Les outils du développeur

Microsoft®

MICROSOFT®

VISUAL C#® 2010

ÉTAPE PAR ÉTAPE

John Sharp

Adapté de l'américain par
Dominique Maniez

Les programmes figurant dans ce livre, et éventuellement sur la disquette ou le CD-ROM d'accompagnement, sont fournis gracieusement sous forme de code source, à titre d'illustration. Ils sont fournis en l'état sans garantie aucune quant à leur fonctionnement une fois compilés, assemblés ou interprétés dans le cadre d'une utilisation professionnelle ou commerciale. Ils peuvent nécessiter des adaptations et modifications dépendant de la configuration utilisée. Microsoft Press ne pourra en aucun cas être tenu responsable des préjudices ou dommages de quelque nature que ce soit pouvant résulter de l'utilisation de ces programmes.

Tous les efforts ont été faits pour fournir dans ce livre une information complète et exacte à la date de la parution. Néanmoins, Microsoft Press n'assume de responsabilités ni pour son utilisation, ni pour les contrefaçons de brevets ou atteintes aux droits de tierces personnes qui pourraient résulter de cette utilisation.

Édition et diffusion de la version française : Dunod
Distribution : Hachette Livre Distribution
Couverture : Agence SAMOA
Adapté de l'américain par : Dominique Maniez
Mise en page : Arclemax
ISBN : 978-2-10-056094-3

Sommaire

Introduction I

À qui s'adresse ce manuel I
Recherche de votre meilleur point de départ dans ce manuel II
Assistance technique III

Partie I **Introduction à Microsoft Visual C# et Microsoft Visual Studio 2010**

1 Bienvenue dans l'univers de C# **3**

Débuter dans l'environnement Visual Studio 2010 3
Écrire votre premier programme 8
Utiliser des espaces de noms 14
Créer une application graphique 17
Aide-mémoire du chapitre 1 27

2 Variables, opérateurs et expressions **29**

Instructions 29
Identificateurs 30
Mots-clés 30
Variables 31
Noms des variables 31
Déclaration des variables 32
Types de données primitifs 33
Variables locales non affectées 34
Affichage des valeurs des types de données primitifs 34
Opérateurs arithmétiques 38
Opérateurs et types 38
Travailler avec des opérateurs arithmétiques 40
Contrôle de la priorité 43
Associativité pour évaluer des expressions 44
Associativité et opérateur d'assignation 44
Incrémenter et décrémenter des variables 45
Préfixes et postfixes 46
Déclarer implicitement des variables locales typées 47
Aide-mémoire du chapitre 2 48

3 Écrire des méthodes et définir leur portée 49

Création des méthodes . . . 49
Déclaration de méthode . . . 49
Retour d'informations à partir d'une méthode . . . 51
Appel des méthodes . . . 53
Syntaxe d'un appel de méthode . . . 53
Définition d'une portée . . . 55
Définition d'une portée locale . . . 56
Définition d'une portée de classe . . . 56
Surcharge des méthodes . . . 57
Écriture des méthodes . . . 58
Utilisation de paramètres optionnels et d'arguments nommés . . . 65
Définition de paramètres optionnels . . . 67
Passage d'arguments nommés . . . 68
Résolution des ambiguïtés lors de l'utilisation des paramètres optionnels et des arguments nommés . . . 68
Aide-mémoire du chapitre 3 . . . 74

4 Commandes de prise de décision 75

Déclaration des variables *bool* . . . 75
Opérateurs booléens . . . 76
Opérateurs d'égalité et opérateurs relationnels . . . 76
Opérateurs logiques conditionnels . . . 77
Court-circuit . . . 78
Priorité et associativité des opérateurs . . . 78
Instructions if pour prendre des décisions . . . 79
Syntaxe de l'instruction *if* . . . 79
Blocs pour regrouper des instructions . . . 80
Imbriquer des instructions *if* . . . 81
Instructions *switch* . . . 86
Syntaxe de l'instruction *switch* . . . 87
Règles de l'instruction *switch* . . . 88
Aide-mémoire du chapitre 4 . . . 91

5 Assignation composée et instructions d'itération 93

Opérateurs d'assignation composée . . . 93
Instructions *while* . . . 94
Instructions *for* . . . 99
Portée des instructions *for* . . . 101

Instructions *do* 101
Aide-mémoire du chapitre 5 110

6 Erreurs et exceptions 111

Gestion des erreurs 111
Blocs *Try* et interception des exceptions 112
Exceptions non gérées. 113
Utilisation de plusieurs gestionnaires *catch* 114
Interception de plusieurs exceptions 115
Mots-clés *checked* et *unchecked* pour l'arithmétique sur les entiers. . . . 119
Instructions *checked*. 120
Expressions *checked* 121
Levée des exceptions 123
Bloc *finally* 127
Aide-mémoire du chapitre 6 128

Partie II Comprendre le langage C#

7 Classes et objets 133

Classification 133
Encapsulation. 134
Définition et utilisation d'une classe. 134
Contrôle de l'accessibilité. 136
Constructeurs. 137
Surcharge des constructeurs 139
Méthodes et données *static*. 146
Champ partagé. 147
Création d'un champ statique avec le mot-clé *const* 148
Classes *static* 148
Classes anonymes 151
Aide-mémoire du chapitre 7 153

8 Valeurs et références 155

Copie des variables de type valeur et des classes 155
Valeurs null et types nullable. 160
Types nullable. 161
Propriétés des types nullable 162
Paramètres *ref* et *out*. 162
Paramètres *ref*. 163
Paramètres *out* 164

Organisation de la mémoire de l'ordinateur 166
Utilisation de la pile et du tas 167
Classe *System.Object* 168
Conversion boxing 169
Conversion unboxing 170
Cast des données en toute sécurité 171
Opérateur *is* 171
Opérateur *as* 172
Aide-mémoire du chapitre 8 174

9 Création de types valeur avec des énumérations et des structures 177

Énumérations 177
Déclaration d'une énumération 177
Utilisation d'une énumération 178
Choix des valeurs littérales d'énumération 179
Choix d'un type sous-jacent d'énumération 180
Structures 182
Déclaration d'une structure 183
Comprendre les différences entre structure et classe 184
Déclaration des variables structure 186
Initialisation d'une structure 187
Copie des variables structure 191
Aide-mémoire du chapitre 9 194

10 Tableaux et collections 195

Qu'est-ce qu'un tableau ? 195
Déclaration des variables tableau 195
Création d'une instance de tableau 196
Initialisation des variables tableau 197
Création d'un tableau implicitement typé 198
Accès à un élément de tableau 199
Parcours de tableau 199
Copie de tableaux 201
Tableaux multidimensionnels 202
Utilisation de tableaux pour jouer aux cartes 203

Classes de collection 210
Classe de collection *ArrayList* 211
Classe de collection *Queue* 213
Classe de collection *Stack* 214
Classe de collection *Hashtable* 215
Classe de collection *SortedList* 216
Initialiseurs de collection. 217
Comparaison des tableaux et des collections 218
Utilisation de classes de collection pour jouer aux cartes 218
Aide-mémoire du chapitre 10 221

