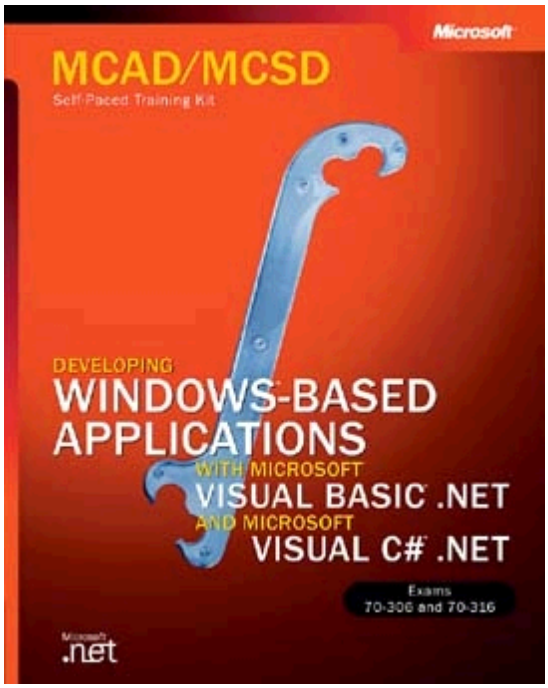
This document is created with the unregistered version of CHM2PDF Pilot



Copyright 2002 by Microsoft Corporation

This document is created with the unregistered version of CHM2PDF Pilot

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Copyright

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

Library of Congress Cataloging-in-Publication Data

This document is created with the unregistered version of CHM2PDF Pilot

About This Book

Welcome to *MCAD/MCSD Training Kit Developing Windows-Based Applications with Microsoft Visual Basic .NET and Microsoft Visual C# .NET*. By completing the lessons and associated exercises in this course, you will acquire the knowledge and skills necessary to develop Windows-based solutions using Visual Basic .NET or Visual C# .NET.

This book also addresses the objectives of the Microsoft Certified Professional Exam 70-306 and Exam 70-316. This self-paced course provides content that supports the skills measured by these exams.
For more information about becoming a Microsoft Certified Application Developer (MCAD) or a Microsoft Certified Solution Developer (MCSD), see the section titled "The Microsoft Certified Professional Program" later in this introduction.

The "Getting Started" section of this introduction provides important setup instructions that describe the hardware and software requirements to complete the procedures in this course. It also provides information about the networking configuration necessary to complete some of the hands-on procedures. Read through this section thoroughly before you start the lessons.

This document is created with the unregistered version of CHM2PDF Pilot

Intended Audience

This book was created for software developers who need to design, plan, implement, and support Windows-based applications or who plan to take the related Microsoft Certified Professional exams:

- *Developing and Implementing Windows-Based Applications with Microsoft Visual Basic .NET and Microsoft Visual Studio .NET (Exam 70-306)*
- *Developing and Implementing Windows-Based Applications with Microsoft Visual C# .NET and Microsoft Visual Studio .NET (Exam 70-316)*

This document is created with the unregistered version of CHM2PDF Pilot

Prerequisites

This course requires that students meet the following prerequisites:

- Be able to create a simple application using Visual Basic .NET or Visual C# .NET.
- Be able to describe the purpose and use of basic controls and menus in a Visual Basic .NET or Visual C# .NET application.
- Be able to describe the relationship between controls and events.
- Have a moderate understanding of basic Structured Query Language (SQL) syntax.

This document is created with the unregistered version of CHM2PDF Pilot

About the CD-ROM

The Supplemental Course Materials CD-ROM contains a variety of informational aids that can be used throughout this book. This includes

- **eBook.** A complete electronic version (eBook) of this training kit.
- **Completed labs.** Each chapter in this training kit concludes with a lab containing a series of exercises that reinforce the skills you learned. Completed versions of these applications are included so that you can compare your results. You can also use these completed applications as a reference if you get stuck while completing an exercise.
- **Required files.** Practice files that are required to perform the hands-on procedures. You should use these files when indicated in the exercises.
- **Sample exam questions.** To practice taking a certification exam, you can use the sample exam questions provided on the CD-ROM. The sample questions help you assess your understanding of the materials presented in this book.

This document is created with the unregistered version of CHM2PDF Pilot

About the DVD

The DVD contains a 60-day evaluation edition of Microsoft Visual Studio .NET Professional.
The 60-day Evaluation Edition provided with this training is not the full retail product and is provided only for the purposes of training and evaluation. Microsoft Technical Support does not support this evaluation edition.

For additional support information regarding this book and the CD-ROM and DVD (including answers to commonly asked questions about installation and use), visit the Microsoft Press Technical Support Web site at http://www.microsoft.com/mspress/support/. You can also email TKINPUT@MICROSOFT.COM, or send a letter to Microsoft Press, Attn: Microsoft Press Technical Support, One Microsoft Way, Redmond, WA 98502-6399.

This document is created with the unregistered version of CHM2PDF Pilot

Features of This Book

Each chapter contains sections that are designed to help you get the most educational value from the chapter:

- Each chapter opens with a "Before You Begin" section, which prepares you for completing the chapter.
- The chapters are then divided into lessons. Each lesson contains the reference and procedural information used for a specific skill.
- At the end of each lesson is the "Lesson Summary" section, which identifies the key concepts from the lesson.
- The "Lab" section provides hands-on exercises that reinforce each of the skills taught in each of the chapter lessons. The exercises give you an opportunity to use the skills being presented or explore the part of the application being described. Wherever possible, the exercises in a lab build on each other to create a complete application by the end of that lab.
- At the end of each chapter is the "Review" section that you can use to test what you have learned.

Appendix A, "Questions and Answers," contains all of the questions asked in each chapter with their corresponding answers.

Notes

Two types of Notes appear throughout the lessons.

- Notes marked **Note** contain supplemental information.
- Notes marked **Caution** contain warnings about the possible loss of data.

Notational Conventions

The following notational conventions are used throughout this book.

- Characters or commands that you type appear in **bold** type.
- *Italic* in syntax statements indicates placeholders for variable information. *Italic* is also used for book titles and program elements in text.
- Names of files and folders appear in initial capital letters except when you are to type them directly. Unless otherwise indicated, you can use lowercase letters when you type a filename in a dialog box or at a command prompt.
- Filename extensions, when they appear without a filename, are in lowercase letters.
- Acronyms appear in all uppercase letters.
- Monospace type represents code samples, examples of screen text, or entries that you might type at a command prompt or in initialization files.
- Angle brackets < > are used in syntax statements to enclose optional items. For example, <*filename*> in command syntax indicates that you can choose to type a filename with the command. Type only the information within the brackets, not the brackets themselves.
- When Visual Basic and Visual C# terms are mentioned together in text, the Visual Basic term appears first, followed by the C# term in parentheses.
- Icons represent specific sections in the book as follows:

Icon	Represents

This document is created with the unregistered version of CHM2PDF Pilot

Supplemental course materials. You will find these materials on the Supplemental Course Materials CD-ROM.

A hands-on exercise. You should perform the exercise to give yourself an opportunity to use the skills being presented in the lesson.

Chapter review questions. These questions at the end of each chapter allow you to test what you have learned in the chapter. You will find the answers to the review questions in Appendix A, "Questions and Answers" at the end of the book.

Keyboard Conventions

- A plus sign (+) between two key names means that you must press those keys at the same time. For example, Press Alt+Tab means that you hold down Alt while you press Tab.
- A comma (,) between two or more key names means that you must press each of the keys consecutively, not together. For example, Press Alt, F, X means that you press and release each key in sequence. Press Alt+W, L means that you first press Alt and W at the same time, and then release them and press L.
- You can choose menu commands by using the keyboard. Press the Alt key to activate the menu bar, and then sequentially press the keys that correspond to the highlighted or underlined letter of the menu name and the command name. For some commands, you can also press a key combination listed on the menu.
- You can select or clear check boxes or option buttons in dialog boxes with the keyboard. Press the Alt key, and then press the key that corresponds to the underlined letter of the option name. Or you can press Tab until the option is highlighted, and then press the Spacebar to select or clear the check box or option button.
- You can cancel the display of a dialog box by pressing the Esc key.

Chapter and Appendix Overview

This self-paced training kit combines notes, hands-on procedures, and review questions to teach you how to create Windows-based applications with Visual Basic .NET and Visual C# .NET. It is designed to be completed from beginning to end, but you can choose a customized track and only complete the sections that interest you. (See the next section, "Finding the Best Starting Point for You," for more information.) If you choose the customized track option, see the "Before You Begin" section in each chapter. Any hands-on procedures that require preliminary work from preceding chapters refer to the appropriate chapters.

The book is divided into the following sections and chapters:

- The "About This Book" section contains a self-paced training overview and introduces the components of this training kit. Read this section thoroughly to get the greatest educational value from this self-paced training and to plan which lessons you will complete.
- Chapter 1, "Introduction to the .NET Framework," introduces the .NET Framework, the common language runtime, and the .NET base class library. It describes how memory is managed using garbage collection. It explains how to create a class and a structure, and how to implement and scope methods.
- Chapter 2, "Creating the User Interface," describes the elements of the user interface and the steps you take in creating it. It explains controls, menus, using form and control events, and validating user input.
- Chapter 3, "Types and Members," explains in detail how to implement and use custom types. It explains how to create arrays and collections, and how to implement properties and events.

This document is created with the unregistered version of CHM2PDF Pilot

- Chapter 4, "Object-Oriented Programming and Polymorphism," introduces the concepts of object-oriented programming. It describes how to create overloaded members and how to implement polymorphism through inheritance and interface implementation.
- Chapter 5, "Testing and Debugging Your Application," describes how to use the tools provided by Visual Studio .NET to debug your application. It explains how to use the *Trace* and *Debug* classes to get real-time feedback from your application, and how to throw and handle exceptions. This chapter also describes how to develop a unit test plan for your application.
- Chapter 6, "Data Access Using ADO.NET," explains in detail how to use Microsoft ADO.NET to access data from a variety of sources. Connected and disconnected data access is explained, and binding data to the user interface is examined as well. This chapter also provides a sampling of XML data topics.
- Chapter 7, "Creating Controls Using the .NET Framework," describes how to use the .NET Framework to create user controls, custom controls, and inherited controls. It describes how to render visual elements with GDI+ technology, and how to accomplish some common tasks with controls.
- Chapter 8, "Advanced .NET Framework Topics," describes several advanced development topics. It explains how to create localizable applications, and how to implement printing in your application. It describes how to access COM components, the Windows API, and Web Services. It also explains how to implement help and set accessibility properties for your application.
- Chapter 9, "Assemblies, Configuration, and Security," explains how to create assemblies and use resources. It describes how to retrieve values from the configuration file and use them in your application. This chapter also explains how to implement role-based and code access security in your application.
- Chapter 10, "Deploying Your Application," explains how to create and configure a setup project for your application and how to plan a deployment scheme for your program.
- Appendix A, "Questions and Answers," lists all of the review questions from the book, showing the page number for each question and the suggested answer.
- The Glossary provides definitions for many of the terms and concepts presented in this training kit.

Finding the Best Starting Point for You

Because this book is self-paced, you can skip some lessons and revisit them later. Use the following table to find the best starting point for you:

If you	Follow this learning path
Are preparing to take the Microsoft Certified Professional Exam 70-306 or Exam 70-316	Read the "Getting Started" section. Then work through the remaining chapters in any order.
Want to review information about specific topics from the exam	Use the "Where to Find Specific Skills in This Book" section that follows this table.

Where to Find Specific Skills in This Book

The following table provides a list of the skills measured on certification Exam 70-306, *Developing and Implementing Windows-Based Applications with Microsoft Visual Basic .NET and Microsoft Visual Studio .NET*, and Exam 70-316, *Developing and Implementing Windows-Based Applications with Microsoft Visual C# .NET and Microsoft Visual Studio .NET*. The table provides the skill and where in this book you will find the lesson relating to that skill.

Exam skills are subject to change without prior notice and at the sole discretion of Microsoft.

Table A.1 *Creating User Services*

Skill Being Measured	Location in Book

This document is created with the unregistered version of CHM2PDF Pilot

Create a Windows Form by using the Windows Form Designer	Chapter 2, Lesson 2
Add controls to a Windows Form	Chapter 2, Lesson 3
Implement navigation for the user interface	Chapter 2, Lesson 2
Validate user input	Chapter 2, Lesson 5
Implement error handling in the UI	Chapter 5, Lesson 4
Implement online user assistance	Chapter 8, Lesson 4
Display data from a data source	Chapter 6, Lesson 4
Incorporate existing code into a Microsoft Windows-based application	Chapter 8, Lesson 2
Instantiate and invoke Web Services and components	Chapter 8, Lesson 2
Implement globalization	Chapter 8, Lesson 5
Create, implement, and handle events	Chapter 3, Lesson 4
Implement print capability	Chapter 8, Lesson 1
Implement accessibility features	Chapter 8, Lesson 3

Table A.2 *Creating and Managing Components and .NET Assemblies*

Skill Being Measured	**Location in Book**
Create and modify a .NET assembly	Chapter 9, Lesson 1
Create a Windows control.	Chapter 7, Lesson 2

Table A.3 *Consuming and Manipulating Data*

Skill Being Measured	**Location in Book**
Access and manipulate data from a Microsoft SQL Server database by creating and using ad hoc queries and stored procedures	Chapter 6, Lessons 1, 2, 3
Access and manipulate data from a data store	Chapter 6, Lessons 1, 2, 3
Handle data errors	Chapter 6, Lesson 3

Table A.4 *Testing and Debugging*

Skill Being Measured	**Location in Book**
Create a unit test plan	Chapter 5, Lesson 3

This document is created with the unregistered version of CHM2PDF Pilot

Implement tracing	Chapter 5, Lesson 2
Debug, rework, and resolve defects in code	Chapter 5, Lesson 1

Table A.5 *Deploying a Windows-Based Application*

Skill Being Measured	**Location in Book**
Plan the deployment of a Windows-based application	Chapter 10, Lesson 1
Create a setup program that installs an application and allows for the application to be uninstalled	Chapter 10, Lesson 1
Deploy a Windows-based application	Chapter 10, Lesson 1
Add assemblies to the Global Assembly Cache	Chapter 9, Lesson 1; Chapter 10, Lesson 2
Verify security policies for a deployed application	Chapter 10, Lesson 2

Table A.6 *Maintaining and Supporting a Windows-Based Application*

Skill Being Measured	**Location in Book**
Optimize the performance of a Windows-based application	Chapter 5, Lesson 2; Chapter 9, Lesson 2
Diagnose and resolve errors and issues	Chapter 5, Lesson 1; Chapter 9, Lesson 2

Table A.7 *Configuring and Securing a Windows-Based Application*

Skill Being Measured	**Location in Book**
Configure a Windows-based application	Chapter 9, Lesson 2
Configure security for a Windows-based application	Chapter 9, Lesson 3
Configure authorization	Chapter 9, Lesson 3

This document is created with the unregistered version of CHM2PDF Pilot

Getting Started

This self-paced training kit contains hands-on procedures to help you learn about developing Windows-based applications with Visual Basic .NET and Visual C# .NET.

Hardware Requirements

Each computer must have the following minimum configuration. All hardware should be on the Microsoft Windows 2000 Hardware Compatibility List.

- Pentium II class processor, 450 megahertz (MHz).
- 160 MB physical memory, 256 MB recommended.
- CD-ROM or DVD drive, 12x or faster recommended.
  A DVD drive is required to install the Visual Studio .NET Professional Evaluation Edition software.
- 3.5 gigabytes (GB) on installation drive, which includes 500 megabytes (MB) on system drive.
- Super VGA (800 x 600) or higher-resolution monitor with 256 colors.
- Microsoft Mouse or compatible pointing device.

Software Requirements

The following software is required to complete the procedures in this course.

- Microsoft Windows 2000 or Microsoft Windows XP
- Microsoft Visual Studio .NET, Professional Edition, Enterprise Developer Edition, or Enterprise Architect Edition
- Microsoft Access 2000 with the Jet 4.0 data engine

Setup Instructions

Set up your computer according to the manufacturer's instructions.

The Exercise Files

The Supplemental Course Materials CD-ROM contains a set of exercise files that you will need to install on your hard disk drive to complete many of the exercises in this book.

To install the exercise files to your hard disk drive

1. Insert the Supplemental Course Materials CD-ROM into your CD-ROM drive.
   If AutoRun is disabled on your machine, refer to the Readme.txt file on the CD-ROM.

This document is created with the unregistered version of CHM2PDF Pilot

2. Select Labs Folder on the user interface menu, and then select the exercise file you want to view. If required for the exercise, copoy the files to a working folder on your hard disk drive.

The eBook

The CD-ROM also includes an electronic version of the book that you can view using a HyperText Markup Language (HTML) browser.

To use the eBook

1. Insert the Supplemental Course Materials CD-ROM into your CD-ROM drive.
   If AutoRun is disabled on your machine, refer to the Readme.txt file on the CD-ROM.
2. Click eBook on the user interface menu and follow the prompts.
   You must have the Supplemental Course Materials CD-ROM inserted in your CD-ROM drive to run the eBook.

Sample Exam Questions

To install the sample exam questions to your hard disk drive

1. Insert the Supplemental Course Materials CD-ROM into your CD-ROM drive.
   If AutoRun is disabled on your machine, refer to the Readme.txt file on the CD-ROM.
2. Click Sample Exam Questions on the user interface menu and follow the prompts.

This document is created with the unregistered version of CHM2PDF Pilot

The Microsoft Certified Professional Program

The Microsoft Certified Professional (MCP) program provides the best method to prove your command of current Microsoft products and technologies. Microsoft, an industry leader in certification, is on the forefront of testing methodology. Microsoft exams and corresponding certifications are developed to validate your mastery of critical competencies as you design and develop, or implement and support, solutions with Microsoft products and technologies. Computer professionals who become Microsoft certified are recognized as experts and are sought after industry-wide.

The Microsoft Certified Professional program offers multiple certifications based on specific areas of technical expertise including:

- **Microsoft Certified Application Developer (MCAD) for Microsoft .NET.** Qualified to develop and maintain department-level applications, components, Web or desktop clients, or back-end data services.
- **Microsoft Certified Solution Developer (MCSD).** Qualified to design and develop custom business solutions with Microsoft development tools, technologies, and platforms and Microsoft Windows architecture.
- **Microsoft Certified Professional (MCP).** Individuals who have demonstrated in-depth knowledge of at least one Microsoft Windows operating system or architecturally significant platform. An MCP is qualified to implement a Microsoft product or technology as part of a business solution for an organization.
- **Microsoft Certified Systems Engineer (MCSE) on Microsoft Windows 2000.** Qualified to effectively analyze the business requirements, and design and implement the infrastructure for business solutions based on the Microsoft Windows 2000 platform and Microsoft .NET Enterprise Servers.
- **Microsoft Certified Systems Administrator (MCSA) on Microsoft Windows 2000.** Individuals who implement, manage, and troubleshoot existing network and system environments based on the Microsoft Windows 2000 and Windows .NET Server operating systems.
- **Microsoft Certified Database Administrator (MCDBA) on Microsoft SQL Server 2000.** Individuals who derive physical database designs, develop logical data models, create physical databases, create data services by using Transact-SQL, manage and maintain databases, configure and manage security, monitor and optimize databases, and install and configure Microsoft SQL Server.
- **Microsoft Certified Trainer (MCT).** Instructionally and technically qualified to deliver Microsoft Official Curriculum through a Microsoft Certified Technical Education Center (CTEC).

