



Expert
EXPERT

C# 10

Développez
des applications Windows
avec **Visual Studio 2022**

En téléchargement

 code source des exemples

 + QUIZ

Version en ligne
OFFERTE !
pendant 1 an

Jérôme HUGON

eni



Les éléments à télécharger sont disponibles à l'adresse suivante :
<http://www.editions-eni.fr>
Saisissez la référence de l'ouvrage **EI10C22VIS** dans la zone de recherche et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

Avant-propos

Chapitre 1

Travailler avec Visual Studio 2022

- 1. Introduction 19
- 2. L'interface de développement 20
 - 2.1 L'éditeur de texte 22
 - 2.2 Le concepteur de vues 27
 - 2.3 Le débogueur intégré 28
 - 2.4 Le gestionnaire d'extensions 29
 - 2.5 NuGet 31
 - 2.6 Fenêtres personnalisées 33
- 3. La création de solutions 33
 - 3.1 Définir le point d'entrée 34
 - 3.2 La différence entre projets et solutions 36
 - 3.3 Configurer le projet 37
 - 3.4 La conversion de solutions 39
 - 3.5 Les projets partagés 39
 - 3.6 Les outils de refactorisation 41

Chapitre 2**L'architecture .NET**

1. Introduction	43
2. CLR	44
3. Les bibliothèques de classes	45
4. Les types	47
4.1 Les types valeur	48
4.2 Les types référence	49

Chapitre 3**Introduction au langage C#**

1. La syntaxe	51
1.1 Les identifiants	51
1.2 Les mots-clés	51
1.3 La ponctuation	53
1.4 Les opérateurs	54
1.4.1 Les opérateurs de calcul	54
1.4.2 Les opérateurs d'assignation	55
1.4.3 Les opérateurs de comparaison	55
1.5 La déclaration de variables	56
1.6 Les instructions de contrôle	57
1.6.1 Les instructions conditionnelles	57
1.6.2 Les instructions itératives	62
1.6.3 Les instructions de saut	65
1.7 Les commentaires	67
2. Les espaces de noms	70
2.1 Déclarer un espace de noms	71
2.2 Le mot-clé using	71
2.3 Le mot-clé alias	72
2.4 Les classes statiques	73
2.5 Les directives using globales	73

- 3. Les types de base. 73
 - 3.1 Les types numériques 73
 - 3.1.1 Les entiers. 74
 - 3.1.2 Les décimaux 75
 - 3.2 Les booléens 75
 - 3.3 Les chaînes de caractères 75
 - 3.4 Les types nullable 78
 - 3.5 La conversion de types 80
 - 3.5.1 La conversion implicite 80
 - 3.5.2 La conversion explicite 81
- 4. Les constantes et les énumérations 82
 - 4.1 Les constantes. 82
 - 4.2 Les énumérations 82
- 5. Les tableaux. 85
- 6. Les collections 86
- 7. Les directives preprocessor 88

Chapitre 4
La création de types

- 1. Introduction 91
- 2. Les niveaux d'accès 92
- 3. Les structures 93
- 4. Les classes 94
 - 4.1 Les champs 94
 - 4.2 Les propriétés 95
 - 4.3 Les méthodes. 98
 - 4.3.1 La surcharge 100
 - 4.3.2 Les paramètres 100
 - 4.3.3 Les tuples 107
 - 4.3.4 Les méthodes partielles 107
 - 4.4 Les constructeurs 108

4.5	Les destructeurs	109
4.6	Les classes et membres statiques	110
4.7	Les classes partielles	110
4.8	Le mot-clé this	112
4.9	Les indexeurs	113
4.10	Les attributs	114
4.11	La surcharge d'opérateurs	115
4.11.1	Les opérateurs arithmétiques	116
4.11.2	Les opérateurs de comparaison	118
5.	Les records	120