11 Tableaux de paramètres 223

Arguments de tableau 224
Déclaration d'un tableau *params* 225
Params object[ ] 227
Tableau *params* 228
Comparaison des tableaux de paramètres et des paramètres optionnels 231
Aide-mémoire du chapitre 11 234

12 Héritage 235

Qu'est-ce que l'héritage ? 235
Utilisation de l'héritage. 236
Appel des constructeurs de classe de base 238
Assignation des classes 239
Déclaration des méthodes *new* 241
Déclaration des méthodes virtuelles. 242
Déclaration des méthodes *override*. 243
Accès *protected*. 246
Méthodes d'extension. 252
Aide-mémoire du chapitre 12 255

13 Interfaces et classes abstraites 257

Interfaces 257
Définition d'une interface. 258
Implémentation d'une interface 259
Référence à une classe par le biais de son interface. 260
Interfaces multiples 261

Implémentation explicite d'une interface 261
Restrictions d'une interface 263
Définition et utilisation des interfaces 264
Classes abstraites 273
Méthodes abstraites 274
Classes sealed 274
Méthodes sealed 275
Implémentation et utilisation d'une classe abstraite 275
Aide-mémoire du chapitre 13 281

14 Garbage collection et gestion des ressources 283

Durée de vie d'un objet 283
Écriture des destructeurs 285
Pourquoi utiliser le garbage collector ? 286
Comment fonctionne le garbage collector ? 287
Recommandations 288
Gestion des ressources 289
Méthodes d'élimination 289
Libération avec gestion des exceptions 290
Instruction *using* 290
Appel de la méthode *Dispose* à partir d'un destructeur 292
Sécurisation des exceptions 293
Aide-mémoire du chapitre 14 296

Partie III Créer des composants

15 Implémentation de propriétés pour accéder à des champs 301

Encapsulation à l'aide de méthodes 301
Concept de propriété 303
Utilisation des propriétés 305
Propriétés en lecture seule 306
Propriétés en écriture seule 306
Accessibilité des propriétés 307
Restrictions des propriétés 308
Déclaration des propriétés d'interface 310
Utilisation des propriétés dans une application Windows 311
Génération automatique de propriétés 313
Initialisation d'objets avec des propriétés 314
Aide-mémoire du chapitre 15 318

16 Indexeurs 321

Qu'est-ce qu'un indexeur ? . . . 321
Exemple n'utilisant pas d'indexeurs . . . 321
Exemple identique utilisant des indexeurs . . . 323
Accesseurs d'indexeur . . . 325
Comparaison entre les indexeurs et les tableaux . . . 326
Indexeurs dans les interfaces . . . 328
Indexeurs dans une application Windows . . . 329
Aide-mémoire du chapitre 16 . . . 335

17 Interruption du flux du programme et gestion des événements 337

Déclaration et utilisation des délégués . . . 337
Scénario d'une usine automatisée . . . 338
Implémentation de l'usine sans utiliser des délégués . . . 338
Implémentation de l'usine en utilisant un délégué . . . 339
Utilisation des délégués . . . 342
Expressions lambda et délégués . . . 347
Création d'un adaptateur de méthode . . . 347
Utilisation d'une expression lambda comme adaptateur . . . 348
Formes des expressions lambda . . . 349
Activation des notifications grâce aux événements . . . 351
Déclaration d'un événement . . . 351
Abonnement à un événement . . . 352
Désabonnement à un événement . . . 352
Déclenchement d'un événement . . . 353
Événements d'interface utilisateur WPF . . . 353
Utilisation des événements . . . 355
Aide-mémoire du chapitre 17 . . . 359

18 Introduction aux génériques 363

Le problème avec les *objets* . . . 363
La solution des génériques . . . 365
Génériques versus classes généralisées . . . 367
Génériques et contraintes . . . 368
Création d'une classe générique . . . 368
Théorie des arbres binaires . . . 368
Génération d'une classe d'arbre binaire à l'aide de génériques . . . 371

Création d'une méthode générique 380
Définition d'une méthode générique pour générer un arbre binaire 381
Variance et interfaces génériques 384
Interfaces covariantes 385
Interfaces contravariantes 387
Aide-mémoire du chapitre 18 389

19 Énumération des collections 391

Énumération des éléments d'une collection 391
Implémentation manuelle d'un énumérateur 393
Implémentation de l'interface *IEnumerable* 397
Implémentation d'un énumérateur en utilisant un itérateur 399
Itérateur simple 400
Définition d'un énumérateur pour la classe *Arbre<TElement>* en utilisant un itérateur 401
Aide-mémoire du chapitre 19 404

20 Recherche des données en mémoire avec des expressions de requête 405

Qu'est-ce que le LINQ (*Language Integrated Query*) ? 405
Utilisation de LINQ dans une application C# 406
Sélection des données 408
Filtrage des données 411
Classement, regroupement, et agrégation des données 411
Jointure des données 414
Opérateurs de requête 415
Interrogation des données des objets Arbre<TElement> 417
LINQ et l'évaluation différée 422
Aide-mémoire du chapitre 20 426

21 Surcharge des opérateurs 429

Opérateurs 429
Contraintes des opérateurs 430
Opérateurs surchargés 430
Opérateurs symétriques 432
Évaluation d'assignation composée 434
Déclaration des opérateurs d'incrémentation et de décrémentation . . . 435
Opérateurs de comparaison dans les structures et les classes 436

Définition des paires d'opérateurs 437
Implémentation des opérateurs 438
Opérateurs de conversion 444
Conversions intégrées 444
Opérateurs de conversion définis par l'utilisateur 445
Retour sur la création d'opérateurs symétriques 446
Écriture d'opérateurs de conversion 447
Aide-mémoire du chapitre 21 449

Partie IV Créer des applications Windows Presentation Foundation

22 Introduction à *Windows Presentation Foundation* 453

Création d'une application WPF 453
Création de l'application WPF 454
Ajout de contrôles au formulaire 468
Utilisation de contrôles WPF 468
Modification des propriétés de façon dynamique 478
Gestion des événements d'un formulaire WPF 482
Traitement des événements de Windows Forms 482
Aide-mémoire du chapitre 22 488

23 Saisie utilisateur 491

Interface des menus 491
Menus et événements de menu 492
Création d'un menu 493
Gestion des événements de menu 499
Menus contextuels 505
Création de menus contextuels 505
Boîtes de dialogue courantes 509
Utilisation de la classe *SaveFileDialog* 510
Amélioration de la réactivité d'une application WPF 513
Aide-mémoire du chapitre 23 523

24 Validation 525

Validation des données 525
Stratégies pour valider la saisie de l'utilisateur 525
Exemple : commande de billets de spectacles 526
Validation grâce à la liaison de données 527