Microsoft Certification Benefits

Microsoft certification, one of the most comprehensive certification programs available for assessing and maintaining software-related skills, is a valuable measure of an individual's knowledge and expertise. Microsoft certification is awarded to individuals who have successfully demonstrated their ability to perform specific tasks and implement solutions with Microsoft products. Not only does this provide an objective measure for employers to consider; it also provides guidance for what an individual should know to be proficient. And as with any skills-assessment and benchmarking measure, certification brings a variety of benefits to the individual and to employers and organizations.

Microsoft Certification Benefits for Individuals

This document is created with the unregistered version of CHM2PDF Pilot

As a Microsoft Certified Professional, you receive many benefits:

- Industry recognition of your knowledge and proficiency with Microsoft products and technologies.
- A Microsoft Developer Network subscription. MCPs receive rebates or discounts on a one-year subscription to the Microsoft Developer Network (http://www.msdn.microsoft.com/subscriptions/) during the first year of certification. (Fulfillment details will vary depending on your location; please see your Welcome Kit.)
- Access to technical and product information direct from Microsoft through a secured area of the MCP Web site (go to http://www.microsoft.com/traincert/mcp/mcpsecure.asp).
- Access to exclusive discounts on products and services from selected companies. Individuals who are currently certified can learn more about exclusive discounts by visiting the MCP secured Web site (go to http://www.microsoft.com/traincert/mcp/mcpsecure.asp and select the "Other Benefits" link).
- MCP logo, certificate, transcript, wallet card, and lapel pin to identify you as a Microsoft Certified Professional (MCP) to colleagues and clients. Electronic files of logos and transcript can be downloaded from the MCP secured Web site (go to http://www.microsoft.com/traincert/mcp/mcpsecure.asp) upon certification.
- Invitations to Microsoft conferences, technical training sessions, and special events.
- Free access to *Microsoft Certified Professional Magazine Online*, a career and professional development magazine. Secured content on the *Microsoft Certified Professional Magazine Online* Web site includes the current issue (available only to MCPs), additional online-only content and columns, an MCP-only database, and regular chats with Microsoft and other technical experts.
- Discount on a membership to PASS (for MCPs only), the Professional Association for SQL Server. In addition to playing a key role in the only worldwide, user-run SQL Server user group endorsed by Microsoft, members enjoy unique access to a world of educational opportunities (go to http://www.microsoft.com/traincert/mcp/mcpsecure.asp).

An additional benefit is received by Microsoft Certified System Engineers (MCSEs):

- A 50-percent rebate or discount off the estimated retail price of a one-year subscription to *TechNet* or *TechNet Plus* during the first year of certification. (Fulfillment details will vary depending on your location. Please see your Welcome Kit.) In addition, about 95 percent of the CD-ROM content is available free online at the *TechNet* Web site (http://www.microsoft.com/technet/).

An additional benefit is received by Microsoft Certified Database Administrators (MCDBAs):

- A 50-percent rebate or discount off the estimated retail price of a one-year subscription to *TechNet* or *TechNet Plus* during the first year of certification. (Fulfillment details will vary depending on your location. Please see your Welcome Kit.) In addition, about 95 percent of the CD-ROM content is available free online at the *TechNet* Web site (http://www.microsoft.com/technet/).
- A one-year subscription to *SQL Server Magazine*. Written by industry experts, the magazine contains technical and how-to tips and advice a must for anyone working with SQL Server.

A list of benefits for Microsoft Certified Trainers (MCTs) can be found at http://www.microsoft.com/traincert/mcp/mct/benefits.asp.

Microsoft Certification Benefits for Employers and Organizations

Through certification, computer professionals can maximize the return on investment in Microsoft technology. Research shows that Microsoft certification provides organizations with:

- Excellent return on training and certification investments by providing a standard method of determining training needs and measuring results.
- Increased customer satisfaction and decreased support costs through improved service, increased productivity, and greater technical self-sufficiency.
- Reliable benchmarks for hiring, promoting, and career planning.

This document is created with the unregistered version of CHM2PDF Pilot

- Recognition and rewards for productive employees by validating their expertise.
- Retraining options for existing employees so they can work effectively with new technologies.
- Assurance of quality when outsourcing computer services.

Requirements for Becoming a Microsoft Certified Professional

The certification requirements differ for each certification and are specific to the products and job functions addressed by the certification.

To become a Microsoft Certified Professional, you must pass rigorous certification exams that provide a valid and reliable measure of technical proficiency and expertise. These exams are designed to test your expertise and ability to perform a role or task with a product, and are developed with the input of professionals in the industry. Questions in the exams reflect how Microsoft products are used in actual organizations, giving them "real-world" relevance.

- Microsoft Certified Professional candidates are required to pass one operating system exam. Candidates can pass additional Microsoft certification exams to further qualify their skills with other Microsoft products, development tools, or desktop applications.
- Microsoft Certified Systems Engineers are required to pass five core exams and two elective exams.
- Microsoft Certified Systems Administrators are required to pass three core exams and one elective exam that provide a valid and reliable measure of technical proficiency and expertise.
- Microsoft Certified Database Administrators are required to pass three core exams and one elective exam that measure technical proficiency and expertise.
- Microsoft Certified Solution Developers are required to pass three core Microsoft Windows operating system technology exams and one BackOffice technology elective exam.
- Microsoft Certified Trainers are required to meet instructional and technical requirements specific to each Microsoft Official Curriculum course they are certified to deliver. The MCT program requires on-going training to meet the requirements for the annual renewal of certification. For more information about becoming a Microsoft Certified Trainer, visit http://www.microsoft.com/traincert/mcp/mct/ or contact a regional service center near you.

Technical Training for Computer Professionals

Technical training is available in a variety of ways including instructor-led classes, online instruction, or self-paced training available at thousands of locations worldwide.

Self-Paced Training

For motivated learners who are ready for the challenge, self-paced instruction is the most flexible, cost-effective way to increase your knowledge and skills.

A full line of self-paced print and computer-based training materials is available direct from the source Microsoft Press. Microsoft Official Curriculum course-ware kits from Microsoft Press are designed for advanced computer system professionals and are available from Microsoft Press and the Microsoft Developer Division. Self-paced training kits from Microsoft Press feature print-based instructional materials along with CD-ROM-based product software, multimedia presentations, lab exercises, and practice files. The Mastering Series provides in-depth, interactive training on CD-ROMs for experienced developers. They're both great ways to prepare for the Microsoft

This document is created with the unregistered version of CHM2PDF Pilot

Certified Professional (MCP) exams.

Online Training

For a more flexible alternative to instructor-led classes, turn to online instruction. It's as near as the Internet, and it's ready whenever you are. Learn at your own pace and on your own schedule in a virtual classroom, often with easy access to an online instructor. Without ever leaving your desk, you can gain the expertise you need. Online instruction covers a variety of Microsoft products and technologies. It includes options ranging from Microsoft Official Curriculum to choices available nowhere else. It's training on demand with access to learning resources 24 hours a day. Online training is available through Microsoft Certified Technical Education Centers (CTECs).

Microsoft Certified Technical Education Centers

Microsoft Certified Technical Education Centers (CTECs) are the best source for instructor-led training that can help you prepare to become a Microsoft Certified Professional. The Microsoft CTEC program is a worldwide network of qualified technical training organizations that provide authorized delivery of Microsoft Official Curriculum courses by Microsoft Certified Trainers to computer professionals.

For a listing of CTEC locations in the United States and Canada, visit http://www.microsoft.com/traincert/ctec/.

This document is created with the unregistered version of CHM2PDF Pilot

Technical Support

Every effort has been made to ensure the accuracy of this book and the contents of the companion disc. If you have comments, questions, or ideas regarding this book or the companion disc, please send them to Microsoft Press using either of the following methods:

E-mail:	TKINPUT@MICROSOFT.COM
Postal Mail:	Microsoft Press Attn: Editor *MCAD/MCSD Training Kit Developing Windows-Based Applications with Microsoft Visual Basic .NET and Visual C# .NET* One Microsoft Way Redmond, WA 98052-6399

The Microsoft Press Web site (http://www.microsoft.com/mspress/support/) provides corrections for books. Please note that product support is not offered through this Web site. For further information regarding Microsoft software support options, please connect to http://www.microsoft.com/support/ or call Microsoft Support Network Sales at (800) 936-3500.

For information about ordering the full version of any Microsoft software, please call Microsoft Sales at (800) 426-9400 or visit http://www.microsoft.com.

This document is created with the unregistered version of CHM2PDF Pilot

Chapter 1

Introduction to the .NET Framework

About This Chapter

This chapter discusses the Microsoft .NET Framework and the common language run time, and provides an introduction to the syntax of class, structure, and method declaration.

Before You Begin

To complete the lessons in this chapter, there are no prerequisites.

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 1: The .NET Framework and the Common Language Runtime

The Microsoft .NET Framework is an integrated and managed environment for the development and execution of your code. This lesson is an introduction to the .NET Framework, the philosophy behind it, and how it works.

After this lesson, you will be able to

- Describe the elements of the .NET Framework
- Describe the parts of an assembly and identify what is contained in each
- Describe how a .NET application is compiled and executed

Estimated lesson time: 20 minutes

Overview of the .NET Framework

The .NET Framework is a managed, type-safe environment for application development and execution. The framework manages all aspects of the execution of your program: it allocates memory for the storage of data and instructions, grants or denies the appropriate permissions to your application, initiates and manages application execution, and manages the reallocation of memory for resources that are no longer needed. The .NET Framework consists of two main components: the common language runtime and the .NET Framework class library.

The common language runtime can be thought of as the environment that manages code execution. It provides core services, such as code compilation, memory allocation, thread management, and garbage collection. Through the common type system (CTS), it enforces strict type safety, and it ensures that code is executed in a safe environment by enforcing code access security.

The .NET Framework class library provides a collection of useful and reusable types that are designed to integrate with the common language runtime. The types provided by the .NET Framework are object-oriented and fully extensible, and allow you to seamlessly integrate your applications with the .NET Framework. The .NET base class library is discussed further in Lesson 2.

Languages and the .NET Framework

The .NET Framework is designed for cross-language compatibility. Simply put, this means that .NET components can interact with each other no matter what language they were originally written in. So, an application written in Microsoft Visual Basic .NET might reference a DLL file written in Microsoft C#, which in turn might access a

This document is created with the unregistered version of CHM2PDF Pilot

resource written in managed Microsoft C++ or any other .NET language. This language interoperability extends to full object-oriented inheritance. A Visual Basic .NET class might be derived from a C# class, for example, or vice versa.

This level of cross-language compatibility is possible because of the common language run time. When a .NET application is compiled, it is converted from the language it was written in (Visual Basic .NET, C#, or any other .NET compliant language) to *Microsoft Intermediate Language* (MSIL or IL). This is a low-level language designed to be read and understood by the common language run time. Because all .NET executables and DLLs exist as intermediate language, they can freely interoperate. The *Common Language Specification* defines the minimum standards that .NET language compilers must conform to, and thus ensures that any source code compiled by a .NET compiler can interoperate with the .NET Framework.

The CTS ensures type compatibility between .NET components. Because .NET applications are converted to IL prior to deployment and execution, all primitive data types are represented as .NET types. Thus, a Visual Basic *Integer* and a C# *int* are both represented in IL code as a System.Int32. Because both languages use a common and interconvertable type system, it is possible to transfer data between components and avoid time-consuming conversions or hard-to-find errors.

Visual Studio .NET ships with with such languages as Visual Basic .NET, Visual C#, and Visual C++ with managed extensions as well as the JScript scripting language. You can also write managed code for the .NET Framework in other languages. Third party compilers exist for Fortran .NET, Cobol .NET, Perl .Net, and a host of other languages. All of these languages share the same cross-language compatibility and inheritability. Thus you can write code for the .NET Framework in the language of your choice, and it will be able to interact with code written for the .NET Framework in any other language.

The Structure of a .NET Application

To understand how the common language run time manages the execution of code, you must examine the structure of a .NET application. The primary unit of a .NET application is the *assembly*. An assembly is a self-describing collection of code, resources, and metadata. The *assembly manifest* contains information about what is contained within the assembly. The assembly manifest provides

- Identity information, such as the name and version number of the assembly.
- A list of all types exposed by the assembly.
- A list of other assemblies required by the assembly.
- A list of code access security instructions for the assembly. This includes a list of permissions required by the assembly and permissions to be denied the assembly.

Each assembly has one and only one assembly manifest, and it contains all the description information for the assembly. The assembly manifest can be contained in its own separate file, or it can be contained within one of the assembly's *modules*.

An assembly also contains one or more modules. A module contains the code that makes up your application or library, and metadata that describes that code. When you compile a project into an assembly, your code is converted from high-level code to IL. Because all managed code is first converted to IL code, applications written in different languages can easily interact. For example, one developer might write an application in Visual C# that accesses a DLL in Visual Basic .NET. Both resources will be converted to IL modules before being executed, thus avoiding any language incompatibility issues.

Each module also contains a number of types. Types are templates that describe a set of data encapsulation and functionality. There are two kinds of types: *reference types* (classes) and *value types* (structures). These types are discussed in greater detail in Lesson 3. Each type is described to the common language run time in the assembly manifest. A type can contain fields, properties, and methods, each of which should be related to a common

This document is created with the unregistered version of CHM2PDF Pilot

functionality. For example, you might have a class that represents a bank account. It would contain fields, properties, and methods related to the functions needed to implement a bank account. A field represents storage of a particular type of data. You might have a field that stores the name of an account holder. Properties are similar to fields, but usually provide some kind of validation when the data is set or retrieved. You might have a property that represents the balance available in an account. When an attempt is made to change the value, the property could check to see if the attempted change was greater than a predetermined limit, and if so, could disallow the change. Methods represent behavior, such as actions taken on data stored within the class or changes to the user interface. Continuing with the bank account example, you might have a *Transfer* method that transfers a balance from a checking account to a savings account, or an *Alert* method that warns the user when his balance has fallen below a predetermined level.

Compilation and Execution of a .NET Application

When you compile a .NET application, it is not compiled to binary machine code; rather, it is converted to IL, which is a low-level set of instructions understood by the common language run time. This is the form that your deployed application takes one or more assemblies consisting of executable files and DLL files in IL form. At least one of these assemblies will contain an executable file that has been designated as the entry point for the application.

When execution of your program begins, the first assembly is loaded into memory. At this point, the common language run time examines the assembly manifest and determines the requirements to run the program. It examines security permissions requested by the assembly and compares them to the system's security policy. If the system's security policy does not allow the requested permissions, the application will not be run. If the application passes the system's security policy, the common language run time executes the code. It creates a process for the application to run in and begins application execution. When execution starts, the first bit of code that needs to be executed is loaded into memory and compiled into native binary code from IL by the common language run time's Just-In-Time (JIT) compiler. Once compiled, the code is executed and stored in memory as native code, so each portion of code is compiled only once during the execution of an application. Whenever program execution branches to code that has not yet been executed, the JIT compiler compiles it ahead of execution and stores it in memory as binary code. This way, application performance is maximized because only the parts of a program that are executed are compiled.

Lesson Summary

- The .NET Framework is a foundation for software development. The .NET Framework consists of the common language run time, which provides many of the core services required for program execution, and the .NET base class library, which exposes a set of predeveloped classes to facilitate program development. The Common Language Specification defines a minimum set of standards that all languages using the .NET Framework must support, and the CTS ensures type compatibility between components developed in different languages.
- The primary unit of a .NET application is the assembly, which consists of an assembly manifest. The assembly manifest describes the assembly and one or more modules, which contain the source code for the application.
- A .NET executable is stored as an IL file. When loaded, the assembly is checked against the security policy of the local system. If it is allowed to run, the first assembly is loaded into memory and JIT compiled into native binary code where it is stored for the remainder of the program's execution.

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 2: The .NET Base Class Library

The .NET base class library is a collection of object-oriented types and interfaces that provide object models and services for many of the complex programming tasks you will face. Most of the types presented by the .NET base class library are fully extensible, allowing you to build types that incorporate your own functionality into your managed code. In this lesson, you are introduced to some of the .NET base class library namespaces and learn how to reference the library and use its types and methods.

After this lesson, you will be able to

- Describe the .NET base class library
- Explain the difference between value types and reference types
- Create a reference to a namespace
- Create an instance of a .NET Framework's class and value type

Estimated lesson time: 30 minutes

The .NET Framework base class library contains the base classes that provide many of the services and objects you need when writing your applications. The class library is organized into *namespaces*. A namespace is a logical grouping of types that perform related functions. For example, the *System.Windows.Forms* namespace contains all of the types that make up windows forms and the controls used in those forms.

Namespaces are logical groupings of related classes. The namespaces in the .NET base class library are organized hierarchically. The root of the .NET Framework is the *System* namespace. Other namespaces can be accessed with the . (period) operator. Typical namespace construction appears as follows:

```
System
System.Data
```

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 3: Using Classes and Structures

You have seen how the .NET Framework base class library provides a plethora of standard types to help you in the development of your applications. You can also create user-defined types that implement custom behaviors. Classes and structures represent the two principal user-defined types.

After this lesson, you will be able to

- Create a new class or structure
- Create an instance of a class or structure
- Explain the difference between a class and a structure
- Create a nested type

Estimated lesson time: 30 minutes

Classes are templates for *objects*. They describe the kind and amount of data that an object will contain, but they do not represent any particular instance of an object. A real-world example of a class might be "Car" the abstract idea of what a car is. You know that a car has an engine and four wheels, a color, a fuel efficiency, and a dozen other properties. Although the *Car* class would describe all of these properties as well as have descriptions of actions the car might perform (go forward, turn on windshield wipers, etc.), the class would not represent any *particular* car. Your car, on the other hand, is an object. It has a specific color, a specific fuel efficiency, a specific kind of engine, and four specific wheels. A different car might have different values for each of these properties, but both would be recognizable as being an instance of the *Car* class.

Members

Classes describe the properties and behaviors of the objects they represent through members. Members are methods, fields, properties, and events that belong to a particular class. Fields and properties represent the data about an object the color of the car, its fuel efficiency, and whether it has an automatic or manual transmission, for example. A method represents something the object can do, such as go forward or turn on headlights. An event represents something interesting that happens to the object, such as overheating or crash.
This chapter limits the discussion to fields and methods. Properties and events are discussed in Chapter 3.

Creating Classes

You create a new class by using the *Class* (Visual Basic .NET) or *class* (C#) keyword. For example:

This document is created with the unregistered version of CHM2PDF Pilot

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 4: Using Methods

Methods do the work of classes and structures. They calculate values, update data, receive input, and perform all the manipulations that make up the behavior of a type. In this lesson, you will learn how to create methods, how to use parameters, and how to create constructors and destructors for your class.

After this lesson, you will be able to

- Create a new method
- Specify return types for your method
- Specify input and output parameters for your method
- Create a constructor and a destructor for your class

Estimated lesson time: 45 minutes

Adding Methods

You can add methods as members to your classes. Methods represent actions your class can take. Methods generally come in two varieties: those that return a value (functions in Visual Basic) and those that do not return a value (Subs in Visual Basic). The following code shows an example of both kinds of methods:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 5: Scope and Access Levels

Access levels define how types are instantiated and how members are accessed. You use access levels to encapsulate data and methods in your types, and to expose functionality to outside objects. In this lesson, you will learn how access modifiers control code access and how to use them in your types.