Chapitre 5

L'héritage

1.	L'héritage de classe	121
1.1	Implémenter l'héritage	121
1.2	Les membres virtuels	123
1.3	Masquer les membres hérités	124
1.4	Le mot-clé base	125
1.5	Les classes et membres abstraits	125
1.6	Les classes et les méthodes scellées	126
1.7	Les constructeurs dérivés	128
1.8	Le polymorphisme	130
2.	Les interfaces	132
2.1	L'implémentation d'interfaces	132
2.2	Le polymorphisme d'interface	134
2.3	L'héritage d'interfaces	136

Chapitre 6
Types génériques

- 1. Introduction 137
- 2. La création de types génériques 138
- 3. Les contraintes de type 141
- 4. Les interfaces génériques 142
 - 4.1 La variance dans les interfaces génériques 143
 - 4.1.1 La covariance 143
 - 4.1.2 La contravariance 144
 - 4.2 La création d'interfaces génériques variantes 145
 - 4.3 L'héritage d'interfaces génériques variantes 146
- 5. La création de méthodes génériques 147
- 6. Valeur par défaut générique 150
- 7. L'héritage de classe générique 150

Chapitre 7
Délégués, événements et expressions lambda

- 1. Les délégués 151
 - 1.1 Les paramètres de méthode 152
 - 1.2 Les méthodes cibles multiples 153
 - 1.3 Les délégués génériques 154
 - 1.4 La compatibilité des délégués 154
- 2. Les événements 156
- 3. Les expressions lambda 159
 - 3.1 L'utilisation des expressions lambda 160
 - 3.2 Les délégués génériques 162
 - 3.3 La capture de variable 162
 - 3.4 Les fonctions locales 165

Chapitre 8

Création de formulaires

1. Utiliser les formulaires	167
1.1 Ajouter des formulaires au projet	167
1.2 Modifier le formulaire de démarrage	170
1.3 Les propriétés des formulaires	170
1.4 Les méthodes des formulaires	173
1.5 Les événements des formulaires	174
2. Utiliser les contrôles	175
2.1 Les types de contrôles.	175
2.2 Ajouter des contrôles aux formulaires	177
2.3 Les propriétés des contrôles	179
2.4 Les menus	181
2.5 Les conteneurs	183
2.6 L'ergonomie.	185
2.7 Ajouter des contrôles à la boîte à outils	186

Chapitre 9

Implémentation de gestionnaires d'événements

1. Introduction	189
2. La création de gestionnaires d'événements	190
2.1 La mécanique d'un événement.	192
2.2 L'ajout dynamique d'un gestionnaire d'événements	192
2.3 La suppression dynamique d'un gestionnaire d'événements	193
3. Les gestionnaires d'événements avancés	194
3.1 Un gestionnaire pour plusieurs événements.	194
3.2 Plusieurs gestionnaires pour un événement	195

Chapitre 10
Validation de la saisie

- 1. Introduction 197
- 2. La validation au niveau des champs 197
 - 2.1 Les propriétés de validation 197
 - 2.2 Les événements de validation 198
 - 2.2.1 KeyDown et KeyUp 198
 - 2.2.2 KeyPress 199
 - 2.2.3 Validating et Validated 199
- 3. La validation au niveau du formulaire. 201
- 4. Les méthodes de retour à l'utilisateur 204
 - 4.1 MessageBox. 205
 - 4.2 ErrorProvider. 206

Chapitre 11
Création de contrôles utilisateurs

- 1. Introduction 209
- 2. Les contrôles personnalisés 210
- 3. L'héritage de contrôles 212
- 4. Les contrôles utilisateurs 214

Chapitre 12
Création d'applications UWP

- 1. Introduction 221
- 2. Principes 222
- 3. Les outils de développement 224
- 4. Le langage XAML 226

5. Une première application UWP.....	229
5.1 Les bases d'un projet UWP.....	229
5.2 Les contrôles et événements	231
5.3 Les styles.....	232