Modification du moment où la validation a lieu 544
Aide-mémoire du chapitre 24 548

Partie V Gestion des données

25 Recherche d'informations dans une base de données 553

Interrogation d'une base de données avec ADO.NET 553
Base de données Northwind 554
Création de la base de données. 555
Utilisation d'ADO.NET pour interroger les informations sur les commandes 556
Interrogation d'une base de données avec LINQ to SQL 567
Définition d'une classe de données. 568
Création et exécution d'une requête LINQ to SQL 570
Extraction immédiate et différée 572
Jointure de tables et création de relations 573
Retour sur l'extraction immédiate et différée 577
Définition d'une classe *DataContext* personnalisée 578
Utilisation de LINQ to SQL pour interroger les informations sur les commandes. 579
Aide-mémoire du chapitre 25 583

26 Affichage et édition de données avec Entity Framework et la liaison de données 585

Utilisation de la liaison de données avec Entity Framework. 586
Utilisation de la liaison de données pour modifier les données 604
Mise à jour des données existantes. 604
Gestion des mises à jour conflictuelles 605
Ajout et suppression de données 608
Aide-mémoire du chapitre 26 617

Index 619

Introduction

Microsoft Visual C# est un langage puissant tout en restant simple, qui est destiné avant tout aux développeurs créant des applications avec le .NET Framework. Il hérite des meilleures fonctionnalités de C++ et de Microsoft Visual Basic, mais laisse de côté les incohérences et les anachronismes, ce qui en fait un langage plus clair et plus logique. La version 1.0 de C# est sortie au début de l'année 2001. L'arrivée de C# 2.0 avec Visual Studio 2005, a vu l'ajout de nouvelles fonctionnalités importantes au langage, notamment les génériques, les itérateurs, et les méthodes anonymes. C# 3.0, qui est sorti avec Microsoft Visual Studio 2008, a ajouté les méthodes d'extension, les expressions lambda, et, la fonctionnalité la plus connue de toutes, le langage de requête LINQ (*Language Integrated Query*). La dernière incarnation du langage, C# 4.0, fournit des fonctionnalités supplémentaires qui améliorent son interopérabilité avec d'autres langages et technologies. Parmi ces nouveautés, on peut citer la prise en charge des arguments nommés et optionnels, le type *dynamic* qui indique que le runtime du langage doit implémenter la liaison tardive d'un objet et la variance qui résout certains problèmes dans la manière dont les interfaces génériques sont définies. C# 4.0 tire parti de la dernière version du .NET Framework, qui est la version 4.0. Il y a de nombreux ajouts au .NET Framework dans cette version, mais les plus significatifs sont sans doute les classes et les types qui constituent la Bibliothèque parallèle de tâches. Avec cette bibliothèque, vous pouvez à présent créer des applications hautement évolutives qui peuvent tirer parti rapidement et facilement des processeurs multicœurs. La prise en charge des services Web et de Windows Communication Foundation (WCF) a aussi été étendue ; vous pouvez à présent créer des services qui suivent le modèle REST ainsi que le schéma plus traditionnel SOAP.

L'environnement de développement fourni par Microsoft Visual Studio 2010 facilite l'emploi de ces fonctionnalités puissantes et ses nombreux assistants améliorent votre productivité.

À qui s'adresse ce manuel

Cet ouvrage présuppose que vous êtes un développeur qui souhaite apprendre les principes de base de la programmation C# avec Visual Studio 2010 et la version 4.0 du .NET Framework. Dans cet ouvrage, vous allez apprendre les fonctionnalités du langage C#, puis les utiliser pour générer des applications qui s'exécutent sur le système d'exploitation Windows. Après avoir terminé la lecture de cet ouvrage, vous aurez une compréhension complète de C# et vous l'aurez utilisé pour générer des applications Windows Presentation Foundation, accéder à des bases de données Microsoft SQL Server, avec ADO.NET et LINQ, créer des applications réactives et évolutives avec la Bibliothèque parallèle de tâches, et créer des services Web REST et SOAP avec Windows Communication Foundation.

Recherche de votre meilleur point de départ dans ce manuel

Cet ouvrage est conçu pour vous aider à rassembler des compétences dans un certain nombre de domaines essentiels. Vous pouvez utiliser ce livre si vous débutez en programmation ou si vous utilisiez un autre langage de programmation comme C, C++, Java, ou Visual Basic. Utilisez le tableau suivant pour trouver votre meilleur point de départ.

Si vous	Accomplissez ces opérations
Êtes débutant en programmation orientée objet	**1.** Installez les fichiers d'exercices comme expliqué dans la prochaine section, « Installation et utilisation des fichiers d'exercices ». **2.** Étudiez les chapitres des parties I, II, et III dans l'ordre. **3.** Analysez les parties IV, V, et VI selon votre expérience et votre intérêt.
Êtes familier avec les langages de programmation procédurale comme C, mais débutant en C#	**1.** Installez les fichiers d'exercices comme expliqué dans la prochaine section, « Installation et utilisation des fichiers d'exercices ». Passez en revue les cinq premiers chapitres pour obtenir un aperçu de C# et de Visual Studio 2010, puis concentrez-vous sur les chapitres 6 à 21. **2.** Analysez les parties IV, V, et VI selon votre expérience et votre intérêt.
Utilisiez avant un langage orienté objet comme C++ ou Java	**1.** Installez les fichiers d'exercices comme expliqué dans la prochaine section, « Installation et utilisation des fichiers d'exercices ». **2.** Passez en revue les sept premiers chapitres pour obtenir un aperçu de C# et de Visual Studio 2010, puis concentrez-vous sur les chapitres 8 à 21. **3.** Pour obtenir plus d'informations sur la génération d'applications Windows et l'utilisation d'une base de données, lisez les parties IV et V. **4.** Pour obtenir plus d'informations sur la création d'applications évolutives et de services Web, lisez la partie VI.
Utilisiez avant Visual Basic 6	**1.** Installez les fichiers d'exercices comme expliqué dans la prochaine section, « Installation et utilisation des fichiers d'exercices ». **2.** Étudiez les chapitres des parties I, II, et III dans l'ordre. **3.** Pour obtenir plus d'informations sur la génération d'applications Windows, lisez la partie IV. **4.** Pour obtenir plus d'informations sur l'accès à une base de données, lisez la partie V. **5.** Pour obtenir plus d'informations sur la création d'applications évolutives et de services Web, lisez la partie VI. **6.** Étudiez les sections Aide-mémoire à la fin des chapitres pour trouver des informations sur les constructions spécifiques de C# et de Visual Studio 2010.

Si vous	Accomplissez ces opérations
Vous vous reportez au livre après avoir effectué les exercices	**1.** Servez-vous de l'index ou de la table des matières pour trouver des informations sur des sujets particuliers. **2.** Étudiez les sections « Aide-mémoire » à la fin des chapitres pour trouver un bref récapitulatif de la syntaxe et des techniques présentées dans le chapitre.

