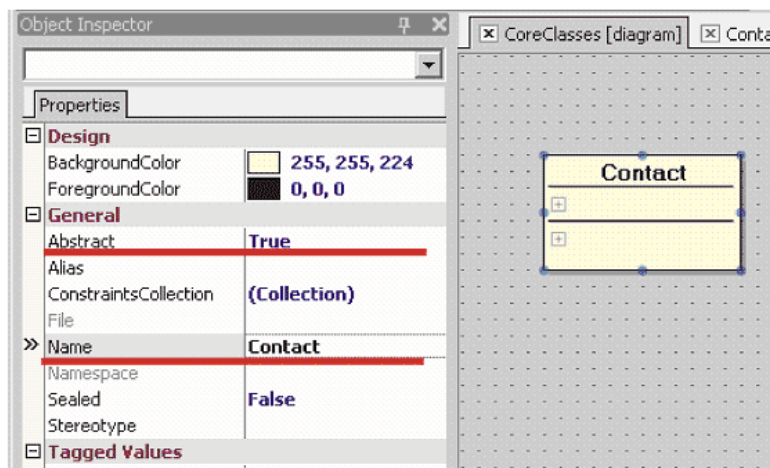
Dans la "Vue Modèle" double-cliquez l'élément du digramme "CoreClasses". Cela ouvrira l'éditeur de surface de modélisation. A l'ouverture, la "Palette d'Outils" change afin de contenir les outils pour générer votre modèle.



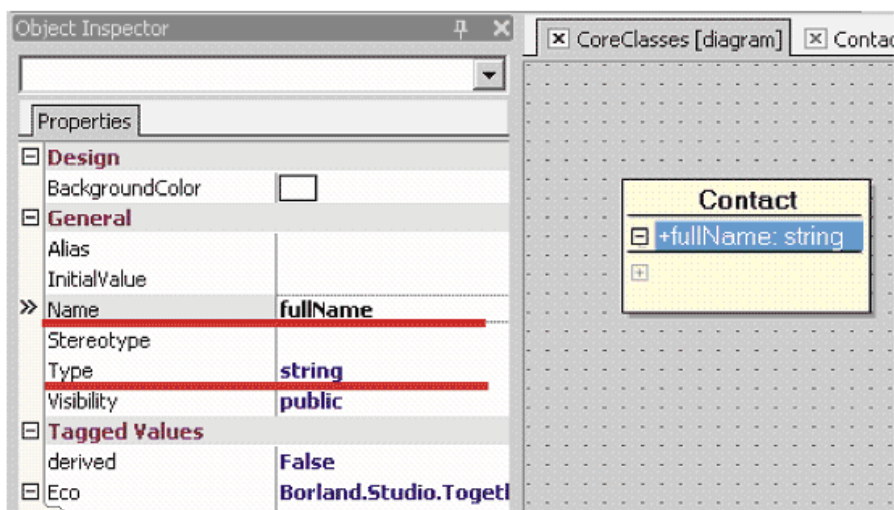
Commencez par créer une nouvelle classe. Vous pouvez faire cela de trois façons.

1. Sélectionnez l'outil "Classe" dans la "Palette d'Outils" et cliquez dans la surface de modélisation.
2. Cliquez-droit dans la surface de modélisation et sélectionnez Ajouter|Classe depuis le menu pop-up.
3. Cliquez-droit dans la surface de modélisation et sélectionnez "Ajouter un nouvel élément..." depuis le menu pop-up. Sélectionnez le modèle de classe et validez.

Une fois la classe créée, assurez-vous qu'elle est sélectionnée de telle sorte que les propriétés de la classe soient visibles dans "l'Inspecteur d'Objets". Sélectionnez la propriété "Name" et saisissez "Contact". Positionnez la propriété "Abstract" à "True". Ainsi Borland Delphi 8 créera une classe abstraite Delphi dénommée Contact.