Chapitre 13

Débogage

1. Les types d'erreurs.....	237
1.1 Les erreurs de syntaxe	237
1.2 Les erreurs d'exécution.....	238
1.3 Les erreurs de logique.....	240
2. Le débogueur.....	240
2.1 Contrôler l'exécution.....	242
2.2 Les points d'arrêt.....	243
2.2.1 Les conditions d'arrêt.....	244
2.2.2 Le nombre d'accès.....	245
2.2.3 Le filtrage.....	246
2.2.4 Les actions.....	246
2.2.5 Exécuter l'exécution jusqu'ici.....	247
2.3 Les DataTips.....	247
2.4 Les PerfTips.....	248
2.5 Les attributs Caller.....	249
3. Les fenêtres.....	251
3.1 La fenêtre Sortie.....	252
3.2 La fenêtre Variables locales.....	252
3.3 La fenêtre Automatique.....	253
3.4 La fenêtre Espion.....	253
3.5 La fenêtre Exécution.....	254
3.6 Les autres fenêtres.....	255

Chapitre 14
Gestion des exceptions

- 1. La classe Exception 257
- 2. La création d'exceptions personnalisées 258
- 3. Le déclenchement des exceptions 259
- 4. L'interception et la gestion des exceptions 262

Chapitre 15
Monitoring

- 1. Le traçage. 269
 - 1.1 Les classes Debug et Trace 269
 - 1.2 La collection d'écouteurs 272
 - 1.2.1 La création d'écouteurs 272
 - 1.2.2 La sauvegarde des traces 273
 - 1.3 Les commutateurs de trace 275
 - 1.3.1 Le fonctionnement des commutateurs de trace 275
 - 1.3.2 La configuration des commutateurs de trace. 276
- 2. Les journaux d'événements 277
 - 2.1 L'interaction avec les journaux d'événements. 278
 - 2.2 La gestion des journaux d'événements 279
 - 2.3 L'écriture d'événements 280
- 3. Les compteurs de performance 281
 - 3.1 La création de compteurs de performance 282
 - 3.1.1 Depuis Visual Studio. 282
 - 3.1.2 Depuis le code 283
 - 3.2 L'utilisation de compteurs de performance. 285
 - 3.3 L'analyse de compteurs de performance 288

Chapitre 16**Tests unitaires**

1. Introduction aux tests unitaires	291
1.1 La création du projet	291
1.2 Les classes de tests unitaires	292
2. La mise en place d'une série de tests	294
2.1 La création de tests au projet	294
2.2 Le déroulement des tests	295

Chapitre 17**Création du modèle de données**

1. Introduction	299
2. La création d'un modèle	300
3. La création d'entités	302
4. La génération de la base de données	308
5. La création d'entités à partir du code (Code First)	313

Chapitre 18**Présentation d'Entity Framework**

1. Introduction	319
2. Le mappage	320
2.1 La couche logique	320
2.2 La couche conceptuelle	322
2.3 La couche de mappage	325
3. Travailler avec les entités	326
3.1 Les entités	328
3.2 La classe DbContext	329
3.3 Les relations	330
3.3.1 Le concept de table par type	330
3.3.2 Le concept de table par hiérarchie	330

Chapitre 19
Présentation de LINQ

- 1. Les requêtes LINQ 333
 - 1.1 La syntaxe 333
 - 1.2 Les méthodes d'extension. 334
- 2. Les opérateurs de requêtes 336
 - 2.1 Filtrer 336
 - 2.1.1 Where 336
 - 2.1.2 OfType<TResult> 336
 - 2.1.3 SelectMany 337
 - 2.1.4 Skip et Take 337
 - 2.2 Ordonner 338
 - 2.2.1 OrderBy 338
 - 2.2.2 ThenBy 339
 - 2.3 Grouper 339
 - 2.3.1 GroupBy 339
 - 2.3.2 Join 340
 - 2.4 Agréger 341
 - 2.5 Convertir 341
- 3. Les requêtes parallèles 342
 - 3.1 Partitionner une requête 343
 - 3.2 Annuler une requête. 344