Conventions de ce manuel

Ce livre présente des informations en utilisant des conventions destinées à les rendre plus lisibles et plus faciles à comprendre. Avant de commencer l'étude de ce manuel, passez en revue la liste suivante, qui explique les conventions que vous rencontrerez tout au long du livre.

- Chaque exercice est une série de tâches. Chaque tâche se présente sous la forme d'une série d'étapes numérotées (1, 2, etc.). Une puce (•) indique qu'un exercice ne comporte qu'une seule étape.
- Les remarques intitulées « Astuce » fournissent des informations supplémentaires ou d'autres méthodes permettant d'accomplir correctement une étape.
- Les remarques intitulées « Important » vous informent que vous devez effectuer une vérification avant de poursuivre.
- Le texte que vous saisissez apparaît en gras.
- Un signe plus (+) entre deux noms signifie que vous devez appuyer sur ces touches en même temps. Par exemple, « Appuyez sur Alt+Tab » signifie que vous devez maintenir la touche Alt enfoncée pendant que vous pressez sur la touche Tab.
- Les encadrés tout au long du livre fournissent des informations approfondies à propos de l'exercice. Ils contiennent des informations de base, des astuces de conception, ou des fonctionnalités liées aux thèmes abordés.
- Chaque chapitre se termine par une section Aide-mémoire. Cette section comporte des éléments permettant de vous rappeler rapidement les opérations étudiées dans le chapitre.

Pré-requis du système

Vous devez posséder le matériel et les logiciels suivants pour effectuer les exercices de ce manuel :

- Une version de Windows 7 (Home, Premium, Entreprise, Ultimate, etc.). Ces exercices s'exécuteront aussi en utilisant Microsoft Windows Vista avec le Service Pack 2

- Microsoft Visual Studio 2010 Professional, Visual Studio 2010 Premium, ou Microsoft Visual C# 2010 Express et Microsoft Visual Web Developer 2010 Express
- Microsoft SQL Server 2008 Express (ce logiciel est fourni avec toutes les versions de Visual Studio, avec Visual C# 2010 Express, et Visual Web Developer 2010 Express)
- Un processeur 1.6-GHz, ou plus rapide (les chapitres 27 et 28 nécessitent un processeur dual core)
- 1 Go de RAM pour les processeurs 32 bits et 2 Go de RAM pour les processeurs 64 bits
- Un écran (1 024 × 768 ou résolution supérieure) avec au moins 256 couleurs
- Une souris

Vous devrez également avoir l'accès Administrateur à votre ordinateur pour configurer SQL Server 2008 Express.

Suppléments en ligne et fichiers d'exercices

Le site *www.dunod.com* vous permettra de télécharger les fichiers d'exercices que vous utiliserez au moment des exercices. Grâce à ces fichiers, vous ne perdrez pas de temps à créer des fichiers qui ne sont pas importants pour l'exercice. Les fichiers et les instructions étape par étape de ces leçons vous permettent d'apprendre par la pratique, ce qui constitue un moyen simple et efficace d'acquérir et d'assimiler de nouvelles connaissances.

Remarque La dernière partie de l'ouvrage, *Création de solutions professionnelles* (qui est composée des chapitres 27 à 29, Introduction à la Bibliothèque parallèle de tâches, Accès parallèle aux données, Création et utilisation d'un service Web) est téléchargeable sur le site des éditions Dunod sous la forme d'un fichier PDF.

Installation et utilisation des fichiers d'exercices

Suivez ces étapes pour installer les fichiers d'exercice sur le disque dur de votre ordinateur pour pouvoir les utiliser dans les exercices.

1. Téléchargez les fichiers d'exercice sur la page consacrée à cet ouvrage sur le site *www.dunod.com*. Une fois les fichiers téléchargés, décompactez l'archive dans votre dossier Documents.

   Les fichiers d'exercice seront ainsi installés dans le dossier suivant : *Documents\Visual C Sharp Etape par étape*

2. Chaque chapitre de ce livre explique quand et comment utiliser ces fichiers. Au moment où vous devez utiliser un fichier d'exercice, le manuel vous indique toutes les instructions nécessaires pour ouvrir ce fichier.

Important Les fichiers d'exercices ont été testés en utilisant un compte qui est un membre du groupe local Administrateurs. Il est recommandé d'effectuer les exercices en utilisant un compte possédant des privilèges d'administrateur.

Assistance technique

Tous les efforts ont été faits pour assurer l'exactitude du contenu de ce livre. Pour tous commentaires, questions ou suggestions concernant ce livre, contactez Microsoft Press par l'une des méthodes suivantes :

Courriel : mspinput@microsoft.com

Poste :
Microsoft Press
Attn : Microsoft Visual C# 2010 Step by Step Series Editor
One Microsoft Way
Redmond, WA 98052-6399

Pour toutes questions spécifiques concernant cet ouvrage, visitez le site d'assistance technique de Microsoft Press *www.microsoft.com/learning/support/books*. Vous pouvez aussi vous connecter directement à la base de connaissances Microsoft et saisir une question en allant sur *http://support.microsoft.com/search*. Pour une assistance technique sur les logiciels Microsoft, consultez le site *http://support.microsoft.com*.

Partie I

Introduction à Microsoft Visual C# et Microsoft Visual Studio 2010

Dans cette partie :

Chapitre 1 Bienvenue dans l'univers de C# 3
Chapitre 2 Variables, opérateurs et expressions. 29
Chapitre 3 Écrire des méthodes et définir leur portée 49
Chapitre 4 Commandes de prise de décision. 75
Chapitre 5 Assignation composée et instruction d'itération 93
Chapitre 6 Erreurs et exceptions. 111

Chapitre 1

Bienvenue dans l'univers de C#

Au terme de ce chapitre, vous saurez :

- Utiliser l'environnement de programmation Microsoft Visual Studio 2010.
- Créer une application console C#.
- Expliquer le but des espaces de noms.
- Créer une application graphique simple C#.

Microsoft Visual C# est un puissant langage orienté composant créé par Microsoft. C# joue un rôle essentiel dans l'architecture de Microsoft .NET Framework, et certaines personnes ont comparé son rôle à celui joué par C dans le développement d'UNIX. Si vous connaissez déjà un langage comme C, C++ ou Java, vous trouverez que la syntaxe de C# en est très proche. Si vous êtes habitué à programmer dans d'autres langages, vous devriez rapidement vous familiariser avec la syntaxe de C# et vous n'aurez qu'à apprendre à placer les accolades et les points-virgules aux bons endroits.

La Partie I présente les bases de C#. Vous découvrirez comment déclarer des variables et comment utiliser des opérateurs arithmétiques, comme le signe plus (+) ou le signe moins (–) pour manipuler des valeurs dans les variables. Vous étudierez également la manière d'écrire des méthodes et de transformer des arguments en méthodes. Vous apprendrez aussi à vous servir des instructions de prise de décision, telles que *if*, et des instructions d'itération, comme *while*. Enfin, vous analyserez la manière dont C# utilise des exceptions pour gérer avec souplesse les erreurs. Quand vous maîtriserez toutes ces bases de C#, vous pourrez évoluer vers des fonctions plus avancées qui sont abordées dans les Parties II à VI.