After this lesson, you will be able to

- Explain the meanings of different access levels and how they affect access to classes, variables, and nested types
- Explain what scope is and how it affects program execution

Estimated lesson time: 20 minutes

You can control how elements of your application are accessed by using access modifiers. Access modifiers are keywords such as *Public* (*public*), *Private* (*private*), and *Friend* (*internal*) that precede a variable or type declaration. The keyword that is used controls the level of access the member is allowed. When an access modifier precedes a member declaration, it affects the *scope* of that member, meaning it controls what code can access it. When a modifier precedes a type declaration, it determines both the scope of its members and how that type is instanced.

Member Access Modifiers

Type members can have modifiers to control their scope. Table 1.2 summarizes the different access levels.

Table 1.2 *Access Levels*

Access Modifier	Affect on Members
Public (Visual Basic .NET) *public* (Visual C#)	Can be accessed from anywhere.
Private (Visual Basic .NET) *private* (Visual C#)	Can be accessed only by members within the type that defines it.
Friend (Visual Basic .NET) *internal* (Visual C#)	Can be accessed from all types within the assembly, but not from outside the assembly.
Protected (Visual Basic .NET) *protected* (Visual C#)	Can be accessed only by members within the type that defines it or types that inherit from that type.

This document is created with the unregistered version of CHM2PDF Pilot

Protected Friend (Visual Basic .NET) *protected internal* (Visual C#)	Can be accessed from all types within the assembly or from types inheriting from the owning type. This is the union of *Protected* (*protected*) and *Friend* (*internal*) access.

Any member with the *Public* (*public*) modifier is visible to all code outside the class. Thus, other objects can access and modify public fields, and can call public methods. Conversely, *Private* (*private*) methods are visible only inside the type to which they belong and cannot be accessed from the outside. A third access modifier, *Friend* (*internal*), indicates that members can be accessed by other types in the same assembly, but cannot be accessed from types outside the assembly. The *Protected* (*protected*) modifier allows access from within the type to which the member belongs and to any types that inherit that type. The *Protected Friend* (*protected internal*) level provides the union of *Protected* (*protected*) and *Friend* (*internal*) access. For member variables, the access modifier can replace the *Dim* statement. If the *Dim* statement is used (in Visual Basic .NET) or no access modifier is used (in Visual C#), the variable is considered *private* in Visual C# and Visual Basic .NET classes, *Public* in Visual Basic .NET structures, and *private* in Visual C# structures. Methods do not require an access modifier. If no access modifier is specified, the method is *Private* (*private*) by default if in a class or structure in C#, and *Public* (*public*) if in a class or structure in Visual Basic .NET.
Inheritance is discussed in depth in Chapter 4.

The following example demonstrates how to use the access modifiers and illustrates how they control access:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 6: Garbage Collection

The automatic memory management scheme employed by the .NET Framework is called *garbage collection* (GC). Memory from objects that are no longer used is traced and reclaimed without any action required by the application. In this lesson, you learn how garbage collection works.

After this lesson, you will be able to

- Describe how garbage collection manages the reclamation of unused memory
- Describe how garbage collection deals with circular references

Estimated lesson time: 15 minutes

The .NET Framework employs automatic memory management. This means that when an object is no longer being used, the .NET Framework can automatically reclaim the memory that was being used by that object. This process is called garbage collection. Consider the following example:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lab 1: Classes and Garbage Collection

In this lab, you will practice creating classes and members, and create a demonstration of how garbage collection automatically manages memory. You will create a class that interacts with a premade user interface. This class will have a shared variable that keeps track of the number of instances that currently exist in memory. Additionally, you will add code to the constructor and destructor of this class to increment and decrement this variable. You will then create multiple instances of this class and watch as their memory is reclaimed by garbage collection. The solution to this lab is available in the \Labs\Ch01\Solution folder on the Supplemental Course Materials CD-ROM.

For this and all of the labs in this book, you will find Visual Basic .NET and Visual C# solutions in their respective folders in *\Labs\Chxx\Solutions*.

Before You Begin

There are no prerequisites to complete this lab.

Estimated lab time: 20 minutes

Exercise 1.1: Making the Demo Class

In this exercise, you will create the Demo class that interacts with the DemoTest project. The DemoTest project is available in the \Labs\Ch01\Partial folder on the CD-ROM.

To make the Demo class

1. Open the DemoTest.sln solution in \Labs\Ch01\Partial. This solution contains all of the front-end code you will need for your project.
2. From the Project menu, choose Add Class.
3. In the Add New Item dialog box, name your class **Demo.vb** or **Demo.cs**, depending on the language you are using.
4. Add a public, shared field named Instances to Demo. This field will track the number of instances of Demo that are currently in memory. The following line shows an example:

This document is created with the unregistered version of CHM2PDF Pilot

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Review

The following review questions are intended to reinforce key concepts and information presented in this chapter. If you are unable to answer a question, return to the appropriate lesson and review, then try the lesson again. Answers to the questions can be found in Appendix A.

1. Briefly describe the major components of the .NET Framework and describe what each component does.
2. Briefly explain what is meant by a reference type and a value type.
3. How do you enable your application to use .NET base class library members without referencing their fully qualified names?
4. Briefly describe how garbage collection works.
5. Briefly describe what members are, and list the four types of members.
6. Explain what constructors and destructors are and describe what they are used for.
7. Briefly explain the difference between *Public* (*public*), *Friend* (*internal*), and *Private* (*private*) access levels as they apply to user-defined types and members.
8. Do you need to instantiate a class before accessing a *Shared* (*static*) member? Why or why not?

This document is created with the unregistered version of CHM2PDF Pilot

Chapter 2

Creating the User Interface

About This Chapter

This chapter describes how to create the user interface of your form. Essential design elements such as composition, use of controls and menus, and working with forms will be covered. Additionally, you will learn how to validate user input.

Before You Begin

To complete the lessons in this chapter, there are no prerequisites.

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 1: User Interface Design Principles

The user interface provides a mechanism for users to interact with your application. Efficient design of the user interface with an eye towards ease of use is of paramount importance when creating an application. This lesson presents guidelines for designing user friendly, elegant, and simple user interfaces.

When designing the user interface, the primary consideration should be the end users of the application. These end users are commonly called the *target audience*. If you know your target audience, designing the user interface will be effortless. A well designed user interface makes it easy for the target audience to learn and use the application. A poorly designed user interface, on the other hand, can lead to frustration and ultimately inefficiency as your application is avoided or even discarded.

The primary element of a Microsoft Windows application is the form. Forms provide the foundation for each level of user interaction. Controls and menus can be added to provide specific functionality. In addition to being functional, your user interface should be attractive and inviting to the user. The .NET Framework provides support for a variety of graphical effects that aid in the visual presentation of your application including shaped forms and controls, transparent or translucent elements, and complex shading effects.

After this lesson, you will be able to

- Describe the importance of the user interface
- Explain the roles of forms, controls, and menus in the user interface
- Explain the significance of the basic design principles of composition and color
- Explain the use of images, icons, and fonts in the interactive design

Estimated lesson time: 30 minutes

Forms, Controls, and Menus

Forms are the primary element of your user interface. They are the foundation on which your application is built. They generally contain a set of related information or options that provide the user with the information needed to proceed. Every form is also a class, and you can create multiple instances of a form or inherit from a form. These issues are discussed further in Lesson 2 as well as in Chapter 4.

Controls are how information and options are made accessible to end users. Controls such as labels and picture boxes can display information. Controls such as text boxes, list boxes, or combo boxes can both display information and accept input from the user. Controls such as buttons can be used to allow the user to select a course of action. The use of controls is examined in Lesson 3.

Menus and toolbars provide a structured way to expose the commands available to the user of an application. Menus are commonly used to provide access to higher level commands that might be common to all the forms of an

This document is created with the unregistered version of CHM2PDF Pilot

application, such as commands to save data or exit the application. Menus can present such options in a logical, consistent manner that enhances the user experience and enables rapid mastery of the application. Elements of menus can be enabled or disabled to customize options available to the user at different points in the application. The use of menus is discussed in Lesson 4.

Composition

Composition is the look and feel of your user interface. How your user interface is composed will influence how rapidly your application is learned and adopted. Primary elements of composition include

- Simplicity
- Positioning of controls
- Consistency
- Aesthetics

Simplicity

Simplicity is one of the most important aspects of a user interface. If a user interface is busy or overly complex, it makes learning and using the application difficult and time-consuming. A user interface should allow the user to quickly complete all of the interactions required by the program, but should only expose the functionality needed at that stage of the application.

When designing your user interface, you should keep program flow and execution in mind. Controls that display related data should be visually grouped on the form. Controls such as list boxes, combo boxes, and checked list boxes can be used to display data and allow users to choose between preset options. The use of a tab order allows users to rapidly navigate the fields.

A common mistake when designing user interfaces is to try to reproduce a real-world object. For instance, if you are creating a form that takes the place of a paper form, it is natural to attempt to reproduce the paper form in the application. For some applications, this might be an appropriate approach, but it might be an approach that limits the application and provides no real benefit to the user. A reproduction of a paper form is usually limited in the functionality it can expose. You should think about your particular situation and try to use the added capabilities of the computer to enhance your user's experience.

Another way to simplify your user interface is through the use of default values. For example, if 90 percent of the users of an application will select Washington in a State field in your application, make Washington the default choice for that field. You should always make it easy to override the default value when necessary.

The best information to use when designing a user interface is input from the target audience. Find out how an application will be used and tailor your interface to make frequent tasks easy to perform. After your application is completed, input from the users is useful for continued improvement.

Positioning of Controls

The placement of controls should reflect their relative importance and frequency of use. For example, you might have a form that is used to input both required information and optional information. The controls for the required information are clearly more important and should receive greater prominence. Most user interfaces are designed to be read left-to-right and top-to-bottom. The most important or frequently used controls will be most easily accessed at the top of the form. Controls that will be used after a user completes an action on a form, such as a Submit button,

This document is created with the unregistered version of CHM2PDF Pilot

should follow the logical flow of the application and be placed at the bottom of the form.

Another consideration is relatedness of information. Related information should be displayed in visually grouped controls. For example, you might have a form that displays information about a customer, a purchase, or an employee. You might group each set of controls on a *Tab* control so that a user can easily move back and forth between displays.

Aesthetics is also an important consideration in the placement of controls. You should try to avoid *busy* forms that display more information than can be taken in at one time. Whenever possible, information displaying controls should be spaced adequately to create visual appeal and ease of accessibility.

Consistency

Your user interface should have a consistent look and feel to it. An inconsistent interface will feel less cohesive and consequently can make your application seem disorganized or chaotic, hindering adoption among end users.

The key to consistency is in the design process. Before any actual application development takes place, you should decide upon a consistent visual scheme for your application. Colors, fonts, size, and types of control all factor into decisions regarding your program's presentation. Avoid the temptation to *show-off*. Use only what is needed to accomplish the task. Extraneous use of controls or flashy and out-of-place visual elements will only distract a user and in the end make your application less efficient.

Each form of your application should be consistent. The user should not need to reacquaint himself or herself with the visual elements of the interface as he navigates from form to form.

Aesthetics

Whenever possible, an application should project an inviting and pleasant user interface. Although clarity and simplicity should not be sacrificed for the sake of attractiveness, you should endeavor to create an application that will not dissuade users.

Color

The use of color is one way to make your user interface attractive and inviting to use. However, it is easy to overuse color. Loud, vibrant colors might be visually appealing to some people, but will cause others to react with discomfort. When designing a background color scheme for your application, it is best to use muted, agreeable colors.

An important consideration when designing your color scheme is your target audience. If you are designing an application for a company, you might consider using that company's corporate color scheme for your application. When designing for international audiences, be aware that certain colors might have cultural significance. Always research any special meanings that the use of color might have in your particular context. Maintain consistency, and do not overdo it.

Think about how your use of color affects usability. For example, if you use blue text on a black background, your text might be difficult to read and thus impair usability. Also, be aware of color blindness because it might relate to usability. Some people, for example, are unable to perceive a distinction between red and green. Thus, red text on a green background would be invisible to someone with this condition. Keep contrast high in important elements of your application, and do not rely on color alone to convey information.

Fonts

This document is created with the unregistered version of CHM2PDF Pilot

The fonts you choose for your application should also be chosen on the basis of usability and should be kept consistent throughout the application. Avoid fonts that are difficult to read or are highly embellished. Stick to simple, easy-to-read fonts such as Palatino or Times New Roman. Use script or decorative fonts only for visual effects, such as a title page if appropriate, never to convey important information.

Images and Icons

Pictures and icons can add visual interest to your application, but careful design is essential. Images that appear busy or distract the user will hinder use of your application. Icons have been successfully used in many applications to convey information to the user in a nontextual manner, but again, careful consideration of the end user is required. For example, you might use a red octagon (like a stop sign) to indicate that a user might not want to proceed. Some countries, however, use differently shaped traffic signs, and thus the significance of this icon might be lost on an international audience. Know your end users, and tailor the use of icons and visual elements to them.

Whenever possible, icons should be kept to simple shapes that are easily rendered in a 16-by-16 pixel square. Complex pictures can suffer a severe loss of resolution when degraded and thus become unusable.

Shapes and Transparency

The .NET Framework provides tools that allow you to create forms and controls with varying levels of opacity using shapes other than the traditional rectangle. Although these tools can provide some very powerful visual effects for your interface, these effects should not be overused. For example, although it might be interesting and unique to have a text box shaped like a doughnut, it would hardly be efficient or enhance the usability of your application. Always keep the end user in mind when employing these effects. Controls or forms of different shapes (such as round buttons) might make a user interface more attractive and allow consistency throughout an application as long as the effects are applied tastefully. Similarly, translucent forms might be used to allow a user to manipulate a form in the foreground while monitoring action on a background form. The aesthetics of your application should always serve the ultimate purpose usability.

Lesson Summary

- Interface design is important because a visually consistent and logically designed interface will be easier to learn and will be more efficiently used.
- Major elements of a user interface include forms, controls, and menus.
- A good user interface will be carefully designed with attention to the following principles:
  - Simplicity
  - Positioning of Controls
  - Consistency
  - Aesthetics
- Make choices that invite the target audience to use your application. Be aware of the cultural significance of your choices, and keep the principles of consistency and simplicity foremost in your mind when designing your interface.

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 2: Using Forms

Forms are the fundamental unit of your user interface. They provide the foundation for user interactions by creating a backdrop that hosts the controls and allows you to present your application in a consistent and attractive manner. Forms can display data and receive user input. Although it is possible to create an application that has no forms at all, such as a Windows service or a console application, applications that have frequent user interaction will usually contain at least one form. More complex applications might require several forms to allow the program to flow in a consistent and logical manner.

After this lesson, you will be able to

- State the role of forms in an application
- Explain how to add forms to your application
- Explain how to set the startup form and the startup location
- Explain how to set the visual appearance of your form
- Explain how to use form methods
- Explain how to use form events

Estimated lesson time: 30 minutes

Adding Forms to Your Project

Forms enable interaction between your application and the user. When you create a new Windows forms project, an initial form named Form1 is added to your project by default. This is not actually an instance of a form, but rather a class that represents the code behind an instance of that form. You can edit this form in the graphical designer by adding controls, menus, and other visual elements to it. As your application grows in size, you will want to add additional form classes to your project.

To add a new form to a project

1. On the Project menu, click Add Windows Form.

   The Add New Item dialog box opens.
2. Click Windows Form, and click Open.

   A new form is added to the development environment.

You can also add a new form using code. In this case, you declare a variable that represents a type of Form and create an instance of that Form. This Form can then be used and displayed during execution of your application. Note that you will be unable to use any visual designers to create this form, and it will not be available at design time. Oftentimes, you would employ this method if you want to display a form that you previously designed.

To add a form to your application at run time

This document is created with the unregistered version of CHM2PDF Pilot

Declare and instantiate a variable representing your form in the same manner as you would any other class. For example:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 3: Using Controls and Components

Controls (also referred to as Windows Forms controls) are the second element of the visual interface. They are graphical tools that create or enhance the functionality of an application. Tools are added to the form from the Toolbox. The Toolbox contains a variety of controls. Some, such as *Button* and *TextBox*, are designed to receive user input and carry out the basic tasks associated with user interaction. Others are specialized components designed to manage complex interactions with other parts of the application. Components are similar to controls in that they are premade units of code that encapsulate specific functionality. The main difference between components and controls is that controls have a visual representation, whereas components do not. This lesson provides information on using controls and components when building a user interface.

After this lesson, you will be able to

- Define the role of controls in your application
- Explain the difference between controls and components
- Explain how to set the Control Tab order
- Describe which controls can contain other controls and how they are used
- Describe docking and anchoring, and explain their use with controls
- Explain how to dynamically add controls to your form
- Describe the process for adding controls to the Toolbox
- Describe how to create event handlers for controls
- Explain what an extender is and how to use one

Estimated lesson time: 45 minutes

Working with Controls

The Visual Studio Toolbox contains a variety of preset controls that you can use to develop your solutions. Controls are hosted in forms and implement most of the actual functionality of the user interface. Take the *Button* control, for example. It can be placed upon a form, and it will be displayed, usually with a title that provides some kind of information as to its function. When the user clicks the button with the mouse, an event handler responds to the click and causes code to execute. Other controls such as *Label* controls and *PictureBox* controls are primarily used to display data to the user. Controls such as *TextBox* and *ListBox* serve a dual purpose; both display information and allow the user to input information to the application.

You can add controls to the forms in your application by using the designer. The designer displays the form you are currently composing in a graphical state, similar to how the form will look at run time. The Toolbox allows you to graphically select a control with the mouse and add it to the surface of the form in the designer. All of the code necessary to add this control is added to your application automatically. You can reposition your controls by clicking and dragging them with the mouse. You can resize most controls by grabbing the edge with the mouse and adjusting the size. Because an in-depth discussion of the different controls and their functionality is beyond the scope of this

This document is created with the unregistered version of CHM2PDF Pilot

book, you should familiarize yourself with the controls in the Toolbox and how they work.

To add a control to your application

1. Select the control you want to add in the Toolbox.
2. Click the form in the location where you want your control to appear. While holding down the left mouse button, draw the control to the size you want it to be on the form.

   Alternatively, you can drag the control from the Toolbox onto the form, or double-click the desired control in the Toolbox. The control will be added to the form with default values for size, and position if appropriate.
3. Use the mouse to set the size and position of the control as desired. You can also use the arrow keys to position the control.

The Properties window displays all of the properties of the selected control and allows you to edit them graphically. Setting a property at design time sets the default value of a property, which can still be changed in the course of code execution. You can set the position of the control by positioning the control on the form in the designer. Although you can accomplish all of the same tasks in code, the designer allows you to rapidly set startup properties for your controls without spending valuable development time hand coding values for each and every control on your form.

To edit properties of a control at design time

1. Right-click a control and select Properties, or left-click the control and press F4, or choose Properties from the View menu.
2. Edit the value in the appropriate property box.

To edit properties for multiple controls at design time

1. Click and drag the mouse over the controls you want to select, or hold down the Ctrl key and click the controls you want to select.

   The Properties window displays the properties that are common to all controls selected.
2. Edit the value in the appropriate property box.