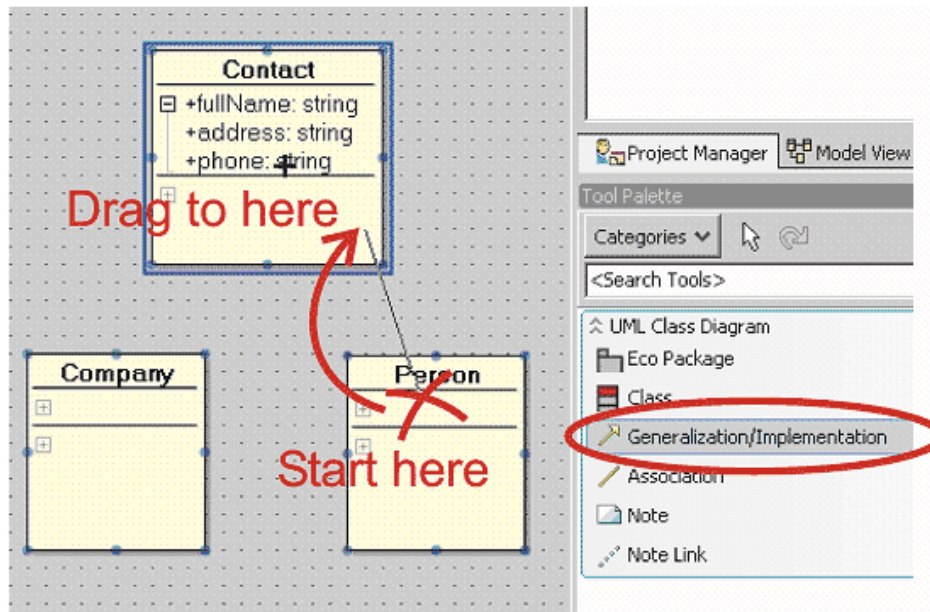
Cliquez-droit sur la classe et sélectionnez "Ajout|Attribut". Sélectionnez l'attribut dans la classe à l'aide de la souris de telle sorte que "l'Inspecteur d'Objets" affiche les propriétés de l'attribut. Entrez "fullName" pour le nom de l'attribut et positionnez son type à "String".



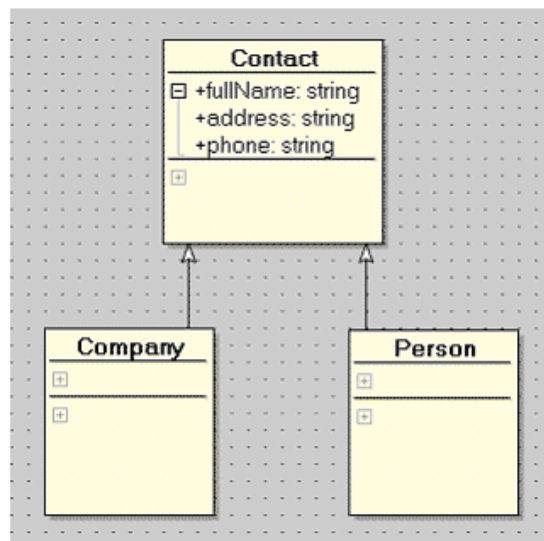
Donnez l'attribut string à "Address" et "phone" pour la classe "Contact". Ajoutez les classes "Person" et "Company" au modèle. Laissez la propriété "Abstract" à "False" puisque ce sont des classes bien concrètes pour lesquelles nous allons avoir des instances.

Nous avons besoin maintenant de créer une relation de généralisation entre la classe "Person" et la classe "Contact". Une relation de généralisation spécifie que "Person" est un type de "Contact", et hérite de tous les attributs et opérations de "Contact".

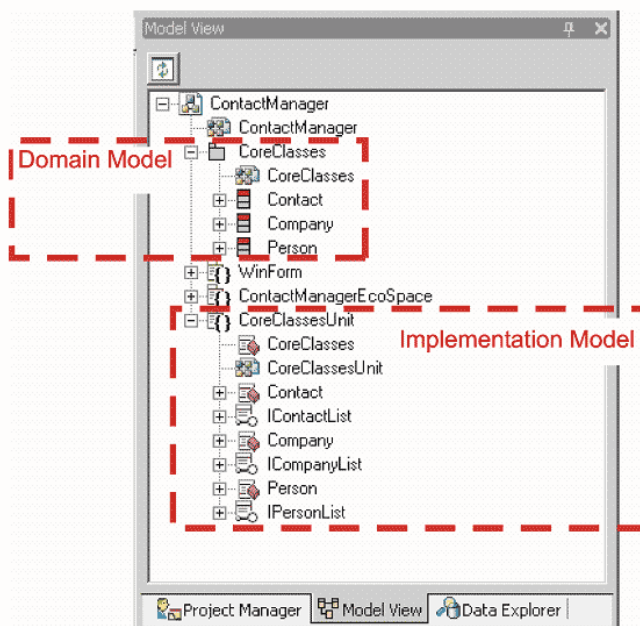
Pour créer cette relation, sélectionnez l'outil "Généralisation/Implémentation" et sélectionnez la classe "Person" et glissez la vers la classe "Contact".