Chapitre 20
LINQ to Entities

- 1. Introduction 345
- 2. Extraire les données 346
 - 2.1 L'extraction simple 346
 - 2.2 L'extraction conditionnelle 347

3.	Ajouter, modifier et supprimer des données	349
3.1	Ajouter des données	349
3.2	Modifier des données	350
3.3	Supprimer des données	350
3.4	L'envoi des modifications	350

Chapitre 21

LINQ to SQL

1.	La création de classes LINQ to SQL	351
2.	L'objet DataContext	354
2.1	La méthode ExecuteQuery	355
2.2	Utiliser des transactions	355
2.3	Les autres membres de DataContext	356
3.	Exécuter des requêtes avec LINQ	357
3.1	Les requêtes simples	357
3.2	Les requêtes filtrées	358
3.3	Les requêtes de jointure	358
4.	Les procédures stockées	358
4.1	L'ajout de procédures stockées au modèle	359
4.2	L'exécution de procédures stockées	360

Chapitre 22

LINQ to XML

1.	Les objets XML	361
1.1	XDocument	361
1.2	XElement	362
1.3	XNamespace	363
1.4	XAttribute	364
1.5	XComment	364

- 2. Exécuter des requêtes avec LINQ 365
 - 2.1 Les requêtes simples 365
 - 2.2 Les requêtes filtrées 366
 - 2.3 Les requêtes de jointure 366

Chapitre 23
Le système de fichiers

- 1. Les classes de gestion du système de fichiers 367
 - 1.1 DriveInfo 367
 - 1.2 Directory et DirectoryInfo 369
 - 1.3 File et FileInfo 371
 - 1.4 Path 374
- 2. Travailler avec le système de fichiers 377
 - 2.1 Les objets Stream 377
 - 2.2 La classe FileStream 377
 - 2.3 Lire un fichier texte 379
 - 2.3.1 Lire avec la classe File 379
 - 2.3.2 Lire avec la classe StreamReader 380
 - 2.4 Écrire dans un fichier texte 382
 - 2.4.1 Écrire avec la classe File 382
 - 2.4.2 Écrire avec la classe StreamWriter 383

Chapitre 24
Sérialisation

- 1. Introduction 385
- 2. La sérialisation binaire 386
 - 2.1 Les bases 386
 - 2.2 Contrôler la sérialisation 388
 - 2.2.1 Le contrôle par attribut 388
 - 2.2.2 Le contrôle par interface 390

3. La sérialisation XML	393
3.1 Les bases	394
3.2 Contrôler la sérialisation	397
3.3 La sérialisation XML SOAP	398

Chapitre 25

Expressions régulières

1. Introduction	401
2. Une première expression régulière.	402
3. Les options de recherche	403
4. Les caractères d'échappement	404
5. Les ensembles	404
6. Les groupes	406
7. Les ancres	407
8. Les quantifieurs.	408

Chapitre 26

Multithreading

1. Introduction	409
2. La classe Thread	410
2.1 Créer un thread.	410
2.2 Suspendre un thread.	411
2.3 Échanger des données avec un thread.	412
2.4 Verrouiller un thread	414
2.5 Priorité des threads.	415
3. Fonctions asynchrones.	416
3.1 Task et Task<TResult>	417
3.2 async et await.	419
4. Le composant BackgroundWorker	421

Chapitre 27
Globalisation et localisation

- 1. Introduction 425
- 2. La culture..... 426
- 3. La globalisation..... 428
- 4. La localisation 430