You will also find components in the Toolbox. Components are very similar to controls in that they are preassembled units of functionality that you can incorporate into your application. The primary difference between controls and components is that components are not visible in the user interface. An example of a component is the *Timer* component. The *Timer* is a component that raises an event at a specified interval. Because they have no visual interface, components are not added to the form when added to your application in the designer. Instead, they are added to the component tray, a region near the bottom of the designer that allows you to graphically manipulate the properties of nonvisual components of your application. You add a component to your application in the same manner in which you would add a control, and use the Properties window to edit the properties of a component at design time.

Setting the Control Tab Order

The users of your application can use the Tab key to quickly move the focus from one control to another. You can set the order in which the focus is transferred by setting the *tab order*. The tab order is the order in which controls on the form receive the focus and is specified by the *TabIndex* property. To change the order in which a control receives the focus, simply set the *TabIndex* property to a different value. Lower values receive the focus first and proceed numerically through higher values. In the event of a tie between *TabIndex* values, the focus first goes to the control that is closest to the front of the form. You can bring a control to the front of the form or send it to the back

This document is created with the unregistered version of CHM2PDF Pilot

of the form by right-clicking the control and choosing Bring To Front or Send To Back, respectively.

Visual Studio contains a graphical tool for setting the tab order. Under the View menu, choose Tab Order. A box containing a number appears inside each control on the designer. To set the tab order, all you have to do is click on the controls in the order that you want them to receive the focus.
Some controls, such as *PictureBox*, cannot receive the focus and thus do not have a *TabIndex* property.

To set the tab order using the TabIndex property

1. In the designer, select each control capable of receiving the focus.
2. In the Properties window, set the *TabIndex* property to an appropriate value. The focus passes from control to control in the order of lowest to highest *TabIndex*.

To set the tab order graphically with Visual Studio

1. From the View menu, choose Tab Index.

   Boxes containing the current tab order appear in each control.
2. Click each control in the correct tab order.
3. From the View menu, choose Tab Order again to resume editing.

Controls That Can Contain Other Controls

Some controls, known as *container controls*, allow you to place other controls inside of them. Controls that fall into this category include the *Panel* control, the *GroupBox* controls, and the *TabControl* control. Forms can also be thought of as container controls. You can use these controls to logically organize groups of controls or a form. For example, you might group a set of related radio buttons in a *GroupBox* control. Using these controls can provide logical groupings of controls that you can manipulate programmatically. These controls can also help to create a sense of style or flow in your user interface and provide visual cues to the user.
A container control acts as a host for other controls, but is independent of those controls. This is contrasted by user controls, which can bind multiple controls together into a single interdependent unit. User controls will be discussed in detail in Chapter 7.

When a control contains another control, changes in properties of the container control can affect the contained controls. For example, if the *Enabled* property of a *GroupBox* control is set to false, it automatically disables all of the controls contained within it. Likewise, changes in user interface related controls, such as *BackColor*, *ForeColor*, *Visible*, and *Font*, are also applied to the contained controls. This allows you to easily create a consistent look for sections of your user interface. The *Anchor* and *Dock* properties of contained controls also function relative to the containing control. These properties are discussed further in the next section.
Changes in the visual properties of a container control will automatically change the corresponding property in contained controls, but you can still manually change any of these properties on a contained control. If the container control is disabled, however, there is no way to enable a contained control short of reenabling the containing control.

Using the Group Box and Panel Controls

The *GroupBox* control and the *Panel* control are very similar. Both provide a logical and physical grouping of controls. These controls can be thought of as physical subdivisions of a form. Changes in the properties of a *Panel* or *GroupBox* affect all of the controls contained within. Controls contained within a *Panel* or *GroupBox* can be moved and repositioned as a single unit at design time. At run time, you can disable the entire group by setting the *Enabled*

This document is created with the unregistered version of CHM2PDF Pilot

property of the containing control to false.

The *GroupBox* control provides a caption that allows you to label the group of controls within. You can set this caption by setting the *Text* property. The *Panel* control does not provide a caption but is a scrollable container. By setting the *AutoScroll* property to true, you enable scroll bars within the panel.

Using the TabControl Control

The *TabControl* control allows you to group controls onto a set of tabs, rather like files in a filing cabinet or dividers in a notebook. The *TabControl* is a host for a number of *TabPages*, which host other controls. An example of the *TabControl* might be property pages for an application, with each tab representing properties relating to a specific component of the application.

The most important property of the *TabControl* is the *TabPages* property. The *TabPages* property is a *collection* of *TabPage* controls, each with its own set of properties. A collection is a logical organization of objects similar to an array. Collections are discussed in detail in Chapter 3. You can access each *TabPage* by clicking on the tab that represents it, as shown in Figure 2.6.

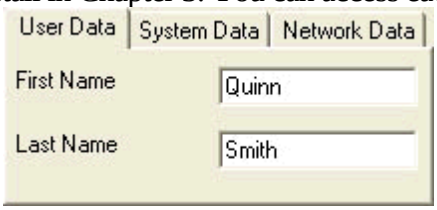


Figure 2.6 *TabPages in a TabControl*

Individual *TabPages* are very similar to *Panel* controls. They provide a scrollable subdivision of a form that can host a set of other controls. By setting the *Autoscroll* property to *true*, you enable the scroll bars of each *TabPage*.

TabPages are added to the *TabControl* by setting the *TabPages* property. When the *TabPages* property is selected in the designer, the *TabPage* collection editor appears. From this window, you can add *TabPages* to your control and set the properties of each. Once added, you can click the appropriate tab in the designer and add controls to the design surface represented by the *TabPage*. At run time, the user is able to move between different tab pages by clicking on the appropriate tabs.

Docking and Anchoring Controls

You can use the *Anchor* and *Dock* properties of a control to dictate how it behaves inside its form or parent control. The *Anchor* property allows you to define a constant distance between a control and one or more of the edges of a form. Thus, if a user resizes a form at run time, the control will always maintain a specific distance from the edge(s) indicated. The *Dock* property allows you to attach a control to an edge of the form or to completely fill the form. A docked control will automatically resize itself when the form is resized.

Using the Anchor Property

You can set the *Anchor* property for any control in the same manner in which you would set any other property at design time. In the Properties window, choose *Anchor*. When the drop-down button is clicked, you are presented with a visual interface to help choose your anchoring properties, as shown in Figure 2.7.

This document is created with the unregistered version of CHM2PDF Pilot

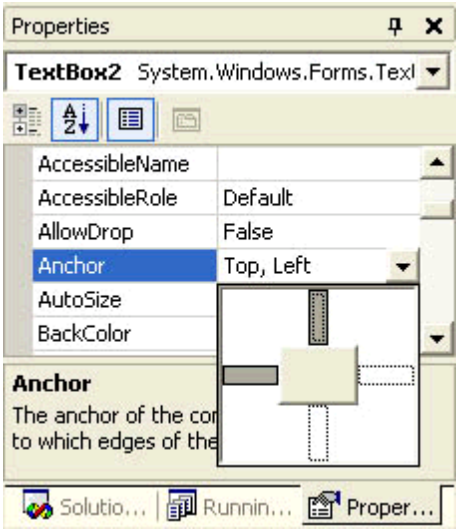


Figure 2.7 *Choosing the Anchor property*

You can use the window shown in Figure 2.7 to choose the edges you want to anchor your control to. The default value for this property is Top, Left, which causes the control to maintain a constant distance between the top and left edges of the form. Because forms are resized from the bottom right corner, the net result of this setting is a control that always maintains its position when the form is resized.

In this exercise, you add a button to a form and set the *Anchor* property to a variety of values to observe the effect the *Anchor* property has on control location and sizing.

To demonstrate the Anchor property

1. In Visual Studio, create a new Windows Forms project.
2. In the designer, add a button to the form.
3. Click the button once, and in the Properties window, select the *Anchor* property.

   The *Anchor* property window appears. Note that the top and left bars are darkened, indicating that the control is currently anchored to the top and left edges of the form.
4. In the designer, resize the window by grabbing the lower right corner with the mouse and dragging. Note that the button maintains its position relative to the top and left edges.
5. In the *Anchor* property window, click both bars. The button is no longer anchored.
6. In the designer, resize the form again. Now the button *floats* on the screen in response to resizing.

   If a control has an *Anchor* property setting of None, it will maintain a proportional distance between the edges instead of an absolute distance.
7. Next, set the *Anchor* property to Top and resize the form.

   Although the button maintains a constant distance from the top edge, it *floats* relative to the left and right edges.
8. In the anchor designer window, click the bottom bar, setting the *Anchor* property to Top, Bottom.
9. Resize the window once more. The button still floats relative to the left and right edges, but it maintains a constant distance between the top and bottom edges. In order to do this, it must resize itself.

The *Anchor* property allows you to manage how controls respond to the resizing of the form. You can allow controls to remain fixed in the form by anchoring them to the top and left edges, cause them to stretch when resized by anchoring them to opposite edges, or allow them to float freely by releasing the anchor. Use of the *Anchor* property allows you to implement a variety of resize and relocation behaviors.

Using the Dock Property

This document is created with the unregistered version of CHM2PDF Pilot

Docking refers to attaching your control to the edge of a parent control. The parent control is usually a form, but can include other container controls such as the *Panel* or *Tab* control. An example of a docked control might be a menu bar at the top of a form.

You can set the *Dock* property of a control at design time in the Properties window. After selecting the *Dock* property, a graphic interface appears that allows you to choose the docking characteristics of your control. The interface is shown in Figure 2.8.

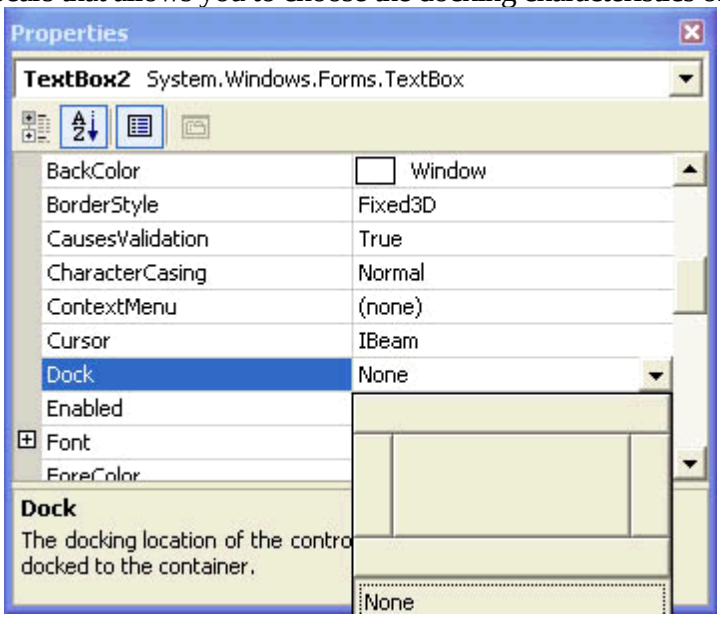


Figure 2.8 *Setting the Dock property*

To set the *Dock* property, simply click on the section of the interface that corresponds to where you want your control to dock. For example, if you want your control to dock at the top of the form, you would click the bar at the top of the interface. To release docking, choose None. Clicking the center square of the *Dock* property interface causes the control to fill the form that it lives in. If you set docking for a control, the *Anchor* property is automatically set to Top, Left.

To set docking characteristics for your control

1. In the Properties window, choose the *Dock* property.

   The *Dock* property visual interface appears.
2. Click the appropriate section of the interface to choose your dock setting.

Using the Controls Collection

Every container control, including forms, exposes a collection of all the controls it contains. This is called the *controls collection*. The controls collection exposes a *Count* property, which returns the number of items contained within it, and an *Item* property, which returns a specific item. The controls collection also has member methods that can be used to add to and remove controls from the collection.

In Visual Basic .NET, the *Item* property returns a specific control based on the *Index* of the item. In Visual C#, the indexer of the property is used. For example:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 4: Using Menus

Menus allow your users to easily access critical application functions and tools. Proper design and planning of menus ensures proper functionality and accessibility of your application to the end users.

After this lesson, you will be able to

- Explain the importance of menus in interface design
- Describe the process of creating a menu using the *MainMenu* component
- Describe the process of creating a context menu using the *ContextMenu* component
- Explain how to enable or disable a menu item
- Explain how to create shortcut keys for menu items
- Explain how to display a check mark or radio button on a menu item
- Explain how to make menu items invisible
- Explain how to dynamically add items to a menu
- Explain how to dynamically clone a menu

Estimated lesson time: 30 minutes

Menus allow users to access top-level commands and functions in a familiar, easy to understand interface. A well designed menu that exposes your application's functionality in a logical, consistent manner will make your application easy to learn and use. A poorly designed menu, on the other hand, will be avoided whenever possible and used reluctantly only when necessary.

When designing menus, you should take into account the logical flow of the application. Menu items should be grouped according to related functionality. Using access keys to enable keyboard shortcuts to menu items will also make the use of your application easier.

Creating Menus at Design Time

Main menus are created at design time with the *MainMenu* component. The *MainMenu* component contains and manages a collection of *MenuItem* controls, which form the visual element of a menu at run time. With the *MainMenu* component, you can rapidly and intuitively create menus for your forms.

Using the MainMenu Component

The *MainMenu* control allows you to do the following at design time:

- Create new menus and menu bars.
- Add new menu items to existing menus.
- Modify the properties of menus and menu items via the Properties window.

This document is created with the unregistered version of CHM2PDF Pilot

- Create event handlers to handle the *Click* event and other events for menu items.

To create a new menu, all you have to do is add a *MainMenu* component to your form. The component appears in the component tray, and a box with the text *Type Here* appears in the menu bar of the form. To create a new menu item, type in the box where indicated. The menu appears on your form as it would at run time. As you type, additional boxes are created beneath and to the right of the first menu item. Submenus are created the same way. If you want to create a submenu, simply type an entry to the right of the menu item that you want to expand. Figure 2.9 shows how to use the *MainMenu* component to create menus.

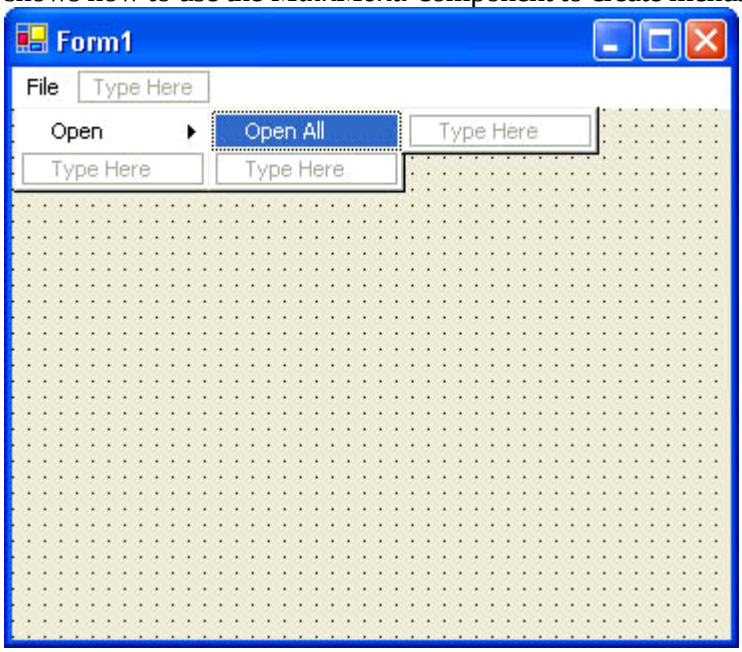


Figure 2.9 *Creating menus with the MainMenu component*

When an item is added to a menu, the designer creates an instance of a *MenuItem* object. Each *MenuItem* object has its own properties and members that can be set in the Properties window. The *Text* property represents the text that will be displayed at run time and is automatically set to the text that you type. The *Name* property indicates how you will refer to this object in code and is automatically given a default value that can be changed if desired.

To create main menus at design time

1. In the Toolbox, add a *MainMenu* component to the form either by double-clicking the MainMenu tool or by dragging it onto the form. A *MainMenu* component appears in the component tray.
2. In the designer, type the text for the first menu item in the box presented on the form's menu bar. As additional boxes appear, add additional menu items until the structure of your menu is complete.
3. In the Properties window, set any of the properties that you want to change for your menu items.
4. In the Properties window of the form, make sure that the *Menu* property of the form is set to the menu you want to display. If you have multiple menus on a form, only the designated menu will be displayed.

Separating Menu Items

You can separate menu items with a separator. A separator is a horizontal line between items on a menu. You can use separator bars to divide menu items into logical groups on menus that contain multiple items, as shown in Figure

This document is created with the unregistered version of CHM2PDF Pilot

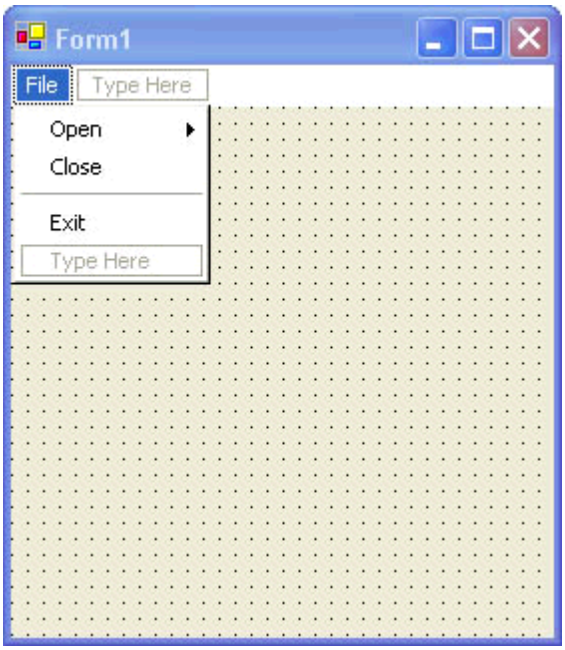
2.10. 

Figure 2.10 *Separator bars on menus*

You can add a separator to your menus by entering a hyphen as the text of a menu item. It will be displayed as a separator.

To create a separator bar on your menu

1. Select the menu item that represents where you want your separator to be.
2. Type a hyphen (-). This will be displayed at run time as a separator bar.

Menu Access and Shortcut Keys

You can enable keyboard access to your menus through the use of Access and Shortcut keys.

Access Keys

Access keys allow the user to open a menu by pressing the Alt key and typing a designated letter. When the menu is open, you can select a menu command by pressing the Alt key and the correct access key. For example, in most programs the Alt+F key opens the File menu. When it is open, you can choose Alt+N to create a new item. Access keys are displayed on the form as an underlined letter on the menu items.

You can use the same access key for different menu items as long as the menu items are contained in different menu groups. For example, you might want to use Alt+C to access the Close command on the File menu group and the Copy command on the Edit menu group. You should avoid using the same access key for multiple items on a menu group, such as using Alt+C for both the Cut and the Copy commands of an Edit menu group. If you do use the same access key combination for two items on a menu group, you will be able to use the access key to toggle your selection between the items, but will not be able to select the item without pressing the Enter key.

To assign an access key to a menu item

1. In the designer, click the menu item that you want to assign an access key.
2. Type an ampersand (&) in front of the desired letter for the access key.