Débuter dans l'environnement Visual Studio 2010

Visual Studio 2010 est un environnement de programmation riche en outils comportant toutes les fonctionnalités nécessaires pour créer des projets C# de toute taille. Vous pouvez même créer des projets qui combinent harmonieusement des modules écrits dans différents langages de programmation, comme C++, Visual Basic ou F#. Dans le premier exercice, vous lancerez l'environnement de programmation Visual Studio 2010 et vous apprendrez à concevoir une application console.

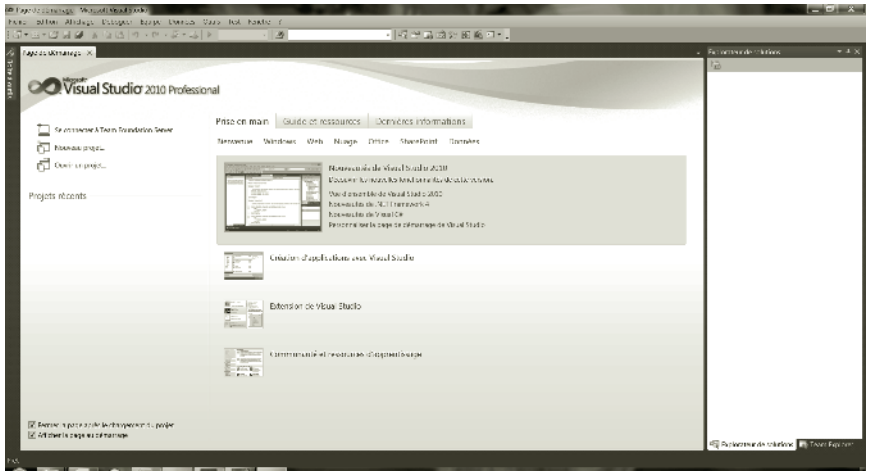
Remarque Une application console est une application qui s'exécute dans une fenêtre d'invite de commandes, et non pas dans une interface utilisateur graphique.

Création d'une application console dans Visual Studio 2010

- Si vous utilisez Visual Studio 2010 Professional ou Visual Studio 2010 Premium, effectuez les opérations suivantes pour démarrer Visual Studio 2010 :

  1. Dans la barre de tâches de Windows, cliquez sur le bouton *Démarrer*, pointez *Tous les programmes,* puis pointez le groupe de programmes *Microsoft Visual Studio 2010*.
  2. Dans le groupe de programmes Microsoft Visual Studio 2010, cliquez sur *Microsoft Visual Studio 2010*.

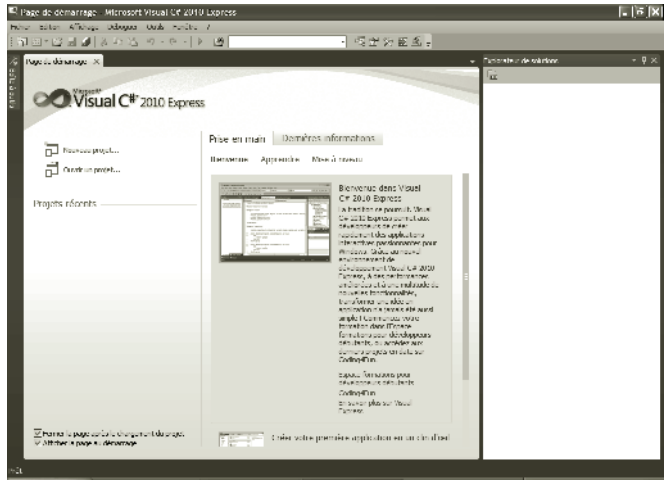
  Visual Studio 2010 affiche au démarrage la fenêtre suivante :



Remarque Si c'est la première fois que vous exécutez Visual Studio 2010, vous verrez certainement une boîte de dialogue vous demandant de choisir les paramètres par défaut de votre environnement de développement. Visual Studio 2010 s'adapte à votre langage de développement favori : ainsi, les diverses boîtes de dialogue et les différents outils de l'environnement de développement intégré (ou IDE pour *integrated development environment*) ont leurs paramètres par défaut définis en fonction du langage choisi. Sélectionnez *Paramètres de développement Visual C#* dans la liste, puis cliquez sur le bouton *Démarrer Visual Studio*. Après un bref instant, l'IDE de Visual Studio 2010 apparaît.

- Si vous utilisez Visual C# 2010 Express, cliquez sur le bouton *Démarrer* dans la barre de tâches Microsoft Windows, pointez *Tous les programmes*, et cliquez sur *Microsoft Visual Studio 2010 Express*, puis *Microsoft Visual C# 2010 Express*.

  Visual C# 2010 Express affiche au démarrage la fenêtre suivante :



Remarque Visual Studio 2010 s'adapte à votre langage de développement favori : ainsi, les diverses boîtes de dialogue et les différents outils de l'environnement de développement intégré (ou IDE pour *integrated development environment*) ont leurs paramètres par défaut définis en fonction du langage choisi. Sélectionnez *Paramètres de développement Visual C#* dans la liste, puis cliquez sur le bouton *Démarrer Visual Studio*. Après un bref instant, l'IDE de Visual Studio 2010 apparaît.

Remarque Afin d'éviter la répétition, tout au long du livre, j'indiquerai simplement, « Démarrer Visual Studio » lorsque vous aurez besoin d'ouvrir Visual Studio 2010 Professional, Visual Studio 2010 Premium, ou Visual C# 2010 Express. En outre, à moins d'être clairement spécifié, toutes les références à Visual Studio 2010 s'appliquent indifféremment à Visual Studio 2010 Professional, Visual Studio 2010 Premium, ou Visual C# 2010 Express .

- Si vous utilisez Visual Studio 2010 Professional ou Visual Studio 2010 Premium, effectuez les tâches suivantes pour créer une nouvelle application console.

  1. Dans le menu *Fichier*, pointez *Nouveau*, puis cliquez sur *Projet*.

     La boîte de dialogue *Nouveau projet* s'ouvre. Cette boîte de dialogue liste les modèles que vous pouvez utiliser comme point de départ pour construire une application. Les modèles sont présentés en fonction du langage de programmation et du type d'application que vous utilisez.

  2. Dans le volet de gauche, sous *Modèles installés*, cliquez sur *Visual C#*. Dans le volet du milieu, vérifiez que la liste au sommet du volet indique bien *.NET Framework 4.0*, puis cliquez sur l'icône *Application console*.

3. Dans le champ *Emplacement*, si vous utilisez le système d'exploitation Windows Vista, saisissez **C:\Utilisateurs*VotreNom*\Documents\Visual C Sharp Etape par étape\Chapitre 1**. Si vous utilisez Windows 7, saisissez **C:\Utilisateurs*VotreNom*\Mes Documents\Visual C Sharp Etape par étape\Chapitre 1**.