Chapitre 28
Sécurité

- 1. Introduction 435
- 2. Les éléments de base..... 436
 - 2.1 L'interface IPermission 436
 - 2.2 La classe CodeAccessPermission 437
 - 2.3 L'interface IPrincipal..... 437
- 3. Implémentation de la sécurité..... 439
 - 3.1 La sécurité basée sur les rôles..... 439
 - 3.2 La sécurité basée sur les droits d'accès..... 441
 - 3.2.1 Sécurité impérative 441
 - 3.2.2 Sécurité déclarative 442
- 4. Introduction à la cryptographie..... 443

Chapitre 29
Pour aller plus loin

- 1. Le dessin avec GDI+ 447
 - 1.1 La classe Graphics..... 448
 - 1.1.1 Les coordonnées..... 448
 - 1.1.2 Les formes 449
 - 1.2 La structure Color et les classes Brush et Pen..... 451
 - 1.2.1 La structure Color 451
 - 1.2.2 La classe Brush..... 452

1.2.3	La classe Pen	452
1.2.4	Les paramètres système	453
1.3	Les exemples	453
1.3.1	L'affichage de texte	453
1.3.2	Redimensionner une image	455
2.	Le remoting	456
2.1	Le principe	456
2.2	L'implémentation	457
2.2.1	La couche commune	457
2.2.2	L'application serveur	458
2.2.3	L'application cliente	460
3.	Reflection	464
3.1	La classe System.Type	464
3.2	Charger un assemblage dynamiquement	466
3.2.1	L'énumération des types	466
3.2.2	L'instanciation d'objets	467
3.2.3	L'utilisation des membres	468

Chapitre 30

Assemblages et configurations

1.	Introduction	471
2.	Les assemblages privés	472
3.	Les assemblages partagés	475
4.	Les fichiers de configuration	476

Chapitre 31
Déploiement

- 1. Introduction 479
- 2. Les projets de déploiement..... 480
 - 2.1 XCOPY 480
 - 2.2 Projet CAB..... 481
 - 2.3 Projet de module de fusion..... 482
 - 2.4 Projet d'installation..... 482
- 3. L'assistant Installation 483
- 4. Configuration du projet 487
 - 4.1 Les propriétés du projet 487
 - 4.2 Les éditeurs de configuration..... 490
 - 4.2.1 Éditeur du système de fichiers 491
 - 4.2.2 Éditeur du registre 492
 - 4.2.3 Éditeur des types de fichiers 493
 - 4.2.4 Éditeur de l'interface utilisateur 495
 - 4.2.5 Éditeur des actions personnalisées 497
 - 4.2.6 Éditeur des conditions de lancement 498

- Index 501

Chapitre 4

La création de types

1. Introduction

Les classes représentent la majorité des types référence. La définition la plus simple d'une classe sera :

```
class MaClasse  
{  
}
```

Au fur et à mesure de la construction d'une classe, des éléments sont ajoutés :

- Les membres (méthodes, propriétés, indexeurs, événements...) sont placés entre les accolades.
- Les attributs et les modificateurs de classe comme le niveau d'accès sont placés avant le mot-clé `class`.
- L'héritage et les implémentations d'interfaces sont placés après le nom de la classe.

2. Les niveaux d'accès

Les niveaux d'accès permettent de définir comment vont pouvoir s'effectuer l'instanciation des types et l'appel des méthodes. Le niveau d'accès est défini à l'aide de mots-clés précédant la déclaration de la classe, ou du membre. Le tableau suivant présente les modificateurs d'accès disponibles :

Modificateur d'accès	Description
<code>public</code>	Autorise l'accès pour tous les types de l'assemblage et hors de l'assemblage.
<code>private</code>	Autorise l'accès uniquement pour les autres membres du type.
<code>internal</code>	Autorise l'accès pour tous les types de l'assemblage uniquement.
<code>protected</code>	Autorise l'accès uniquement pour les autres membres du type ou pour les types héritant de celui-ci même en dehors de l'assemblage.
<code>protected internal</code>	Autorise l'accès uniquement pour les autres membres du type ou pour les types héritant de celui-ci dans l'assemblage uniquement.