Créez la même relation de généralisation pour la classe "Company". Le modèle complet se trouve ci-dessous.



Il est intéressant d'avoir un oeil sur "Vue Modèle " maintenant que nous avons ajouté des classes. La Vue Modèle montre les deux modèles de l'application ECO. Le modèle de domaine est le modèle que nous venons de créer, le modèle d'implémentation est généré automatiquement par ECO. Le modèle d'implémentation est créé sous forme de classes Delphi.



Le code généré par ECO montre l'étendu des détails d'implémentation que ECO prend en charge pour les développeurs. Ce code comprend les listes de gestion de chaque classe de domaine, des classes énumérateur, un adaptateur et tout le code de gestion pour maintenir les objets dans une application en fonctionnement. Habituellement, le développeur est responsable de ce code qui passe rapidement d'un modèle lisible d'une représentation claire du problème, en une masse de structures complexes et d'éléments d'architecture sans rapport avec le problème réel.

## Ajouter une interface utilisateur

Maintenant que le modèle a été créé, il est nécessaire d'ajouter une interface utilisateur simple pour explorer le comportement à l'exécution d'ECO.

**NOTE: A ce moment du tutorial, il est important de compiler l'application.**

ECO a besoin d'avoir le modèle compilé pour que le concepteur d'interface utilisateur puisse afficher les informations du modèle. Le système d'exécution d'ECO repose sur l'interrogation des classes en utilisant la réflexion .Net. Pour que cela fonctionne, vous devez compiler votre application.

En utilisant la "Palette d'Outils" ajoutez les contrôles suivants à la fiche WinForm.pas.

- Catégorie: Windows® Forms, 2 boutons.
- Catégorie: Data Controls, 3 DataGrids.
- Catégorie: Enterprise Core Objects, 3 ExpressionHandles.

Définissez les propriétés suivantes :

Type: Button

Name: btnAddPerson

Type: Button

Name: btnAddCompany

Type: ExpressionHandle

Name: ehContacts

Root: rhRoot

Expression: Contact.allinstances

Type: ExpressionHandle

Name: ehPersons

Root: rhRoot

Expression: person.allinstances

Type: ExpressionHandle

Name: ehCompanies

Root: rhRoot

Expression: Company.allinstances

Type: DataGrid

Name: dgContacts

DataSource: ehContacts

Type: DataGrid

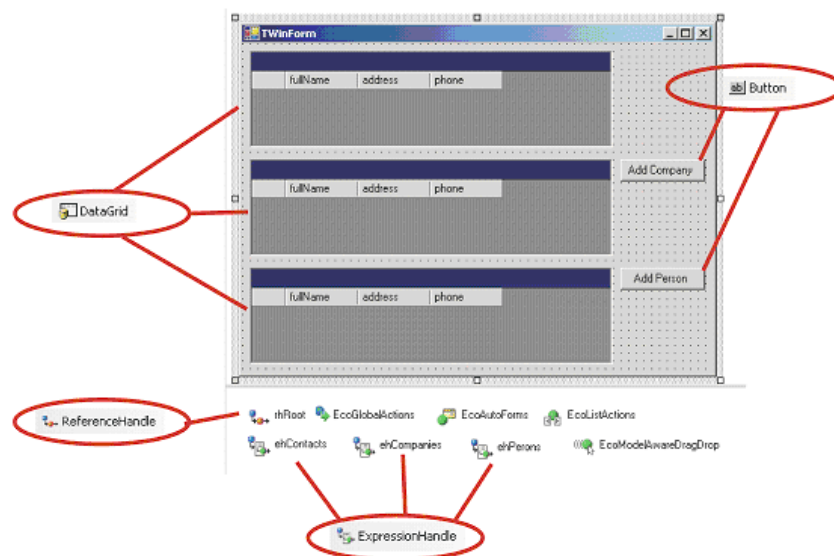
Name: dgPersons

DataSource: ehpersons

Type: DataGrid

Name: dgCompanies

DataSource: ehCompany



Entrez le code suivant pour les événements "OnClick" des boutons :