This document is created with the unregistered version of CHM2PDF Pilot

Shortcut Keys

Shortcut keys enable instant access to menu commands. They can be useful for providing a keyboard shortcut for frequently used menu commands. Shortcut key assignments can either be single keys, such as Delete, F1, or Insert, or key combinations, such as Ctrl+A, Ctrl+F1, or Ctrl+Shift+X, for example. When a shortcut key is designated for a menu item, it is shown to the right of the menu item. The shortcut key combination will not be displayed if the *ShowShortcut* property of the menu item is set to *false*.

To assign a shortcut key

1. Select the menu item for which you want to enable a shortcut key.
2. In the Properties window, select the *Shortcut* property.
3. Choose the appropriate shortcut key combination from the drop-down menu.

Using Menu Item Events

You can create event handlers for menu items in the same manner in which you create event handlers for other controls. The most frequently used event is the *Click* event. The *Click* event handler should contain the code to be executed when the menu item is clicked. This code will also execute when a shortcut key combination is pressed.

The *Select* event is raised when a menu item is highlighted, either with the mouse or through the use of access keys. You might create an event handler that provides detailed help regarding the use of a menu command when selected.

The *Popup* event is raised just before a menu item's list of menu items is displayed. You can use this event to enable and disable menu items at run time before the menu is displayed.

See Lesson 2 for information on creating an event handler.

Creating Context Menus

Context menus are menus that appear when an item is right-clicked. You create context menus with the *ContextMenu* component. The *ContextMenu* component is edited in exactly the same way the *MainMenu* component is edited. The *ContextMenu* appears at the top of the form, and you can add menu items by typing them on the control.

Context menus are very similar to main menus in many respects. Both contain and manage a collection of menu item controls. You can enable shortcut keys for menu items in a context menu, but not access keys. To associate a context menu with a particular form or control, set the *ContextMenu* property of that form or control to the appropriate menu.

To create a context menu

1. In the Toolbox, add a *ContextMenu* component to the form, either by double-clicking the *ContextMenu* tool or by dragging it onto the form. A *ContextMenu* component appears in the component tray.
2. In the designer, type the text for the first menu item in the box presented on the form's menu bar. As additional boxes appear, add additional menu items until the structure of your menu is complete.

This document is created with the unregistered version of CHM2PDF Pilot

3. In the Properties window, set any of the properties that you want to change for your menu items.
4. Select the form or control you want to associate the context menu with. In the Properties window for the control, set the *ContextMenu* property to your context menu. The context menu will be displayed at run time when the control is right-clicked. You can associate a single context menu with several controls, but only one context menu per control.

Modifying Menus at Run Time

You can dynamically manipulate your menus to respond to run-time conditions. For example, if your application is unable to complete a certain command, you can disable the menu item that calls that command. You can display a check mark next to a menu item or a radio button to provide information to the user. You can make menu items invisible at times that it might not be appropriate to choose them. You can add menu items at run time, and menus can be cloned or merged with one another at run time.

Enabling and Disabling Menu Commands

Every menu item has an *Enabled* property. When this property is set to *false*, the menu is disabled and does not respond to user actions. Access and shortcut key actions are also disabled for this menu item, and it appears dimmed on the user interface. The following example demonstrates how to disable a menu item at run time.

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 5: Validating User Input

In most applications, the user enters information for the application through the user interface. Data validation ensures that all data entered by a user falls within acceptable parameters before proceeding with program execution. For example, you might have a field where a user enters a zip code as part of an address. Using validation, you could verify that the field contained five and only five characters, all of which were numeric, before proceeding. Validating user input reduces the chance of an input error and makes your application more robust.

In this lesson, you will learn how to use events to validate user input and direct the focus on your forms. You will learn to use field-level validation, which validates entries as they are made, and form-level validation, which validates all of the entries on a form at once. You will learn to use control properties to help restrict input and use the *ErrorProvider* component to provide informative error messages to your users.

After this lesson, you will be able to

- Explain the difference between form-level and field-level validation
- Direct the focus using control methods and events
- Implement form-level validation for your form
- Implement field-level validation for your form

Estimated lesson time: 30 minutes

You can choose between two different types of validation for user input: form-level validation and field-level validation. Form-level validation verifies data after the user has filled all of the fields. For example, a user might be directed to fill in a name, address, and phone number on a form and then click OK. With form-level validation, all of the fields on the form would be validated when the user clicked OK.

Field-level validation, on the other hand, verifies the data in each field as the field is filled in. For example, if a user fills in a field that holds a phone number, field-level validation could verify that the number contains a valid area code before moving to the next field. As each digit is entered, you could also use control events to verify that only numbers are entered.

Field-Level Validation

You might want to validate data as it is entered into each field. Field-level validation gives the developer control over user input as it occurs. In this section, you will learn how to use control events to validate user input, and how to use some properties of the *TextBox* control to help restrict input to appropriate parameters.

Using TextBox Properties

The most commonly used control for user input is the *TextBox*. Several properties of the *TextBox* control allow you to restrict the values of user input that they will accept. Some of these properties include

This document is created with the unregistered version of CHM2PDF Pilot

- *MaxLength*
- *PasswordChar*
- *ReadOnly*
- *MultiLine*

Setting the MaxLength Property

The *MaxLength* property limits the number of characters that can be entered into a text box. If the user attempts to exceed the number returned by *MaxLength*, the text box will accept no further input, and the system beeps to alert the user. This property is useful for text boxes that always contain data of the same length, such as a zip code field.

Using the PasswordChar Property

The *PasswordChar* property allows you to hide user input at run time. For example, if you set the *PasswordChar* property to an asterisk (*), the text box will display an asterisk for each character regardless of user input. This type of behavior is commonly seen in password login boxes.

Although the password character is most commonly an asterisk, you can set it to any valid character. Thus, you could display all semicolons or ampersands, for example. The value of the *Text* property will always be set to the value the user enters regardless of the password character.

Setting the ReadOnly Property

The *ReadOnly* property determines whether a user can edit the value displayed in a text box. If *ReadOnly* is set to true, the text cannot be changed by user input. If *ReadOnly* is set to false, the user can edit the value normally.

Using the MultiLine Property

The *MultiLine* property determines whether a text box can accept multiple lines. When set to true, the user can enter multiple lines in the text box, each separated by a carriage return. The individual lines are stored as an array of strings in the *TextBox.Lines* collection and can be accessed by their index.

Using Events in Field-Level Validation

Field-level keyboard events allow you to immediately validate user input. Controls that can receive keyboard input raise three keyboard events. They are

- *KeyDown*
- *KeyPress*
- *KeyUp*

KeyDown and KeyUp

The *KeyDown* and *KeyUp* events are raised when a key is pressed and a key is released, respectively. The control

This document is created with the unregistered version of CHM2PDF Pilot

that has the focus raises the event. When these events are raised, they package information about which key or combination of keys has been pressed or released in an instance of *KeyEventArgs*, a class that describes the key combination pressed. A method that handles the *KeyDown* or *KeyUp* event must include a *KeyEventArgs* parameter in its signature.

The *KeyUp* and *KeyDown* events are most commonly used for determining if the Alt, Ctrl, or Shift key has been pressed. This information is exposed through properties in the *KeyEventArgs* reference that is passed to the handler. The *KeyEventArgs* properties Alt, Control, and Shift are properties that return a Boolean value, which indicates whether or not those keys are down. A value of true is returned if the corresponding key is down, and false is returned if the key is up. The following example demonstrates a *KeyUp* event handler that checks whether the Alt key is pressed.

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lab 2: The Virtual Doughnut Factory

In this lab, you will create the user interface for a Virtual Doughnut Factory application in either Visual Basic .NET or Visual C#. Your application will serve as a storefront and control panel for an automated doughnut factory. You will add controls, menus, and implement validation. The solution to this lab is available in the \Labs\Ch02\Solution folder on the Supplemental Course Materials CD-ROM.

Before You Begin

There are no prerequisites to completing this lab.

Estimated lab time: 60 minutes

Exercise 2.1: Creating the User Interface

In this exercise, you will create the user interface by adding controls to the Joe's Doughnut Factory inventory form.

To create a new project

1. Select either Visual Basic .NET or Visual C# as your preference. Open a new Windows Application project and name it Joe's Doughnut Factory.
2. The project is opened with one form. In the Solution Explorer, click Form1. The Properties window displays the File properties for Form1. Change the File Name property to frmMain.
3. Set the properties listed in Table 2.2.
   Table 2.2 *Properties of the Initial Form*

Object	Property	Value

This document is created with the unregistered version of CHM2PDF Pilot

Form1	*Name*	frmMain
	Text	The Doughnut Factory
	Size	480, 400 (approximately)
Libby's Doughnut Factory (Visual Basic .NET only)	*Startup Object*	frmMain

4. If using Visual C#, locate static void Main() and change it so that it reads as follows:

Visual C#

This document is created with the unregistered version of CHM2PDF Pilot

Review

The following review questions are intended to reinforce key concepts and information presented in this chapter. If you are unable to answer a question, return to the appropriate lesson and review, then try the lesson again. Answers to the questions can be found in Appendix A.

1. You are creating an application for a major bank. The application should integrate seamlessly with Microsoft Office XP, be easy to learn, and instill a sense of corporate pride in the users. Name two ways you might approach these goals in the user interface.
2. You are writing an application that needs to display a common set of controls on several different forms. What is the fastest way to approach this problem?
3. If you wanted to prompt a user for input every time a form received the focus, what would be the best strategy for implementing this functionality?
4. Describe two ways to set the tab order of controls on your form.
5. What is an extender provider, and what does one do?
6. Explain when you might implement a shortcut menu instead of a main menu.
7. Describe what is meant by field-level validation and form-level validation.

This document is created with the unregistered version of CHM2PDF Pilot

Chapter 3

Types and Members

About This Chapter

This chapter describes how to use different data types, constants, enums, arrays, and collections. Additionally, this chapter explains how to implement properties and events.

Before You Begin

To complete the lessons in this chapter, there are no prerequisites.

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 1: Using Data Types

The Microsoft .NET Framework provides a robust system of primitive types to represent data. Data primitives exist for the representation of integer numbers, floating-point numbers, Boolean values, characters, and strings. The type system enforces type-safety, so that implicit casts only occur when no loss of data is possible. Explicit conversions can be performed in situations that might cause a loss of data. The .NET data types contain built-in functionality that performs a variety of type related conversions and tasks.

After this lesson, you will be able to

- Describe the data types provided by the .NET Framework
- Explain how to perform implicit and explicit conversions between types
- Describe the basic functionality provided by types of the .NET Framework
- Describe string manipulation functions with the *String* class methods

Estimated lesson time: 30 minutes

Data types are used to store and represent data in your application. The .NET Framework provides an extensive system of types that allow you to store, manipulate, and pass values between members of your application. The .NET languages are *strongly typed*. This means that objects of one type cannot be freely exchanged with objects of a different type. All parameters in method and property calls must be of the appropriate type, or the call will fail. Implicit and explicit conversions allow you to convert data types when necessary. Such conversions are possible because all types in the .NET Framework derive from Object, the base type of all classes and structures.

The .NET Data Types

The .NET data types are the types you use to store your data. They are value types and can be broken down into subcategories: Integer types, Floating-point types, the Boolean type, and the Char type. Two built-in reference types that are an integral part of your application will also be discussed: the String type and the Object type.

Integer Types

The .NET Framework provides a variety of Integer types. Table 3.1 summarizes these types and lists their corresponding Microsoft Visual Basic .NET and Microsoft Visual C# types.

Table 3.1 *Integer Types in the .NET Framework*

Type	Visual C# Name	Visual Basic .NET Name	Description	Range
System.Byte	byte	Byte	8-bit unsigned integer	0 to 255
System.Int16	short	Short	16-bit signed integer	-32768 to 32767

This document is created with the unregistered version of CHM2PDF Pilot

System.Int32	int	Integer	32-bit signed integer	-231 to 231-1
System.Int64	long	Long	64-bit signed integer	-263 to 263-1
System.SByte	sbyte	(Not implemented)	8-bit signed integer	-128 to 127
System.UInt16	ushort	(Not implemented)	16-bit unsigned integer	0 to 65535
System.UInt32	uint	(Not implemented)	32-bit unsigned integer	0 to 232-1
System.UInt64	ulong	(Not implemented)	64-bit unsigned integer	0 to 264-1

You can assign values to the Integer types using either decimal or hexadecimal notation. To use hexadecimal notation in an integer literal, it should be prefixed with &H for Visual Basic .NET and 0x for Visual C#. For example:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 2: Constants, Enums, Arrays, and Collections

In this lesson, you will learn how to use the user-defined Constant and Enum types and the Array type as well as learn how to implement advanced array functionality with the *Collection* classes.

After this lesson, you will be able to

- Create constants and enumerations and use them in methods
- Describe how to create and use arrays
- Explain what a collection is and how to use one
- Describe how to enumerate through the members of a collection or an array

Estimated lesson time: 40 minutes

When working with objects, it is frequently necessary to organize groups of objects into some kind of structure. Groups of similar or dissimilar objects might need to be accessed in sequence and require syntax to keep track of their organization. Arrays allow you to organize groups of similar objects and refer to them by an index rather than a name. Collections are similar to arrays but implement advanced functionality and can be used to organize dissimilar groups of objects as well. Constants allow you to define user friendly names to represent frequently used values. Enumerations (enums) allow you to organize sets of related integral constants and use them to help make your application easy to read and debug.

Constants and Enumerations

You might find that certain values are frequently used in your application. Constants allow you to assign user friendly names to frequently used values and refer to them in code. Enumerations are user-defined sets of integral constants that correspond to a set of friendly names. Using constants and enumerations will make your code easier to read, easier to debug, and less prone to errors caused by typographical errors.

Using Constants

Constants allow you to refer to frequently used values by friendly names. You use the *Const* (*const*) keyword to define a constant. For example:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 3: Implementing Properties

Properties are members of classes that expose member variables or objects. Properties have similarities to both fields and methods. Values are set and retrieved using the same syntax as fields: getting a value from a property or setting a value with a property actually calls a specialized method that carries out these functions. Properties can contain code that validates values before setting them or carries out any other function that is required by the application.

After this lesson, you will be able to

- Explain how to implement properties
- Describe how to create a *read-only* or *write-only* property
- Explain what a default property or indexer is and how to create one
- Explain how to expose a collection of objects as a property

Estimated lesson time: 30 minutes

Implementing Properties

Properties allow you to expose member values and objects in a more robust way than simply using fields. A property is essentially a specialized method that looks like a field. Property values are set and retrieved in the same manner as fields, which is shown in the following example:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 4: Implementing Delegates and Events

Events are messages that indicate an interesting occurrence in another part of the application. When an event is raised, other parts of your application are given the opportunity to respond to that event by executing methods called *event handlers*. In Chapter 2, you learned how to use Visual Studio .NET to write handlers for events raised by forms and controls. In this lesson, you will learn how to declare and raise your own events from your application, write event handlers for those events, and dynamically add and remove event handlers at run time.

After this lesson, you will be able to

- Explain what a delegate is and how to create one
- Describe how to declare and raise events from your application
- Explain how to implement event handlers and associate them with events
- Describe how to dynamically add and remove event handlers at run time

Estimated lesson time: 30 minutes

Events are members of your class. An event represents a message that is sent to other parts of the application. When something noteworthy in the application occurs, your application can *raise* the event, which sends out the message. The event can wrap any arguments that contain information about the event and send them along with the event. Other parts of the application can *handle* the event, which means that a method is executed in response to the event. Any method that handles an event must have the same signature as the event itself; that is, it must take the same kinds of arguments as the event passes. An event can be handled by more than one method, and a single method can handle more than one event.

Delegates

Central to the way in which events work are special classes called delegates. A delegate is essentially a type-safe function pointer. It allows you to pass a reference to the entry point for a method and invoke that method without making an explicit method call. When you declare a delegate, you specify the signature of the method that it can call and the return type. The following example demonstrates how to declare a delegate:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lab 3: Adding Components and Implementing Members

In this lab, you will create a *DoughnutMachine* class that represents a doughnut making machine for your virtual doughnut factory. This class will create instances of a *Doughnut* class on a regular schedule and update the display form through events. The solution to this lab is available in the \Labs\Ch03\Solution folder on the Supplemental Course Materials CD-ROM.

Before You Begin

Before you begin this lab, you must have either completed the Chapter 2 lab or loaded the Chapter 2 solution from the CD-ROM.

Estimated lab time: 45 minutes

Exercise 3.1: Creating the DoughnutMachine Component

In this exercise, you will create a component that represents a doughnut making machine. This class will create and manage instances of a *Doughnut* class and expose a collection of doughnuts. In addition, it will implement member events that inform the containing component when a doughnut is ready.

To create the *DoughnutMachine* component

1. From the Project menu, choose Add Component. The Add New Item window appears. Name your component **DoughnutMachine.vb** or **.cs** as appropriate, and click OK.

   The component designer opens.
2. In the Toolbox, under the Windows Forms tab, double-click Timer. A *Timer* component is added to the designer.
3. In Solution Explorer, right-click DoughnutMachine, and choose View Code.

This document is created with the unregistered version of CHM2PDF Pilot

4. Add an enum to your component that represents the different types of doughnuts. Your enum should look like this:

   Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Review

The following review questions are intended to reinforce key concepts and information presented in this chapter. If you are unable to answer a question, return to the appropriate lesson and review, then try the lesson again. Answers to the questions can be found in Appendix A.

1. Explain when a type conversion will undergo an implicit cast and when you must perform an explicit cast. What are the dangers associated with explicit casts?
2. Explain why you might use enums and constants instead of their associated literal values.
3. Briefly summarize the similarities and differences between arrays and collections.
4. Explain how properties differ from fields. Why would you expose public data through properties instead of fields?
5. Explain what a delegate is and how one works.

This document is created with the unregistered version of CHM2PDF Pilot

Chapter 4

Object-Oriented Programming and Polymorphism

About This Chapter

In this chapter, you will learn the fundamentals of object-oriented programming. You will learn how to create overloaded methods and how to implement polymorphic behavior with interfaces, and through inheritance.

Before You Begin

To complete the lessons in this chapter, you must have completed Chapters 2-3.

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 1: Introduction to Object-Oriented Programming

Programming in the .NET Framework environment is done with objects. Objects are programmatic constructs that represent packages of related data and functionality. Objects are self-contained and expose specific functionality to the rest of the application environment without detailing the inner workings of the object itself. Objects are created from a template called a class. The .NET base class library provides a set of classes from which you can create objects in your applications. You can also use the Visual Studio programming environment to create your own classes to use in your programs. This lesson will introduce you to the concepts associated with object-oriented programming.

After this lesson, you will be able to

- Describe the members of an object
- Describe the difference between an object and a class
- Explain what is meant by "object model"
- Explain what is meant by "abstraction," "encapsulation," and "polymorphism"

Estimated lesson time: 20 minutes

Objects, Members, and Abstraction

An object is a programmatic construct that represents something. In the real world, we think of objects like cars, bicycles, laptop computers, and so on. Each of these items exposes specific functionality and has specific properties. In your application, an object might be a form, a control such as a button, a database connection, or any of a number of other constructs. Each object is a complete functional unit, and contains all of the data and exposes all of the functionality required to fulfill its purpose. The ability of programmatic objects to represent real-world objects is called *abstraction*.