Si aucun modificateur d'accès n'est précisé sur un membre, il est considéré comme `private`. Une classe ou une structure sans modificateur d'accès sera considérée comme `public`.

Les membres ne pourront jamais étendre leur niveau d'accès au-delà de celui du type contenant. Cela signifie que même si un membre est marqué avec le modificateur d'accès `public`, et que la classe dans laquelle il se trouve est marquée comme `internal`, le membre ne sera accessible que pour les types de l'assemblage :

```
internal class MaClasse
{
    // Le membre est accessible depuis l'assemblage uniquement
    public int i;
}
```

Même si des membres sont marqués comme `internal`, il est possible de les exposer à d'autres assemblages. Il suffit d'ajouter un attribut du type `System.Runtime.CompilerServices.InternalsVisibleTo` en spécifiant le nom de l'assemblage dans le fichier **AssemblyInfo.cs** comme ceci :

```
[assembly: InternalsVisibleTo("Assemblage")]
```

Si l'assemblage à autoriser est signé avec un nom fort, vous pouvez spécifier son nom complet :

```
[assembly: InternalsVisibleTo("Assemblage, Version=1.0.0.0, Culture=fr, PublicKeyToken=26381116d3a4ad13")]
```

3. Les structures

Les structures sont très similaires aux classes avec, comme principales différences, le fait qu'une structure est un type valeur alors qu'une classe est un type référence. Les structures sont utilisées à la place des classes quand la sémantique exige un type valeur. Une structure ne supporte pas l'héritage. Elles peuvent posséder tous les membres d'une classe à l'exception d'un constructeur sans paramètre, d'un destructeur et de membres virtuels.

Voici l'exemple de la définition d'une structure :

```
struct GeoPoint
{
    double Longitude;
    double Latitude;
}
```

La déclaration et l'instanciation d'une occurrence de la structure se font comme pour une classe. Un constructeur sans paramètre existe implicitement :

```
GeoPoint g = new GeoPoint();
```

Il est possible de surcharger ce constructeur pour initialiser les membres de la structure mais il faut noter que, lors de l'utilisation du mot-clé `default`, l'initialisation affectera les valeurs par défaut des types aux membres (dans l'exemple ci-dessous, 0 pour les types `double`) :

```
public GeoPoint()
{
    this.Longitude = 1;
```

```
        this.Latitude = 2;
    }
    var g1 = new GeoPoint();
    // g1.Longitude = 1
    // g1.Latitude = 2

    var g2 = default(GeoPoint);
    // g2.Longitude = 0
    // g2.Latitude = 0
```

Il est possible de rajouter son propre constructeur à partir du moment où tous les champs de la structure sont initialisés :

```
Public GeoPoint(double longitude, double latitude)
{
    this.Longitude = longitude;
    this.Latitude = latitude;
}
```

Les champs peuvent être initialisés lors de leur déclaration dans la structure :

```
struct GeoPoint
{
    double Longitude = 1;
    double Latitude;
}
```

4. Les classes

4.1 Les champs

Un champ est une variable qui est un membre de la classe. Il peut s'agir de type valeur ou de type référence.

À la racine du projet, créez un dossier nommé **Library** et créez une nouvelle classe nommée `Project`. Ajoutez les champs suivants :

```
public class Project
{
    protected string filename = "sans titre.smpx", path;
    protected DataTable data = new DataTable();
```

```
protected bool hasChanged;  
}
```

Les champs peuvent être initialisés au moment de la déclaration. Un champ qui n'est pas initialisé explicitement recevra les valeurs par défaut suivant son type. L'initialisation des champs est effectuée avant l'exécution du constructeur de la classe.