**Pour Delphi**

```

procedure TWinForm.btnAddPerson_Click(sender: System.Object;
  e: System.EventArgs);
begin
  Person.Create(EcoSpace);
end;

procedure TWinForm.BtnAddCompany_Click(sender: System.Object;
  e: System.EventArgs);
begin
  Company.Create(EcoSpace);
end;
    
```

**Pour C#**

```

private void btnAddperson_Click(object sender, System.EventArgs e)
{
  new person(EcoSpace);
}

private void btnAddCompany_Click(object sender, System.EventArgs e)
{
  new Company(EcoSpace);
}
    
```

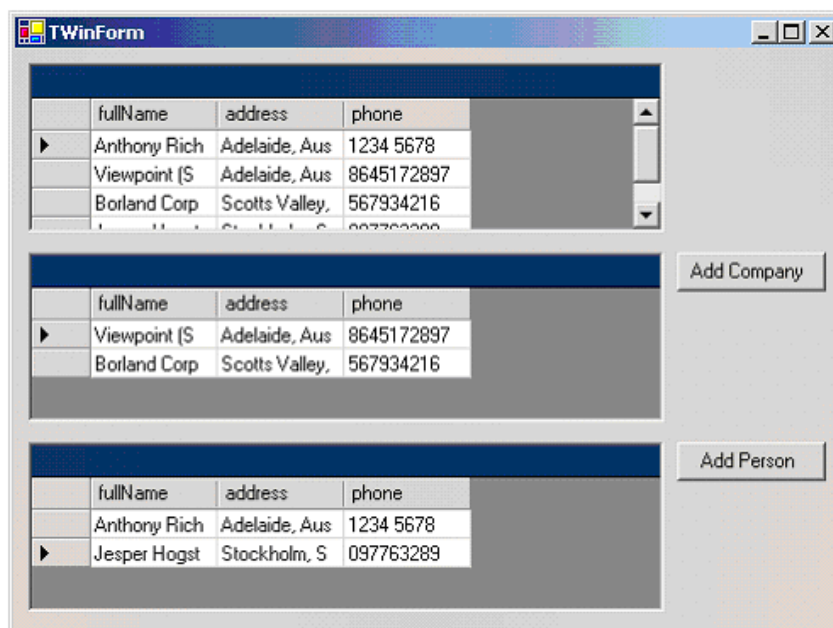
Le composant ReferenceHandle, qui a été placé sur la fiche par l'expert d'applications ECO, fournit le lien entre l'espace ECO et la Win Form. Un ReferencetHandle peut être défini pour référencer un objet spécifique dans un espace ECO ou pour l'espace ECO tout entier. C'est ainsi que le ReferenceHandle est utilisé dans cet exemple. Il est important que le ReferenceHandle utilise l'espace ECO que nous avons précédemment défini dans notre modèle ; définissez le paramètre suivant pour le composant rhRoot.

```
EcoSpaceType = ContactManagerEcoSpace.TContactManagerEcoSpace
```

Le composant ExpressionHandle est lié à un autre handle, soit un ExpressionHandle soit un Referencehandle, et évalue une expression OCL pour fournir un objet résultant, une collection ou valeur. Il est possible de chaîner les ExpressionHandles pour lesquels chaque handle évalue son résultat à partir des résultats du handle précédent.

## Exécuter l'application

Exécutez l'application et créez des personnes et des entreprises.



La grille paramétrée pour montrer "person.allinstances" affiche seulement les personnes créées. La grille paramétrée pour montrer "Company.allinstances" affiche seulement les entreprises créées. La grille paramétrée pour montrer "Contact.allinstances" affiche à la fois les personnes et les entreprises.