   Remplacez le texte *VotreNom* dans ces chemins par votre nom d'utilisateur Windows.

Remarque Par souci de concision, nous désignerons désormais le chemin "C:\Utilisateurs*VotreNom*\Documents" ou "C:\Utilisateurs*VotreNom*\Mes Documents" sous la dénomination votre dossier Documents.

Astuce Si le dossier que vous avez spécifié n'existe pas, Visual Studio 2010 le crée pour vous.

4. Dans le champ *Nom*, saisissez **Bonjour**.
5. Assurez-vous que la case *Créer le répertoire pour la solution* est cochée, puis cliquez sur OK.

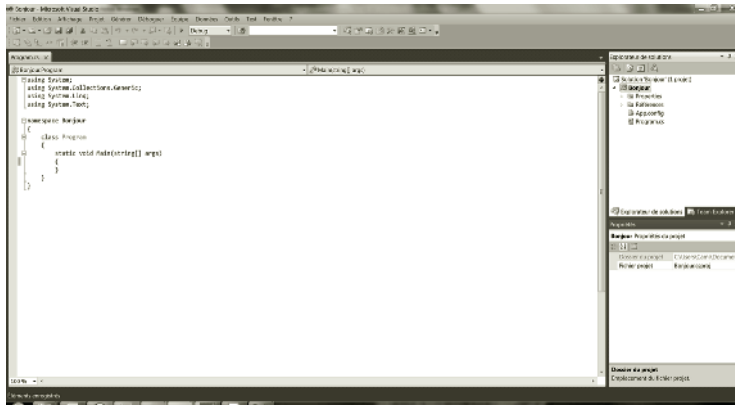
- Si vous utilisez Visual C# 2010 Express, accomplissez les tâches suivantes pour créer une nouvelle application console :

1. Dans le menu *Fichier*, cliquez sur *Nouveau projet*.
2. Dans la boîte de dialogue *Nouveau projet*, dans le volet du milieu, cliquez sur l'icône *Application console*.
3. Dans le champ *Nom*, saisissez **Bonjour**.
4. Cliquez sur *OK*.

Visual C# 2010 Express enregistre par défaut la solution dans le dossier C:\Utilisateurs\Votre nom\AppData\Local\Temporary Projects. Vous pouvez spécifier un autre emplacement au moment de l'enregistrement de la solution.

5. Dans le menu *Fichier*, cliquez sur *Enregistrer Bonjour sous*.
6. Dans le champ Emplacement de la boîte de dialogue *Enregistrer un projet*, spécifiez le dossier **Visual C Sharp Etape par étape\Chapitre 1** à partir de votre dossier Documents.
7. Cliquez sur *Enregistrer*.

Visual Studio crée le projet en utilisant le modèle Application console et affiche le code de démarrage du projet :



La *barre de menus* en haut de l'écran vous permet d'accéder aux fonctions que vous utiliserez dans l'IDE. Vous pouvez utiliser le clavier ou la souris pour accéder aux menus et aux commandes. La *barre d'outils* se situe sous la barre de menus et propose des raccourcis pour exécuter les commandes les plus utilisées.

La fenêtre *Code*, qui occupe la majeure partie de l'IDE, affiche le contenu des fichiers source. Dans un projet multifichier, si vous modifiez plusieurs fichiers, chaque fichier source possède son propre *onglet* avec son nom. Vous pouvez cliquer sur l'onglet pour faire apparaître au premier plan, dans la fenêtre *Code*, le fichier source. Le volet *Explorateur de solutions* (dans la partie droite de la boîte de dialogue) affiche, entre autres, le nom des fichiers associés au projet. Vous pouvez faire un double-clic sur le nom d'un fichier dans le volet *Explorateur de solutions* pour faire apparaître ce fichier source au premier plan dans la fenêtre *Code*.

Avant d'écrire le code, examinez les fichiers listés dans l'*Explorateur de solutions*, que Visual Studio 2010 a créés comme éléments de votre projet :

- **Solution 'Bonjour' :** c'est le fichier solution qui est au sommet de la liste ; il en existe un par application. Si vous vous servez de l'Explorateur Windows pour rechercher votre dossier Documents\Visual C Sharp Etape par

étape\Chapitre 1\Bonjour, vous verrez que le nom de ce fichier est en fait Bonjour.sln. Chaque fichier solution contient des références à un ou plusieurs fichiers projet.

- **Bonjour :** c'est le fichier projet C#. Chaque fichier projet répertorie un ou plusieurs fichiers contenant le code source et d'autres éléments relatifs à ce projet. L'ensemble du code source d'un projet doit être écrit dans le même langage de programmation. Dans l'Explorateur Windows, ce fichier qui s'appelle Bonjour.csproj est stocké dans le dossier \Visual C Sharp Etape par étape\Chapitre1\Bonjour\Bonjour sous le dossier de vos Documents.
- **Properties :** c'est un dossier du projet Bonjour. Si vous le déployez, vous verrez qu'il contient un fichier nommé AssemblyInfo.cs. Ce dernier est un fichier spécial que vous pouvez utiliser pour ajouter des *attributs* à un programme, comme le nom du créateur, la date d'écriture du programme, etc. Vous pouvez spécifier d'autres attributs pour modifier la manière dont s'exécute le programme. Apprendre à utiliser ces attributs dépasse le cadre de ce livre.
- **References :** c'est un dossier qui comporte des références au code compilé que votre application peut utiliser. Quand un code est compilé, il est converti en un *assembly* et se voit attribuer un nom unique. Les développeurs se servent d'assemblys pour regrouper des routines de code utiles qu'ils ont écrites afin de les distribuer à d'autres développeurs pour qu'ils puissent les utiliser dans leurs applications. De nombreuses fonctions, que vous utiliserez quand vous écrirez des applications avec ce manuel, se serviront d'assemblys fournis par Microsoft avec Visual Studio 2010.
- **App.config :** c'est le fichier de configuration de l'application. Vous pouvez spécifier les paramètres que votre application peut utiliser au moment de l'exécution pour modifier son comportement, par exemple la version du .NET Framework à employer pour exécuter l'application. Vous en apprendrez plus sur ce fichier dans les prochains chapitres de cet ouvrage.
- **Program.cs :** c'est un fichier source C#. C'est celui qui s'affiche dans la fenêtre *Code* quand le projet est créé. Vous écrirez le code de l'application console dans ce fichier qui contient du code que Visual Studio 2010 génère automatiquement.

Écrire votre premier programme

Le fichier Program.cs définit une classe intitulée Program qui comporte une méthode appelée *Main*. Toutes les méthodes doivent être définies dans une classe. Vous en apprendrez plus sur les classes dans le chapitre 7, « Classes et objets ». La méthode *Main* est spéciale car elle correspond au point de lancement du programme. Ce doit être une méthode statique (les méthodes sont abordées en détail dans le chapitre 3, « Écrire des méthodes et définir leur portée », et les méthodes statiques sont traitées dans le chapitre 7).