Classes Are Templates for Objects

Classes were discussed in Chapter 1 and represent user-defined reference types. Classes can be thought of as blueprints for objects. They define all of the members of an object, define the behavior of an object, and set initial values for data when appropriate. When a class is instantiated, an in-memory instance of that class is created. This instance is called an object. To review, you instantiate a class using the *New* (*new*) keyword as follows:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 2: Overloading Members

Overloading allows you to create multiple members with the same name. Each member that shares a name must have a different signature. Overloading is most commonly used in methods, but C# allows you to overload operators as well. In this lesson, you will learn how to create overloaded members.

After this lesson, you will be able to

- Explain how to create an overloaded method
- Describe how to create an overloaded operator in C#

Estimated lesson time: 15 minutes

You might want to create a member that can accept different sets of parameters. Take the following method as an example:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 3: Interface Polymorphism

Interfaces allow you to define contracts for behavior. Different and diverse classes can implement the same interfaces and can thus interact with other objects in a polymorphic manner. In this lesson, you will learn how to define an interface, how to implement an interface, and how objects can interact through an interface.

After this lesson, you will be able to

- Describe how to define an interface
- Describe how to implement an interface
- Explain how interfaces enable polymorphism

Estimated lesson time: 30 minutes

An interface is a contract. Any object that implements a given interface guarantees to provide an implementation of the members defined in that interface. If an object requires interaction with a specific interface, any object that implements that interface can supply the requisite interaction.

An interface defines only the members that will be made available by an implementing object. The definition of the interface states nothing about the implementation of the members, only the parameters they take and the types of values they will return. The implementation of an interface is left entirely to the implementing class.

It is therefore possible for different objects to provide dramatically different implementations of the same members. Take, for example, an interface named *IShape*, which defines a single method *CalculateArea*. A *Circle* class that implemented this interface would calculate its area differently than a *Square* class that implemented the same interface. However, an object that needed to interact with an *IShape* could call the *CalculateArea* method in either a *Circle* or a *Square* and obtain a valid result.

Defining Interfaces

Interfaces are defined with the *Interface* (*interface*) keyword. For example:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 4: Inheritance Polymorphism

Inheritance allows you to declare a new class that retains all of the members and functionality of a previously defined class. This enables you to create classes that implement basic, common functionality, and then create specialized subclasses that serve different but related functions. In this lesson, you will learn how to use inheritance to derive new classes from base classes, how to create new implementations of base class members, and how to create abstract base classes.

After this lesson, you will be able to

- Describe how to extend a base class
- Explain how to override and shadow base class members
- Describe an abstract class
- Explain how to create abstract members

Estimated lesson time: 60 minutes

Inheritance

Inheritance allows you to create several classes that are distinct but share a common functionality. Specialized classes, called *derived classes*, *inherit* from a common class, called the *base class*. A derived class is also said to *extend* a base class. The base class encapsulates the common functionality that will be present in each derived class, and the derived classes provide additional functionality that is specific to each class. For example, let's consider a class called *Truck*. The *Truck* class provides all of the basic functionality that a car might need: it has the methods to make it go forward, backwards, and turn, and it encapsulates all of the necessary objects to do so. You might create a derived class called *PickupTruck* that inherits *Truck*. Because it inherits *Truck*, it has all of the basic functionality that *Truck* provides. You could then choose to implement additional functionality specific to the class, such as a *Cargo* property. *PickupTruck* could then serve as a base class for an additional derived class *FourWheelDrivePickupTruck*, for example.

Polymorphic Behavior in Inherited Classes

Inherited classes generally have an is a relationship with the base class. A *FourWheelDrivePickupTruck* is a *PickupTruck*, and a *PickupTruck* is a *Truck*. Any instance of a derived class can polymorphically behave as an instance of its base class. Thus, if a method requires a *Truck* object, you can supply a *PickupTruck* object. Any derived class may be implicitly converted to its base class. When cast to its base class, any members implemented by the derived class will be inaccessible only the members of the base class will be available.

Creating Inherited Classes

This document is created with the unregistered version of CHM2PDF Pilot

You can create an inherited class by using the *Inherits* keyword in Visual Basic .NET or the colon (:) in C#. The following example declares a *PickupTruck* class that inherits the *Truck* class:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lab 4: Using Inherited Classes

In this lab, you will create and use inherited classes in your Doughnut Factory application. You will create a strongly typed collection class by extending the collection base and incorporating it into your application. Then, you will extend the *Doughnut* class to create the *CakeDoughnut* and *FilledDoughnut* classes, and modify your application to use these new classes. The solution to this lab is available in the \Labs\Ch04\Solution folder on the Supplemental Course Materials CD-ROM.

Before You Begin

Before you begin this lab, you must have either completed the Chapter 3 lab or loaded the Chapter 3 solution from the CD-ROM.

Estimated lab time: 60 minutes

Exercise 4.1: Creating a Strongly Typed Collection Class by Extending CollectionBase

In this exercise, you will create a collection class that accepts and returns only instances of the *Doughnut* class. You will create a class that inherits from the .NET Framework class *CollectionBase*. You will then implement the *Add* and *Remove* methods and the *Item* property. In your implementation, you will provide code to ensure that only *Doughnuts* are added to your collection, and that all of the objects returned from the collection are typed as *Doughnuts*. In addition, you will modify your application to use the new collection class.

To create a strongly typed collection class

1. From the Project menu, choose Add Class. The Add New Item window appears. Name your class **DoughnutCollection.vb** or **.cs** as appropriate, and click OK.

   The code editor for your new class opens.
2. Designate this new class to inherit from *System.Collections.CollectionBase* as follows:

   Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

This document is created with the unregistered version of CHM2PDF Pilot

Review

The following review questions are intended to reinforce key concepts and information presented in this chapter. If you are unable to answer a question, return to the appropriate lesson and review, then try the lesson again. Answers to the questions can be found in Appendix A.

1. Briefly explain encapsulation and why it is important in object-oriented programming.
2. What is method overloading, and when is it useful?
3. You need to create several unrelated classes that each exposes a common set of methods. Briefly outline a strategy that will allow these classes to polymorphically expose that functionality to other classes.
4. You need to create several classes that provide a core set of functionality but each must be able to interact with a different set of objects. Outline a strategy for developing these classes with the least development time.
5. Describe an abstract class and explain when one might be useful.

This document is created with the unregistered version of CHM2PDF Pilot

Chapter 5

Testing and Debugging Your Application

About This Chapter

In this chapter, you will learn to use the debugging tools. You will learn about the *Trace* and *Debug* classes, and how they can be used to diagnose problems with your program's execution. You will learn how to create an effective unit test plan. In addition, throwing and handling exceptions will be covered.

Before You Begin

To complete the lessons in this chapter, there are no prerequisites.

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 1: Using the Debugging Tools

Programming errors are inevitable. Even very experienced programmers occasionally introduce errors while writing code. These errors are referred to as bugs. Debugging is the process of locating and fixing these errors. In this lesson, you will learn about the kinds of errors you are likely to encounter and the tools available to isolate and correct these errors.

After this lesson, you will be able to

- Describe the common types of errors
- Explain what Break mode is and how to set breakpoints
- Describe how to step through code
- Describe some of the debugging windows that are available and explain how to use them

Estimated lesson time: 45 minutes

Types of Errors

There are three basic types of errors you might encounter when writing code. Syntax errors represent code that cannot be understood by the compiler. Run-time errors are errors that attempt an operation that is impossible to carry out. Additionally, logical errors are errors that compile and execute correctly but return an unexpected result. In this section, we will examine each type in turn.

Syntax Errors

A syntax error occurs when the compiler cannot compile the code it is given. For example, a syntax error occurs when keywords are typed incorrectly, punctuation is omitted, or an incorrect construct is parsed. The following example demonstrates two syntax errors:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 2: Using the Debug and Trace Classes

The *Debug* and *Trace* classes allow you to produce and log informative messages about the conditions of your application without having to interrupt application execution. In this lesson, you will learn how to use the *Debug* and *Trace* classes, how to add Trace Listeners and Trace switches, and how to configure Trace switches in a program that has been compiled and released.

After this lesson, you will be able to

- Explain how to display error messages using the *Debug* and *Trace* classes
- Describe how to create Trace Listeners and log trace output
- Describe how to create and use Trace switches
- Explain how to configure Trace switches in the application .config file

Estimated lesson time: 45 minutes

When debugging simple applications, you frequently do not need to perform any debugging tasks more complex than examining your code line-by-line as described in the previous lesson. Complex applications are another matter, however. Large programs can easily have hundreds of thousands of lines of code. When an erroneous result occurs, it might not be possible to track down logical errors by stepping through code and examining application variables. The *Debug* class provides a way for you to create and log informative messages about program conditions while your application is executing. The *Trace* class allows you to create diagnostic instrumentation for your application that can deliver informative messages about program conditions even after your application has been compiled and released.

How Tracing Works

The *Trace* and *Debug* classes are classes that contain *Shared* (*static*) methods that allow you to test conditions at run time and log the results. The output from these methods is displayed in the Output window and can be viewed while debugging. The output from these methods is also sent to the *Listeners* collection. The *Listeners* collection contains a group of classes that can receive output from the *Trace* and *Debug* classes. *Trace* and *Debug* share the same *Listeners* collection, and all the listeners in the collection receive the same input from *Trace* and *Debug*. There are different kinds of Listeners, such as Listeners that write to text files and Trace Listeners that write to event logs. After program execution is complete, you can examine the Listener traces to identify errors in your application. Tracing can also be useful in program optimization, which is discussed further in Chapter 9.

The *Trace* and *Debug* classes are identical. They expose the same methods and properties, and write output to the same set of *Listeners*. The only difference between *Trace* and *Debug* is that *Debug* statements are not included by default when the program is compiled into a release build, whereas *Trace* statements are. Thus, the *Debug* class is used principally for debugging in the development phase, whereas *Trace* can be used for testing and optimization

This document is created with the unregistered version of CHM2PDF Pilot

after an application has been compiled and released.

Writing Trace and Debug Output

You can write output to the *Listeners* collection by using the methods exposed by the *Trace* and *Debug* classes. These classes expose six methods for writing output:

- ***Write*.** Writes text to the *Listeners* collection unconditionally.
- ***WriteLine*.** Writes text to the *Listeners* collection and follows the text with a carriage return.
- ***WriteIf*.** Writes text to the *Listeners* collection if the specified Boolean expression is true.
- ***WriteLineIf*.** Writes text and a carriage return to the *Listeners* collection if the specified Boolean expression is true.
- ***Assert*.** Writes an assertion message to the *Listeners* collection if the specified Boolean expression is false. It also causes a message box with the assertion message to be displayed.
- ***Fail*.** Creates an assertion that automatically fails without testing a condition. An assertion message is written to the *Listeners* collection, and a message box is displayed with the message.

The following example demonstrates how to invoke these methods in code:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 3: Creating a Unit Test Plan

Once you have completed the code for your application, you must test it to ensure that it functions according to specifications. Although syntax errors are detected by the compiler, run-time errors and logical errors might not be revealed without thorough testing of your program. In this lesson, you will learn how to design an efficient test plan for your component or application. Additionally, you will learn how to create a multicultural test environment and to provide multi-cultural test data to your application.

After this lesson, you will be able to

- Describe how to create a test plan for your classes
- Explain how to create a multicultural test environment for your application
- Describe how to provide multicultural test data to your application

Estimated lesson time: 30 minutes

The Unit Test Plan

Most code contains errors when first written. Errors are inevitable, and even very experienced programmers still make mistakes from time to time. Thus, you should expect your code to have bugs when first written. However, a final application that is full of bugs or does not function according to spec is useless. Thus, you must strive to deliver bug-free final content. You create a bug-free final version from a bug-laden first draft through testing.

Testing and debugging are separate but related activities. Debugging refers to the actual finding and correcting of code errors, whereas testing is the process by which errors are found. Testing is usually broken down by method. The individual methods are tested with a variety of inputs and execution parameters. This approach is called *unit testing* because it tests the units of your application. Most applications are too complex to allow every possible variation of input and run conditions to be tested. Indeed, consider the following method:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 4: Handling and Throwing Exceptions

After you have completed testing and debugging your application, run-time errors might still be inevitable. For example, an application that saves data to a floppy disk will encounter an error if it attempts to save to a disk that is not ready. Even though the application code might execute as designed, an error will still result. Visual Basic .NET and Visual C# provide structured exception handling as a means of writing code that recovers gracefully from errors and allows program execution to continue. You can also create and throw exceptions from your components that communicate to the main program that an error has occurred.

After this lesson, you will be able to

- Describe how exceptions are handled by the common language run time
- Create a structured exception handler
- Throw exceptions from your code

Estimated lesson time: 30 minutes

How Exceptions Are Handled

When an application encounters a run-time error, it raises an *exception*. Exceptions are instances of specialized classes that all derive from the base class *System.Exception*. Exceptions include features that facilitate diagnosis and management of the error, such as:

- The *Message* property, which can contain a human-readable description of the error as well as any other relevant information about the error.
- The *StackTrace* property, which contains a stack trace to allow tracing and pinpointing of where the error occurred in the code.

In addition to system-defined exceptions, you can also create custom exceptions by extending *System.ApplicationException*. This is discussed in greater detail later in this lesson.

When a run-time error is encountered, an instance of the exception corresponding to the error is created and passed from the method that generated the error up the call stack to the method that called it. If an exception handling structure is encountered in that method, it is handled there. If no exception handling structure is found, the exception is passed further up the call stack to the next method, and so on. If no exception handling structure is found anywhere in the call stack, the default exception handler is engaged. This causes a message box to display the type of exception and any additional information provided by it, and then the application is terminated without any opportunity to save work or attempt to recover from the error.

With structured exception handling, you can provide a means for your application to recover from unexpected errors or at least have the chance to save data before closing.

This document is created with the unregistered version of CHM2PDF Pilot

Creating an Exception Handler

Exception handlers are implemented for specific methods. Thus, each method you create can have its own exception handler, specifically tailored to the method and exceptions it is likely to throw. Methods that are likely to throw exceptions, such as methods that involve file access, should implement exception handling. Exception handlers use a *Try...Catch...Finally* syntax. Three general steps are involved in creating an exception handler:

1. Wrap the code the handler will be associated with in a *Try* (*try*) block.
2. Add one or more *Catch* (*catch*) blocks to handle the possible exceptions.
3. Add any code that must always be executed to a *Finally* (*finally*) block. Such code will execute regardless of whether or not an exception is thrown.

The first step in creating an exception handler is to wrap the code in a *Try* (*try*) block. The *Try* (*try*) block forms the exception handler; if an exception is thrown by any statement in the code or any unhandled exception is received through call stack bubbling, the method will attempt to handle it within this exception handler. If the exception cannot be handled, it will be bubbled further up the call stack. The following example demonstrates the *Try* (*try*) block:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lab 5: Debugging an Application

In this lab, you will open a small application that contains syntax, run-time, and logical errors. You will correct syntax errors, design a test plan to identify and correct run-time and logical errors in two methods, and use the Visual Studio debugging tools to monitor program execution. The solution to this lab is available in the \Labs\Ch05\Solution folder on the Supplemental Course Materials CD-ROM.

Before You Begin

There are no prerequisites to starting this lab.

Estimated lab time: 30 minutes

Exercise 5.1: Identifying Syntax Errors

In this exercise, you will use Visual Studio .NET to identify and correct syntax errors in the sample project. To open the Lab 5 project

1. Start Visual Studio.
2. From the File menu, choose Open, then Project.

   The Open Project dialog box appears.
3. Browse to the \Labs\Ch05\Partial folder and choose either CalculatorVB or CalculatorCSharp, as appropriate. Click Open to open the solution.

To identify and correct syntax errors

1. From the Build menu, click Build Solution.

   Syntax errors appear in the Task List window, as shown in Figures 5.3 and 5.4.

This document is created with the unregistered version of CHM2PDF Pilot

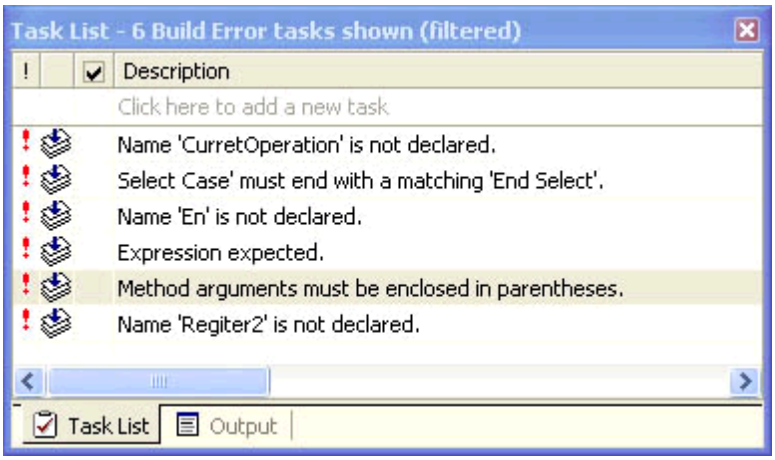


Figure 5.3 *The Task List window in Visual Basic .NET*

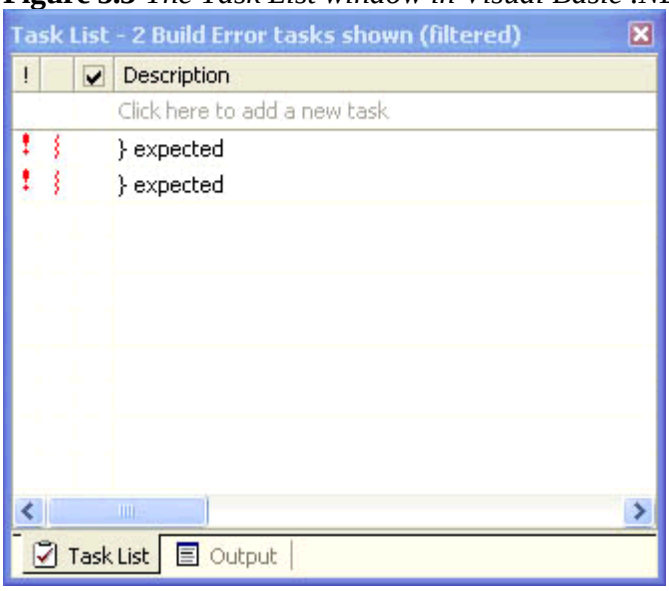


Figure 5.4 *The Task List window in Visual C#*

2. For Visual Basic .NET, double-click the error that reads "Select Case' must end with a matching End Select". You are taken to the end of the Select Case block. Examine the code and correct the error by correcting the syntax in the End Select line. Note that only two errors are displayed in the Task List window now.

   For Visual C#, double-click the first error that reads "} expected". You are taken to the end of the operation handler. Carefully examine each pair of curly braces to determine which one is missing its partner. Correct the error by adding the appropriate curly brace to the end of the *switch* statement.
3. For Visual C# only: From the Build menu, choose Build Solution to reassess your code for errors. Note that two additional errors are found.
4. Double-click the first error in the Task List. You are taken to an error that cannot find a member named CurrentOperation. Change this statement to read CurrentOperation to correct the error.
5. Double-click the final error in the Task List. Correct the spelling of Register2 to correct the error.
6. From the Build menu, choose Build Solution. Note that no further errors are discovered.

To identify and correct a run-time error

1. Locate the *NumberHandler* method. This is a fairly simple method with minimal data path branching. This method builds a string of numbers as the buttons 1 through 9 are pressed on the calculator face.
2. Develop a test plan for this method. Your test plan should explore all possible data branches and test the method with a variety of input and application variable values. Determine in advance the expected results of each test case. Table 5.4 demonstrates some of the test cases you might use.