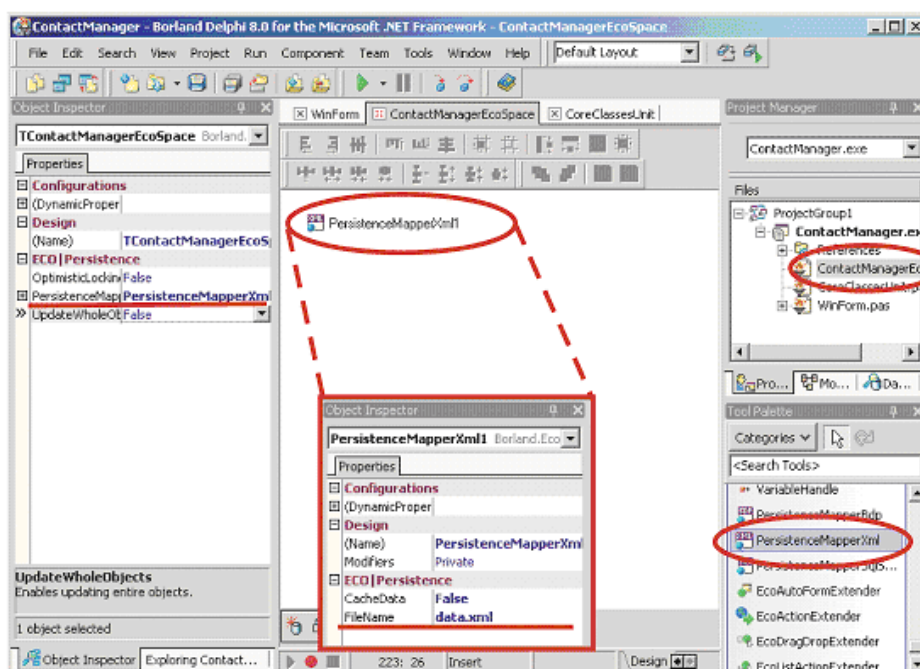
## Données persistantes

Une telle application voudra généralement sauvegarder les données entrées entre chaque invocation. Avec ECO, il y a deux méthodes de persistance supportées. Sauvegarder dans un fichier XML ou dans un SGBDR. Ici nous utiliserons un fichier XML, plus simple à configurer.

Depuis le "Gestionnaire de Projet" double cliquez sur ContactManagerEcoSpace.pas pour appeler l'éditeur de conception. Depuis la "Palette d'Outils" sélectionnez l'outil "persistenceMapperXML" et posez le dans le concepteur. Définissez les propriétés suivantes:

```
Type: persistenceMapperXML1  
FileName: data.xml
```

```
Type: EcoSpace  
persistenceMapper: persistenceMapperXML1
```



Dans la fiche WinForm.pas (ou WinForm.cs) , ajoutez un nouveau bouton. Nommez le "btnSave" et mettez "Save" dans le texte. Ajoutez le code suivant dans l'événement OnClick de btnSave.

### Pour Delphi

```
procedure TWinForm.btnSave_Click(sender: System.Object; e: System.EventArgs);  
begin  
    EcoSpace.UpdateDatabase;  
end;
```