Important C# est un langage sensible à la casse. Vous devez écrire *Main* avec un *M* majuscule.

Dans les exercices suivants, vous allez écrire du code pour afficher le message *Bonjour* sur la console, puis vous générerez et exécuterez l'application console Bonjour ; pour finir, vous étudierez la manière dont les espaces de noms sont utilisés pour séparer les éléments du code.

Écriture du code à l'aide de la technologie Microsoft IntelliSense

1. Dans la fenêtre *Code* qui affiche le fichier Program.cs, placez le curseur dans la méthode *Main* directement après l'accolade ouvrante, {, puis pressez Entrée pour créer une nouvelle ligne. Dans la nouvelle ligne, saisissez le mot **Console**, qui est le nom d'une classe intégrée. Quand vous saisissez la lettre C au début du mot *Console*, une liste IntelliSense apparaît. Cette liste comporte tous les mots-clés C# et les types de données valides dans ce contexte. Vous pouvez continuer votre saisie ou faire défiler la liste et effectuer un double-clic sur l'élément Console. Sinon, après avoir écrit *Con*, la liste Intellisense se positionnera automatiquement sur l'élément *Console* et vous n'aurez qu'à appuyer sur la touche de tabulation ou la touche Entrée pour le sélectionner.

   Main devrait alors ressembler à ceci :

```
static void Main(string[] args)
{
    Console
}
```

Remarque *Console* est une classe intégrée qui contient les méthodes permettant d'afficher des messages à l'écran et de récupérer les entrées depuis le clavier.

2. Mettez un point directement après *Console*. Une autre liste Intellisense apparaît, présentant les méthodes, les propriétés et les champs de la classe *Console*.
3. Faites dérouler la liste, sélectionnez *WriteLine,* puis appuyez sur Entrée. Vous pouvez également continuer à saisir les caractères *W*, *r*, *i*, *t*, *e*, *L* jusqu'à ce que *WriteLine* soit sélectionné*,* puis appuyer sur Entrée.

   La liste IntelliSense se ferme et le mot *WriteLine* est ajouté au fichier source. *Main* devrait maintenant ressembler à ceci :

```
static void Main(string[] args)
{
    Console.WriteLine
}
```

4. Saisissez une parenthèse ouvrante, (. Une autre astuce IntelliSense s'affiche.

Celle-ci présente les paramètres de la méthode *WriteLine*. En fait, *WriteLine* est une *méthode surchargée*, ce qui signifie que la classe *Console* contient plus d'une méthode appelée *WriteLine* (elle fournit en fait 19 versions différentes de cette méthode). Chaque version de la méthode *WriteLine* peut être utilisée pour transmettre en sortie différents types de données (les méthodes surchargées sont abordées au chapitre 3). *Main* devrait maintenant ressembler à ceci :

```
static void Main(string[] args)
{
    Console.WriteLine(
}
```

Astuce Vous pouvez cliquer sur les flèches de l'astuce pour faire défiler les différentes versions de *WriteLine*.

5. Saisissez une parenthèse fermante,), suivie d'un point-virgule, ;.

   Main devrait maintenant ressembler à ceci :

```
static void Main(string[] args)
{
    Console.WriteLine();
}
```

6. Déplacez le curseur, et saisissez la chaîne **"Bonjour"**, sans oublier les guillemets, entre les parenthèses suivant la méthode *WriteLine*.

   Main devrait maintenant ressembler à ceci :

```
static void Main(string[] args)
{
    Console.WriteLine("Bonjour");
}
```

Astuce Habituez-vous à saisir des paires de caractères correspondantes, comme (et) ou { et }, *avant* d'insérer leur contenu. Il est fréquent d'oublier le caractère fermant si vous saisissez d'abord le contenu.

Icônes IntelliSense

Lorsque vous saisissez un point après le nom d'une classe, IntelliSense affiche le nom de chaque membre de cette classe. À gauche de chaque nom de membre se trouve une icône qui indique le type de celui-ci. Voici les principales icônes et leurs types :

Icône	Signification
	méthode (abordée dans le chapitre 3)

Icône	Signification
	propriété (abordée dans le chapitre 15)
	classe (abordée dans le chapitre 7)
	structure (abordée dans le chapitre 9)
	énumération (abordée dans le chapitre 9)
	interface (abordée dans le chapitre 13)
	délégué (abordé au chapitre 17)
	méthode d'extension (abordée dans le chapitre 12)

Vous verrez également d'autres icônes IntelliSense apparaître lorsque vous saisirez du code dans différents contextes

.

Remarque Vous verrez fréquemment des lignes de code contenant deux caractères slash consécutifs suivis de texte ordinaire. Ce sont des commentaires. Ils sont ignorés par le compilateur, mais sont très utiles pour les développeurs, puisqu'ils permettent de documenter les actions d'un programme. Par exemple :

```
Console.ReadLine(); // Attendre que l'utilisateur appuie sur la touche Entrée
```

Tout le texte à partir des caractères slash jusqu'à la fin de la ligne sera ignoré par le compilateur. Vous pouvez ajouter des commentaires sur plusieurs lignes en commençant par un slash suivi d'un astérisque (/*). Le compilateur ignorera tout ce qui se trouve jusqu'à la prochaine séquence (/*), qui peut se trouver plusieurs lignes plus bas. Vous êtes vivement encouragé à documenter votre code en y insérant autant de commentaires explicatifs que nécessaire.

Génération et exécution de l'application console

1. Dans le menu *Générer*, cliquez sur *Générer la solution*.

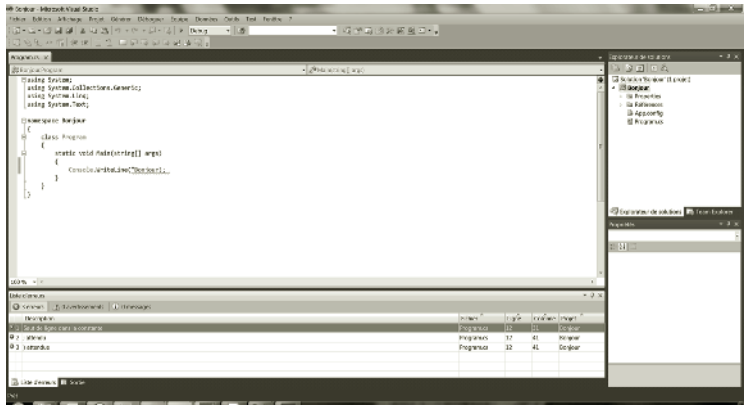
   Cette action compile le code C#, créant un programme que vous pouvez exécuter. La fenêtre *Sortie* apparaît sous la fenêtre *Code*.

Astuce Si la fenêtre *Sortie* n'apparaît pas, dans le menu *Affichage*, cliquez sur *Sortie* pour la faire apparaître.