Il est également possible de déclarer et initialiser plusieurs champs en une seule instruction s'ils ont le même niveau d'accès et le même type :

```
private string filename = "sans titre.smpx", path;
```

Le mot-clé `readonly` permet de spécifier qu'un champ sera en lecture seule, il pourra seulement être assigné lors de la déclaration ou lors de l'instanciation, au sein du constructeur :

```
public readonly int i = 1;
```

4.2 Les propriétés

Les propriétés ressemblent à des champs puisqu'on y accède de la même manière mais leur logique interne les rapproche des méthodes.

Une propriété est déclarée de la même manière qu'un champ en ajoutant des blocs `get` et `set`. Ces deux blocs sont appelés des accesseurs ; l'accesseur `get` est exécuté lorsque la propriété est lue et doit retourner une valeur du type de la propriété. L'accesseur `set` est exécuté lorsque la propriété est assignée. Un paramètre implicite accessible via le mot-clé `value` du type de la propriété est fourni.

Utilisez les outils de refactorisation de Visual Studio (`[Ctrl] R, E` avec le curseur sur le nom d'un champ) pour générer les propriétés des champs précédemment créés dans la classe `Project`. Le code généré est le suivant :

```
public string Filename  
{  
    get { return filename; }  
    set { filename = value; }  
}  
public string Path  
{
```

```
        get { return path; }
        set { path = value; }
    }
    public DataTable Data
    {
        get { return data; }
        set { data = value; }
    }
    public bool HasChanged
    {
        get { return hasChanged; }
        set { hasChanged = value; }
    }
}
```

Il est possible de créer des propriétés automatiques au lieu de créer un champ puis une propriété avec des accesseurs qui ont pour seul but de lire et écrire dans un champ privé :

```
public int i { get; set; }
```

Ce type de déclaration indique au compilateur de générer automatiquement un champ privé de la propriété qui lui servira pour stocker les valeurs.

Depuis la version 6 de C#, il est possible de spécifier uniquement un accesseur `get` dans la déclaration de la propriété. Le compilateur crée alors un champ privé en lecture seule. Il est également possible d'initialiser la propriété directement sans passer par un constructeur :

```
public int i { get; } = 10;
```

Les accesseurs peuvent être marqués avec différents niveaux d'accès. Ainsi, l'accesseur `set` pourra être marqué par le mot-clé `private` afin d'exposer la propriété en lecture seule :

```
public int i { get; private set; }
```

Un accesseur possède, par défaut, le même niveau d'accès que celui qui marque la propriété.

Il est possible de créer une propriété immuable (dont la valeur est fixée à l'instanciation de l'objet) en remplaçant le mot-clé `set` par le mot-clé `init`. Cela aura pour effet de pouvoir éviter l'écriture d'un constructeur avec la syntaxe suivante :

```
MyObject o = new() { i = 10 } ;
```

Vous pouvez utiliser un modèle de propriété pour déterminer sa valeur en fonction de celle des membres imbriqués :

```
static bool ValeurNonZero(GeoPoint g) => g is not { X: 0, Y: 0 } ;
```

Et également en fonction des propriétés des membres imbriqués :

```
public struct Position
{
    public GeoPoint Longitude;
    public GeoPoint Latitude;
}
static bool ValeurNonZero(Position p) => p is not
{ Longitude.X: 0, Longitude.Y: 0 }
and
{ Latitude.X: 0, Latitude.Y: 0 } ;
```

La classe `Project` contient une propriété `HasChanged` de type booléen. Elle doit refléter le fait que l'objet a été modifié ou non depuis la dernière sauvegarde. Pour ce faire, ajoutez le code permettant de mettre à jour le champ `HasChanged` lorsque les propriétés `Filename`, `Path` et `Data` sont modifiées. Voici l'exemple avec la propriété `Filename` :

```
public string Filename
{
    get { return filename; }
    protected set
    {
        if (this.filename != value)
        {
            this.filename = value;
            this.HasChanged = true;
        }
    }
}
```