### Pour C#

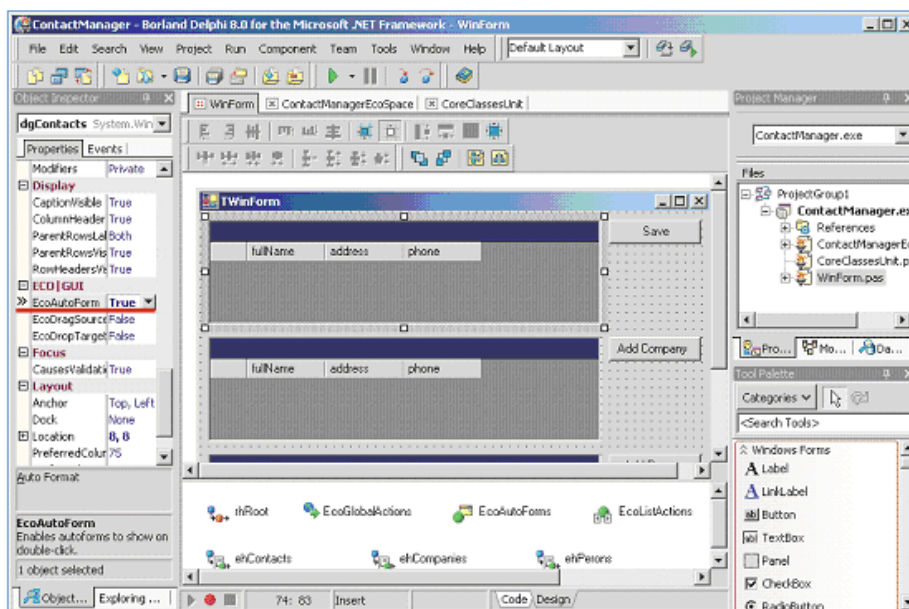
```
private void btnSave_Click(object sender, System.EventArgs e)  
{  
    EcoSpace.UpdateDatabase();  
}
```

Dorénavant lorsque vous exécutez l'application, la donnée que vous saisissez sera sauvegardée dans le fichier data.xml toutes les fois que vous presserez le bouton "Save".

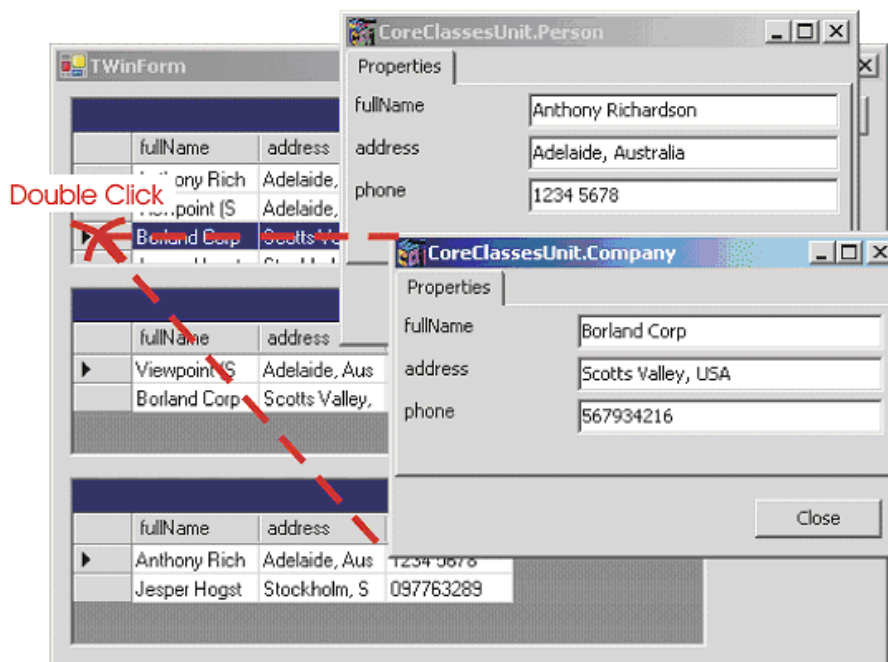
Il est intéressant de noter ici qu'il n'est pas nécessaire de spécifier dans le modèle d'informations relatives à la façon dont les données vont être sauvegardées. Cependant cela est possible si nécessaire.

## Fiches automatiques

ECO supporte la génération automatique de fiches pour éditer vos objets ECO à l'exécution. La Win Form ECO par défaut inclut un composant ECOAutoForms. Ce composant étend les contrôles orientés données de votre fiche pour supporter l'activation par double-clic des fiches automatiques. Définissez la propriété "ECOAutoForm" des trois grilles à "True".



A l'exécution, vous pouvez activer la fiche automatique en double cliquant la colonne fixe de chacune des grilles.



## Conclusion

ECO offre une façon de développer des applications dans laquelle le développeur passe plus de temps à coder la logique métier et moins de temps sur le codage de l'architecture. Ce qui permet d'avoir un code plus clair et réduit les coûts des changements. ECO fournit une séparation nette entre la présentation, la logique métier et la persistance. Ce qui réduit les effets que les changements ont sur l'application.