Dans la fenêtre *Sortie*, vous devez voir des messages similaires à celui reproduit ci-dessous qui indique comment le programme a été compilé :

```
------ Début de la génération : Projet : Bonjour, Configuration : Debug x86 ------
  Bonjour -> C:\Users\Domi\Mes Documents\Visual C Sharp Etape par étape \
Chapitre 1\Bonjour\Bonjour\bin\Debug\Bonjour.exe
========== Génération : 1 a réussi ou est à jour, 0 a échoué, 0 a été ignoré ==========
```

Si vous avez fait des erreurs, elles apparaîtront dans la fenêtre *Liste d'erreurs*. La figure suivante montre ce qui se passe si vous oubliez de saisir le guillemet fermant après le texte Bonjour dans la commande *WriteLine*. Notez bien qu'une seule erreur peut parfois provoquer plusieurs erreurs de compilation.



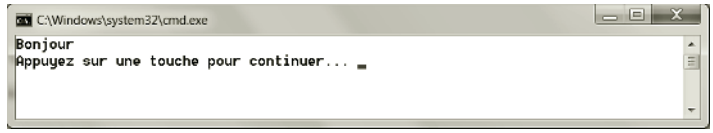
Astuce Vous pouvez faire un double-clic sur un élément dans la fenêtre *Liste d'erreurs*, et le curseur sera alors placé sur la ligne qui provoque l'erreur. Vous remarquerez aussi que, lors de la saisie, Visual Studio affiche un trait ondulé rouge sous les lignes de code qu'il ne compile pas.

Si vous avez soigneusement suivi les instructions précédentes, il ne devrait pas y avoir d'erreurs ou d'alertes, et le programme devrait se générer avec succès.

Astuce Vous n'avez pas besoin d'enregistrer le fichier de façon explicite avant la génération car la commande *Générer la solution* enregistre automatiquement le fichier. Un astérisque après le nom du fichier dans l'onglet au-dessus de la fenêtre *Code* indique que le fichier a été modifié depuis sa dernière sauvegarde.

2. Dans le menu *Déboguer*, cliquez sur Exécuter sans débogage.

   Une fenêtre de Commande s'ouvre et le programme s'exécute. Le message Bonjour apparaît, puis le programme attend que l'utilisateur appuie sur n'importe quelle touche, comme l'illustre la figure suivante :

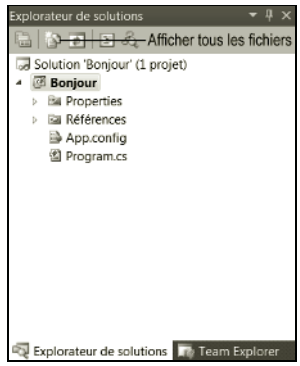


Remarque L'invite "Appuyez sur une touche pour continuer. . ." est générée par Visual Studio ; vous n'avez pas écrit de code pour faire cela. Si vous exécutez le programme au moyen de la commande *Démarrer le débogage* dans le menu *Déboguer*, l'application s'exécute, mais la fenêtre de commande se ferme immédiatement sans attendre que vous appuyiez sur une touche.

3. Assurez-vous que la fenêtre de commande affichant la sortie du programme est active, puis appuyez sur Entrée.

   La fenêtre de commande se ferme et vous êtes à nouveau dans l'environnement de programmation de Visual Studio 2010.

4. Dans l'*Explorateur de solutions*, cliquez sur le projet Bonjour (pas la solution), puis cliquez sur le bouton *Afficher tous les fichiers* dans la barre d'outils *Explorateur de solutions* (il s'agit du deuxième bouton en partant de la gauche dans la barre d'outils de la fenêtre Explorateur de solutions).



   Les entrées nommées *bin* et *obj* apparaissent au-dessus du fichier Program.cs. Ces entrées correspondent directement aux dossiers intitulés *bin* et *obj* dans le dossier du projet (Visual C Sharp Etape par étape\Chapitre 1\Bonjour\Bonjour). Visual Studio crée ces dossiers lorsque vous générez votre application et ils contiennent la version exécutable du programme et d'autres fichiers utilisés pour générer et déboguer l'application.

5. Dans l'*Explorateur de solutions*, développez l'entrée *bin*.

   Un autre dossier nommé *Debug* apparaît.

Remarque Il est possible que vous voyiez aussi un dossier nommé *Release*.

6. Dans l'*Explorateur de solutions*, développez le dossier *Debug*.

   Quatre autres éléments nommés Bonjour.exe, Bonjour.pdb, Bonjour.vshost.exe, et Bonjour.vshost.exe.manifest apparaissent.

   Le fichier Bonjour.exe correspond au programme compilé, et il s'agit du fichier qui s'exécute lorsque vous cliquez sur *Exécuter sans débogage* dans le menu *Déboguer*. Les autres fichiers contiennent des informations utilisées par Visual Studio 2010 si vous exécutez votre programme en mode *Débogage* (lorsque vous cliquez sur *Démarrer le débogage* dans le menu *Déboguer*).

Utiliser des espaces de noms

L'exemple que vous venez d'étudier correspond à un tout petit programme. Toutefois, les petits programmes peuvent rapidement devenir plus importants. Au fur et à mesure de leur développement, deux problèmes surgissent. Tout d'abord, il est bien plus difficile de comprendre et de maintenir des programmes importants que des programmes plus petits. Deuxièmement, l'augmentation de la taille du code implique en général plus de noms, plus de méthodes, et plus de classes. Plus le nombre de noms augmente, plus il y a de risques d'échec lors de la génération du projet, puisque plusieurs noms peuvent entrer en conflit (notamment quand le programme utilise des bibliothèques tierces écrites par des développeurs qui ont également utilisé une grande variété de noms).

Dans le passé, les programmeurs ont tenté de résoudre ce problème de conflit de noms en préfixant les noms avec une sorte de qualificateur (ou des ensembles de qualificateurs). Cette solution n'est pas appropriée puisqu'elle n'est pas évolutive ; les noms s'allongent et vous passez moins de temps à écrire du code et plus de temps à saisir (ce n'est pas la même chose), à lire et à relire des noms incompréhensiblement longs.

Les espaces de noms vous aident à résoudre ce problème en créant un conteneur nommé pour les autres identificateurs comme les classes. Deux classes portant le même nom ne pourront plus être confondues si elles se trouvent dans des espaces de noms différents. Vous pouvez créer une classe appelée *Salut* dans l'espace de noms intitulé *Bonjour*, comme ceci :

```
namespace Bonjour
{
    class Salut
    {
        ...
    }
}
```

Vous pouvez ensuite faire référence à la classe *Salut* sous la forme *Bonjour.Salut* dans vos programmes. Si un autre développeur crée également une classe *Salut* dans un autre espace de noms, comme *NouvelEspaceDeNoms* et l'installe sur votre ordinateur, vos programmes fonctionneront quand même normalement,