   Table 5.4 *Test Cases for the NumberHandler Method*

This document is created with the unregistered version of CHM2PDF Pilot

Button Pressed	Value of *LabelDisplay.Text*	Value of *CurrentNumberString*	Value of *CurrentNumberString expected*
1	"0"	"43234422342"	""
4	"44"	"44"	"444"
9	"1266345578456"	"1266345578456"	"12663455784569"
9	"-3446.222231"	"-3446.222231"	"-3446.2222319"
3	"6"	"6"	"63"

3. Execute your tests. Use the Locals, Autos, and Watch windows to observe the values of variables as the program is executing. Use the Immediate window to assign values to variables. You can use Set Next Statement to rerun code multiple times.
4. Log your results and correct any run-time errors you might find.

   Logical errors are the most difficult types of errors to identify. It frequently takes repeated testing to observe a problem and additional testing to identify the source of the error. In the next section, you will identify a logical error in another method of this application.

To identify and correct a logical error

1. Locate the method named *EqualsHandler*. This method handles operations for the calculator when the = button is pressed. The number contained by the variable Register1 is added to, subtracted from, multiplied by, or divided into the number contained by Register2, and the value of that operation is assigned to Result. The user interface is then updated.
2. Execute the test plan shown in Table 5.5. Although not a complete plan by any measure, it does test every logical path through the method and should be sufficient to find a logical error. Note that boundary analysis and more in-depth testing has been omitted for the purpose of brevity.

   Table 5.5 *Test Cases for the EqualsHandler Method*

Current Operation	Register1	Register2	Expected Value of Result
Operation.Plus	442	5789	6231
Operation.Minus	400000	222222	177778
Operation.Multiply	23	12	276
Operation.Divide	3345	34	98.3823529411765

3. Diagnose and resolve any logical errors you encounter.

This document is created with the unregistered version of CHM2PDF Pilot

Review

The following review questions are intended to reinforce key concepts and information presented in this chapter. If you are unable to answer a question, return to the appropriate lesson and review, then try the lesson again. Answers to the questions can be found in Appendix A.

1. Describe Break mode and some of the available methods for navigating in Break mode.
2. When would you use the Watch window?
3. You are deploying a beta version of a large application and want to collect performance data in text files while the application is in use. Briefly describe a strategy for enabling this scenario.
4. When testing a method, should you test data that is known to be outside of the bounds of normal operation? Why or why not?
5. Briefly explain what each segment of a *Try...Catch...Finally* (*try...catch...finally*) block does.

This document is created with the unregistered version of CHM2PDF Pilot

Chapter 6

Data Access Using ADO.NET

About This Chapter

This chapter describes how to access data with Microsoft ADO.NET. You will learn how to create a data access architecture for your application, connect to Microsoft SQL Server and other databases, bind data to the user interface, and handle data errors.

Before You Begin

To complete the lessons in this chapter, you must have completed Chapters 1-5.

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 1: Overview of ADO.NET

Most applications require some kind of data access. Desktop applications need to integrate with central databases, Extensible Markup Language (XML) data stores, or local desktop databases. ADO.NET data access technology allows simple, powerful data access while maximizing system resource usage.

After this lesson, you will be able to

- Describe the major components of ADO.NET data access
- Explain the role of each data access component
- Describe in general terms how ADO.NET data access is facilitated

Estimated lesson time: 30 minutes

Different applications have different requirements for data access. Whether your application simply displays the contents of a table or processes and updates data to a central SQL server, ADO.NET provides the tools to implement data access easily and efficiently.

Disconnected Database Access

Previous versions of data access technologies have provided continuously connected data access by default. In such a model, an application creates a connection to a database and keeps it open for the life of the application or at least for the amount of time that data is required. As applications become more complex and databases begin to serve more and more clients, a connected data access technology becomes impractical for a variety of reasons. For example:

- Open database connections are expensive in terms of system resources. The more open connections, the less efficient system performance becomes.
- Applications with connected data access are very difficult to scale up. An application that can maintain connections with two clients might do poorly with 10, and might be completely unusable with 100.

ADO.NET addresses these issues by implementing a disconnected database access model by default. In this model, data connections are established and only left open long enough to perform the requisite action. For example, if data is being requested from a database by an application, the connection is opened just long enough to load the data into the application, and then it is closed. Likewise, if a database is being updated, the connection is opened to execute the UPDATE command, and then closed again. By keeping connections open only for the minimum required time, ADO.NET conserves system resources and allows data access to be scaled up with a minimal impact on performance.

ADO.NET Data Architecture

This document is created with the unregistered version of CHM2PDF Pilot

Data access in ADO.NET relies on two components: the *DataSet*, which stores data on the local machine, and the *Data Provider*, which mediates interaction between the program and the database.

The DataSet

The *DataSet* is a disconnected, in-memory representation of data. It can be thought of as a local copy of the relevant portions of the database. Data can be loaded into a *DataSet* from any valid data source, such as a Microsoft SQL Server database, a Microsoft Access database, or an XML file. The *DataSet* is persisted in memory, and the data therein can be manipulated and updated independent of the database. When appropriate, the *DataSet* can then act as a template for updating the central database.

The *DataSet* object contains a collection of zero or more *DataTable* objects, each of which is an in-memory representation of a single table. The structure of a particular *DataTable* is defined by the DataColumns collection, which enumerates the columns in a particular table, and the Constraint Collection, which enumerates the constraints on the table. Together, these two collections make up the *schema* of the table. A *DataTable* also contains a DataRows collection, which contains the actual data in the *DataSet*.

The *DataSet* contains a DataRelations collection. A *DataRelation* object allows you to create associations between rows in one table and rows in another table. The DataRelations collection enumerates a set of *DataRelation* objects that defines these relationships between tables in the *DataSet*. For example, consider a *DataSet* that contains two related tables: a Customers table and an Orders table. In the Customers table, each customer is represented only once and is identified by a CustomerID field that is unique. In the Orders table, the customer who placed the order is identified by the CustomerID field, but can appear more than once if a customer has placed multiple orders. This is an example of a *one-to-many* relationship, and you would use a *DataRelation* object to define that relationship.

Additionally, a *DataSet* contains an ExtendedProperties collection, which is used to store custom information about the *DataSet*.

The Data Provider

The link to the database is created and maintained by the *Data Provider*. A Data Provider is not actually a single component, but a set of related components that work together to provide data in an efficient, performance-driven manner. The Microsoft .NET Framework currently ships with two data providers: the *SQL Server .NET Data Provider*, which is designed specifically to work with Microsoft SQL Server 7.0 or later, and the *OleDb .NET Data Provider*, which connects with other types of databases. Each Data Provider consists of similar versions of the following generic component classes:

- The *Connection* object provides the connection to the database.
- The *Command* object is used to execute a command against a data source. It can execute either non-query commands, such as INSERT, UPDATE, or DELETE, or return a *DataReader* with the results of a SELECT command.
- The *DataReader* object provides a forward-only, read-only, connected recordset.
- The *DataAdapter* object populates a disconnected *DataSet* or *DataTable* with data and performs updates.

Throughout this chapter, when information is applicable to member classes of either Data Provider, they are referred to by their generic name. For example, "Command" might mean *OleDbCommand* or *SqlCommand*.

Data access in ADO.NET is facilitated as follows: A *Connection* object establishes a connection between the application and the database. This connection can be accessed directly by a *Command* object or by a *DataAdapter* object. The *Command* object provides direct execution of a command to the database. If the command returns more than a single value, the *Command* object returns a *DataReader* to provide the data. This data can be directly processed by application logic. Alternatively, you can use the *DataAdapter* to fill a *DataSet* object. Updates to the database can be achieved either through the *Command* object or through the *DataAdapter*.

This document is created with the unregistered version of CHM2PDF Pilot

The generic classes that make up the Data Providers are summarized in the following sections.

The Connection Object

The *Connection* object represents the actual connection to the database. Microsoft Visual Studio .NET provides two types of *Connection* classes: the *SqlConnection* object, which is designed specifically to connect to Microsoft SQL Server 7.0 or later, and the *OleDbConnection* object, which can provide connections to a wide range of database types. The *Connection* object contains all of the information required to open a channel to the database in the *ConnectionString* property. The *Connection* object also provides methods that facilitate data transactions.

The Command Object

The *Command* object is also represented by two corresponding classes: *SqlCommand* and *OleDbCommand*. You can use the *Command* objects to execute commands to a database across a data connection. The *Command* objects can be used to execute stored procedures on the database, execute SQL commands, or return complete tables directly. *Command* objects provide three methods that are used to execute commands on the database:

- ***ExecuteNonQuery***. Executes commands that have no return values, such as INSERT, UPDATE, or DELETE.
- ***ExecuteScalar***. Returns a single value from a database query.
- ***ExecuteReader***. Returns a result set by way of a *DataReader* object.

The DataReader Object

The *DataReader* object provides a forward-only, read-only, connected stream recordset from a database. Unlike other components of the Data Provider, *DataReader* objects cannot be directly instantiated. Rather, the *DataReader* is returned as the result of a *Command* object's *ExecuteReader* method. The *SqlCommand.ExecuteReader* method returns a *SqlDataReader* object, and the *OleDbCommand.ExecuteReader* method returns an *OleDbDataReader* object. The *DataReader* can provide rows of data directly to application logic when you do not need to keep the data cached in memory. Because only one row is in memory at a time, the *DataReader* provides the lowest overhead in terms of system performance, but requires the exclusive use of an open *Connection* object for the lifetime of the *DataReader*.

The DataAdapter Object

The *DataAdapter* is the class at the core of ADO.NET's disconnected data access. It is essentially the middleman, facilitating all communication between the database and a *DataSet*. The *DataAdapter* fills a *DataTable* or *DataSet* with data from the database when the *Fill* method is called. After the memory-resident data has been manipulated, the *DataAdapter* can transmit changes to the database by calling the *Update* method. The *DataAdapter* provides four properties that represent database commands:

- ***SelectCommand***. Contains the command text or object that selects the data from the database. This command is executed when the *Fill* method is called, and fills a *DataTable* or a *DataSet*.
- ***InsertCommand***. Contains the command text or object that inserts a row into a table.
- ***DeleteCommand***. Contains the command text or object that deletes a row from a table.
- ***UpdateCommand***. Contains the command text or object that updates the values of a database.

When the *Update* method is called, changes in the *DataSet* are copied back to the database, and the appropriate *InsertCommand*, *DeleteCommand*, or *UpdateCommand* is executed.

This document is created with the unregistered version of CHM2PDF Pilot

Lesson Summary

- ADO.NET is a data access technology that is primarily disconnected and designed to provide efficient, scalable data access.
- Data is represented within a *DataSet* object, which is a disconnected, in-memory copy of part or all of a database.
- The Data Provider is a set of classes that provide access to databases. The main components of the Data Providers are
  - *Connection*
  - *Command*
  - *DataReader*
  - *DataAdapter*
- Visual Studio .NET includes two Data Providers:
  - The SQL Data Provider, which contains classes optimized for accessing Microsoft SQL Server 7.0 or later.
  - The OleDb Data Provider, which contains classes that provide access to a broad range of database formats.

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 2: Accessing Data

Visual Studio .NET provides many built-in wizards and designers to help you build your data access architecture rapidly and efficiently at design time. With a minimum of actual coding, you can implement robust data access for your application. The ADO.NET object model is fully available through code, however, to implement customized features or to fine tune your program. In this lesson, you will learn how to connect to a database with ADO.NET and learn how to retrieve data to your application. You will learn to use the visual designers provided by Visual Studio .NET as well as direct code access.

After this lesson, you will be able to

- Establish a connection to a database
- Create and configure a *DataAdapter*
- Create a *Command* object that uses SQL strings
- Create a *Command* object that accesses stored procedures on the database
- Create a typed *DataSet*
- Fill a *DataSet* with a *DataAdapter*
- Use a *DataReader* to programmatically access data

Estimated lesson time: 60 minutes

Connecting to a Database

You can implement a connection to a database in your application in many different ways. The easiest way is to use the graphical tools provided by Visual Studio .NET to create a connection at design time.

Current data connections are managed in the Server Explorer window. This window is normally docked to the left pane of the Integrated Development Environment (IDE) and shares the same space as the Toolbox. If the Toolbox is currently visible, you can view the Server Explorer by clicking the Server Explorer tab located at the bottom of the Toolbox.

An undocked view of the Server Explorer is shown in Figure 6.1.

Data connections that are currently available to Visual Studio .NET are displayed as child nodes of the Data Connections node in Server Explorer. If you want to add a connection that is already listed under the Data Connections node to your project, all you need to do is drag the connection from the Server Explorer window to the designer. A new *Connection* object of the appropriate type (*Sql* or *OleDb*) is created and automatically configured to connect to your database.

This document is created with the unregistered version of CHM2PDF Pilot

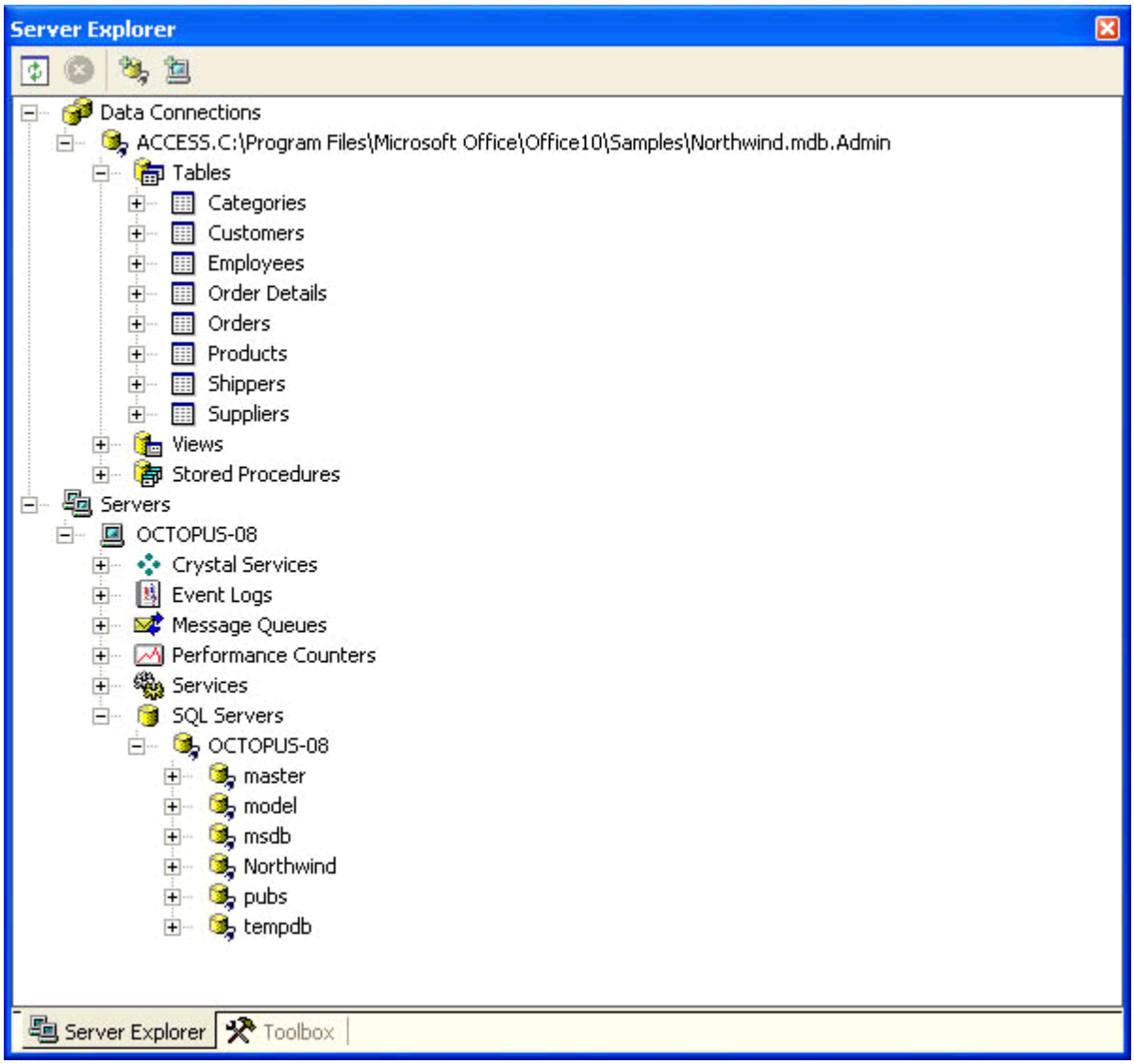


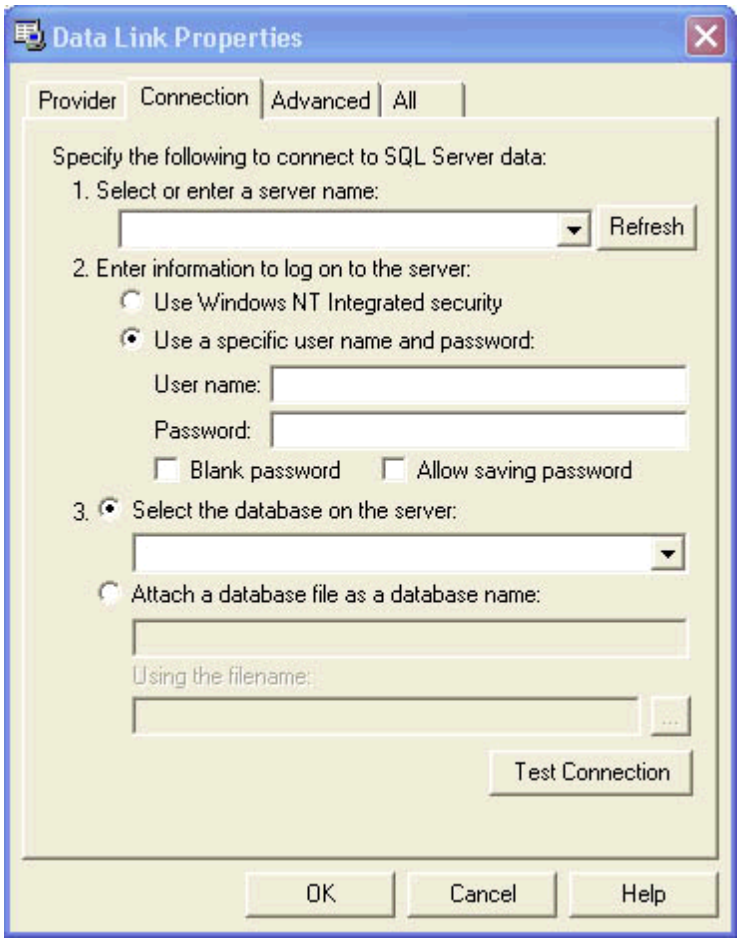
Figure 6.1 *The Server Explorer*

To add a connection from the Server Explorer

Drag the node that represents a database from the Server Explorer window to the designer.

You can also create a new connection in the Server Explorer. You can create a new connection by right-clicking the Data Connections node and choosing Add Connection. This launches the Data Link Properties dialog box, as shown

This document is created with the unregistered version of CHM2PDF Pilot



in Figure 6.2.

Figure 6.2 *The Data Link Properties dialog box*

The Data Link Properties dialog box is a visual way to configure your data connection. The Providers tab allows you to choose the database provider that you will use in your connection. The Connection tab allows you to configure the specific properties of your connections, and the Advanced tab configures properties not normally needed for every connection. The All tab allows you to view and edit the current settings of all the properties. Once you have configured your new connection, you can click OK to close the Data Link Properties window. The new connection appears in the Server Explorer window and can be dragged to your designer.

To create a new connection in the Server Explorer

1. In the Server Explorer, right-click Data Connections and choose Add Connection to open the Data Link Properties window.
2. On the Provider tab, choose the appropriate provider for your data connection.
3. On the Connection tab, choose the database you want to connect to. Depending on the provider, you might also have to set additional properties such as the server name or password settings.
4. When you have entered the appropriate information, click the Test Connection button to verify that the connection is functional.
5. Drag the new connection from the Server Explorer to the designer surface. A new connection object of the appropriate type is created and correctly configured.

You can also create a database connection manually, either by dragging a *Connection* object from the Toolbox to the designer or by declaring and instantiating the object in your code. If you create a connection this way, you will have to manually set the *ConnectionString* property. The following example demonstrates how to create a *Connection* object in code and set the *ConnectionString* property:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 3: Using DataSet Objects and Updating Data

The *DataSet* is the central component in disconnected data access architecture. The ADO.NET data architecture uses *DataSet* objects for the in-memory representation of data. *DataSet* objects can be filled by *DataAdapter* objects, or you can create new *DataSet* objects by reading data from XML or a flat file. You can also create new *DataSet* objects programmatically by adding new rows. The data held in *DataSet* objects can be bound to and displayed in the user interface, updated, and used to update a database through a *DataAdapter*. In this lesson, you will learn how to create *DataSet* objects programmatically, how to add *DataRelation* objects to your *DataSet* objects, and how to update databases from a *DataSet*.

After this lesson, you will be able to

- Describe how to create a *DataSet* programmatically or by reading a flat file
- Explain how to add *Constraints* and *DataRelation* objects to a *DataSet*
- Explain how to update a database through a *DataAdapter*
- Explain how to execute database transactions
- Design a scheme for handling database update errors

Estimated lesson time: 60 minutes

A *DataSet* is an in-memory representation of data. It is, in essence, a copy of part or all of a database, stored locally on the client machine. Accessing the data contained in a *DataSet* does not require interaction with the database, and changes to the data stored in the *DataSet* have no effect on the database until such time as the database is updated.

In the previous lesson, you learned how to fill a *DataSet* from a *DataAdapter*. Although this is the primary way to create *DataSet* objects, you might need to access data that is stored in some other way, such as in an array of values or in a comma-delimited text file. In this section, you will learn alternative ways of creating and filling *DataSet* objects.

Creating and Filling DataSet Objects Without a DataAdapter

You can create *DataSet* objects programmatically just like you would create any other object. For example:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 4: Binding, Viewing, and Filtering Data

Viewing data is a vital part of many applications. Data binding allows you to associate records in a data source with controls on a form, allowing them to be browsed and updated. In this lesson, you will learn how to use ADO.NET to bind data to controls in your application and how to manage currency of records in your application. You will also learn how to use the *DataView* class to create filterable and sortable views of *DataTables*, and how to manage multiple *DataViews* with the *DataViewManager* class.

After this lesson, you will be able to

- Describe the role of a data provider
- Describe the role of a data consumer
- Explain how to bind a property of a data consumer to a data provider
- Describe how currency is managed in Windows Forms
- Explain how to filter and sort data with the *DataView* component
- Describe the role of a *DataViewManager* and how to use one

Estimated lesson time: 45 minutes

Data Binding

Data binding refers to a relationship between a data provider and a data consumer. A data provider is a source for data and is linked to a data consumer, which receives data and processes or displays it. A traditional example of data binding is the relationship between a data-bound control such as a *TextBox* and a data source. The control displays the value of the column in the data source to which it is bound at the current row. As the current row changes, the value displayed by the control changes.

Data Providers

In a data binding relationship, the data provider is the object that provides data to bound properties and controls. In this context, the term data provider refers to any object that data consumers can be bound to, and does not refer to the Data Provider suite of components that facilitate disconnected database access.

In the .NET Framework, any object that implements the IList interface can be a data provider. This not only includes ADO.NET objects, such as *DataSets*, *DataTables*, and *DataColumns*, but also more mundane objects such as arrays or collections. *DataViews*, which are discussed later in this lesson, are a customizable view of data that can also act as a data provider.

Data providers also manage currency of data. In previous data access technologies, a cursor was used to manage data currency. As the cursor moved, the current record changed and bound controls were updated. Because data

This document is created with the unregistered version of CHM2PDF Pilot

access in ADO.NET is fundamentally disconnected, there is no concept of a database cursor. Rather, each data source has an associated *CurrencyManager* object that keeps track of the current record. *CurrencyManager* objects are managed through a form's *BindingContext* object. Data currency is discussed in detail later in this lesson.

Data Consumers

The data consumer represents the control with bound properties. In the .NET Framework, you can bind any run-time accessible property of any control to a data source. For example, you could bind the *Size* property of a control to a database record, or the location, or any other run-time accessible property of the control.

Data binding has a wide variety of uses. A typical scenario that involves data binding is a data entry form. Several controls, such as *TextBox*, *CheckBox*, *ListBox*, and so on are bound to relevant columns of a *DataSet*. New records are manually entered by typing values for each column into the appropriate controls displayed on the form. When a record is complete, it is added to the *DataSet*, which can then be used to update the database.

There are two types of data binding: simple binding and complex binding. A simple-bound control binds one record at a time to a control. For example, a *Label* control can be simply bound to a column in a *DataTable*. In such a case, it would display the member of that column for the current record. Complex binding, on the other hand, allows multiple records to be bound to a single control. Controls such as *ListBox* or *ComboBox* can be complex bound to a single column in a *DataTable* or *DataView*. At run time, these controls would display all members of that column rather than only the current row. Complex-bound controls are usually involved in displaying choices and allowing specific rows of data to be chosen. Controls such as the *DataGrid* are capable of even more complex binding and can be bound to all column and rows of a particular *DataTable* or *DataView*, or even to a *DataSet*.

Creating a Simple-Bound Control

You can create a simple-bound control through the *DataBindings* property. The *DataBindings* property is an instance of the *ControlBindingsCollection* class that keeps track of and organizes which controls are bound to what data sources. At design time, the *DataBindings* property is displayed as a node in the Properties window. This node expands to list the properties that are most commonly data bound, as shown in Figure 6.5.

To bind one of these properties to a data source, click the box next to it in the DataBindings node and open the resulting drop-down menu. A list of available

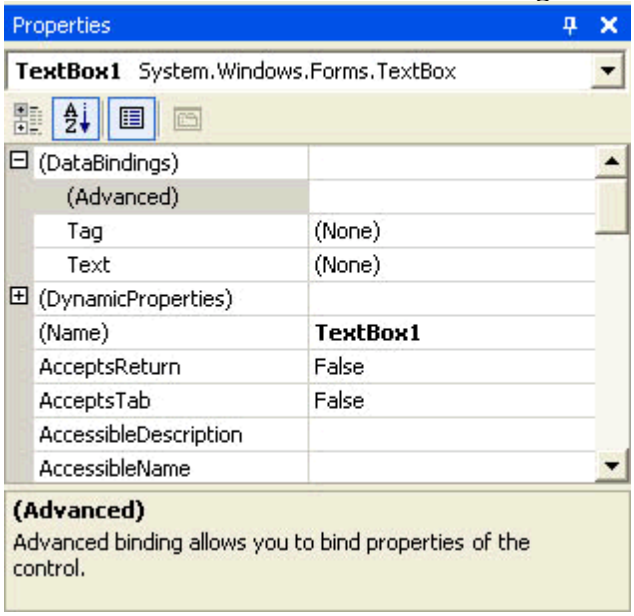


Figure 6.5 *Commonly data-bound properties*

This document is created with the unregistered version of CHM2PDF Pilot

data providers is displayed. Available *DataSet* objects are displayed as expandable nodes that reveal the available *DataTable* objects. Any *DataView* objects will also be listed. Each of these can then be expanded to reveal a list of columns provided by the data source. Browse through the list to the appropriate column and select it. This creates the binding for your control.

You can bind any property of your control to a data source. If you want to bind a property that is not listed beneath the DataBindings node, you can click the (Advanced) box under the DataBindings node and click the ellipses (...) in the box to view the Advanced Data Binding dialog box, as shown in Figure 6.6.

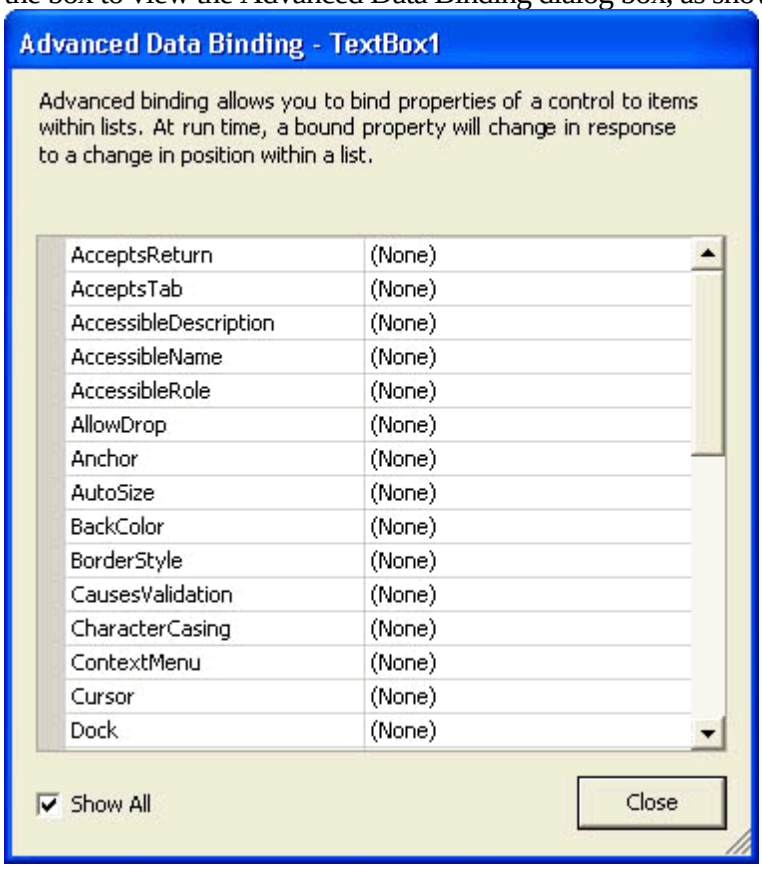


Figure 6.6 *The Advanced Data Binding dialog box*

The Advanced Data Binding dialog box lists all of the run-time available properties of the control and allows you to choose the column to bind to the property.

To bind a property to a data source at design time

1. In the Properties window, expand the DataBindings node.
2. If the property you want to bind to a data source is listed beneath the node, choose the appropriate column of data from the drop-down menu.

   -or-

   If the property you want to bind is not listed, click the ellipses (...) next to the (Advanced) box to open the Advanced Data Binding dialog box. Choose the property you want to bind, and in the adjacent box, choose the column of data to bind it to.

Data Binding at Run Time

You might want to change the data source that a control is bound to at run time. Or, you might not know at design time what data source a particular control might be bound to. In a third scenario, you might want to bind your control to an array or collection that is not instantiated until run time, in which case, you must set the binding in code. Data

This document is created with the unregistered version of CHM2PDF Pilot

binding for a control is managed through the control's *DataBindings* property, which is an instance of a *ControlBindingsCollection* object. At run time, you can add, remove, or clear data binding information by setting the appropriate member of the DataBindings collection.

You can bind a property to a data source by using the *DataBindings.Add* method. This creates a new binding association and adds it to the DataBindings collection. The *Add* method takes three parameters: the property name you want to bind, as a String; the data source you want to bind to, as an object; and the data member of the data source you want to bind the property to. The following example demonstrates how to bind the *Text* property of a *TextBox* control to the CustomerID column of a table called Customers in a *DataSet* called *DataSet1*:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 5: Using XML in ADO.NET

XML is the behind-the-scenes foundation of ADO.NET. Data is described in the XML format, and XML representations of *DataSet* objects, *DataTable* objects, and schemas can be written from in-memory representations of data and persisted as XML text files or streams. The *XmlDataDocument* class allows you to work directly with in-memory representations of XML and synchronize them with a *DataSet*. In this lesson, you will learn some techniques for using the XML-based methods provided by ADO.NET.

After this lesson, you will be able to

- Explain how to retrieve an *XmlReader* from a *SqlCommand* object
- Describe how to fill a *DataSet* from an XML text file
- Describe how to write the contents of a *DataSet* to an XML stream
- Explain how to write the schema of a *DataSet* to an XML stream
- Describe how to read an XML schema onto a *DataSet*
- Explain how to create an *XmlDataDocument* and synchronize it with a *DataSet*
- Describe how to execute an XSLT transform on an *XmlDataDocument*

Estimated lesson time: 45 minutes

Retrieving XML from a SQL Server 2000 Database

Microsoft SQL Server 2000 contains built-in support for XML-based data access. Data can be retrieved from the database and read into memory in the XML format. The ADO.NET SQL Data Provider provides built-in support for retrieving XML from the database.

Retrieving an XmlReader with a SqlCommand

The *SqlCommand* class provides a method for retrieving data as XML. The *ExecuteXmlReader* returns an *XmlReader* object that exposes the data returned by the *SqlCommand* as a set of XML rows. The SQL SELECT query executed by the *ExecuteXmlReader* method must contain a *FOR XML* clause. This method is only available with the *SqlCommand* class and can only be used when connecting to Microsoft SQL Server 2000 or later.

The *XmlReader* class is analogous to the *DataReader* class. It provides read-only, forward-only access to the XML returned by the query. Also like the *DataReader*, the *XmlReader* requires the exclusive use of a connection.

The *XmlReader* exposes a *Read* method, similar to the *Read* method of a *DataReader*, which allows you to iterate through the nodes that are returned. Like a *DataReader*, the *Read* method advances the *XmlReader* to the next

This document is created with the unregistered version of CHM2PDF Pilot

node of the XML stream and returns false when the last node is reached. Also like the *DataReader*, you must call the *Read* method before the first node is accessible. The following example demonstrates how to retrieve an *XmlReader* and read the XML stream it returns:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lab 6: Connecting with a Database

In this lab, you will learn to incorporate different kinds of data access into your application. You will learn to read data from a database with a *DataReader*. You will then learn to use a *DataAdapter* to fill a *DataSet* and use that *DataSet* to update your database. Finally, you will learn to use the XML Designer to create a typed *DataSet*, and use the Data Form wizard to create a master-detail data form. The solution to this lab is available in the \Labs\Ch06\Solution folder on the Supplemental Course Materials CD-ROM.

Before You Begin

Before you begin this lab, you must have the Microsoft Access 2000 data engine or a more recent engine installed on your computer.

Estimated lab time: 60 minutes

Exercise 6.1: Adding Data Access and Using the DataReader

In this exercise, you will learn to use a *DataCommand* to return a *DataReader*, which will allow you to read data directly into your application. You will first create a connection to your database in the Server Explorer.

To add a new connection to an application

1. Open Visual Studio .NET. From the File menu, choose New, Project. Choose a Windows Application project in the language of your choice and click OK.
2. In the Server Explorer, right-click Data Connections and choose Add Connection.

   The Data Link Properties dialog box appears.
3. On the Provider tab, choose Microsoft Jet 4.0 OLE DB Provider. This is the data provider for Microsoft Access 2000. Choose Next to proceed to the Connection tab.

This document is created with the unregistered version of CHM2PDF Pilot

4. Under Select or Enter a Database Name, click the ellipses (...) to open the Select Access Database dialog box.
5. Browse to The Doughnut Shop.mdb and select it. This file will be located in the \Labs\Ch06 folder. Click Open to select the database.
6. Click Test Connection to verify that your connection is valid. A message box should appear indicating that the connection succeeded.
   If you are not notified that your connection succeeded, go back to Step 5 and verify that you have selected the correct database.
7. Click OK.

   A new connection is added to the Server Explorer.

To add and configure a *DataCommand*

1. In the Server Explorer, drag the icon representing the database to the form. A new *OleDbConnection* object representing a connection to your database is added to your application.
2. From the Data tab of the Toolbox, drag an instance of *OleDbCommand* to your form. A new *OleDbCommand* named *OleDbCommand1* (*oleDbCommand1*) is added to your application.
3. Set the properties of *OleDbCommand1* (*oleDbCommand1*) as follows in Table 6.7.
   If you are asked whether you want to regenerate parameters, choose Yes.
4. From the Windows Forms tab of the Toolbox, drag a button and a ListBox over to your form. Set the *Button1*.*Text* (*button1*.*Text*) property to Click to Execute DataReader.
5. In the designer, double-click Button1 (button1) to open the code editor to the default event handler for the *Button1*.*Click* (*button1*.*Click*) event. Add the following code to retrieve a *DataReader* and populate the ListBox:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Review

The following questions are intended to reinforce key concepts and information presented in this chapter. If you are unable to answer a question, return to the appropriate lesson and review, then try the question again. Answers to the questions can be found in Appendix A.

1. What are the major components of a Data Provider, and what function does each fulfill?
2. Briefly contrast connected and disconnected data access in ADO.NET.
3. What are the three possible settings for the *CommandType* property of a *SqlCommand* object or an *OleDbCommand* object, and what does each mean?
4. How could you execute DDL commands, such as ALTER or CREATE TABLE, against a database with ADO.NET?
5. Briefly discuss the advantages and disadvantages of using typed *DataSet* objects.
6. How can you manage data currency on a form with several bound controls?
7. Describe how to use a *DataView* to filter or sort data.
8. Briefly describe an *XmlDataDocument* and how it relates to a *DataSet*.

This document is created with the unregistered version of CHM2PDF Pilot

Chapter 7

Creating Controls Using the .NET Framework

About This Chapter

This chapter describes how to create controls in Microsoft's .NET Framework. You will learn how to create a user control, a custom control, and an inherited control. Additionally, you will learn how to use the GDI+ graphics interface and handle common tasks associated with controls.

Before You Begin

To complete the lessons in this chapter, you must have completed Chapters 1-5.

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 1: Using GDI+

Microsoft Windows is a graphical interface. Users interact with applications through windows, which are graphical representations of application data and options. To take advantage of the rich user experience that Windows offers, you should design your applications so that they expose their functionality to the user graphically. Although the .NET Framework provides a wide array of graphical controls and methods that allow you to create fully visual applications, you might want to render your own graphic content in your application or create a custom appearance for a control. To take full advantage of the features afforded by the Windows operating system, you must learn to use the Graphic Device Interface (GDI). In the .NET Framework, this interface is fully managed and is referred to as GDI+.

After this lesson, you will be able to

- Describe how to create and use a Graphics object
- Explain how to use pens, brushes, and colors
- Describe how to draw an outlined shape and a filled shape
- Describe how to render text
- Describe how to create a *GraphicsPath* and a *Region*

Estimated lesson time: 40 minutes

GDI+ is the name given to the .NET Framework's managed implementation of the GDI, which is used to display graphical information on the computer screen. This interface is wrapped into six different namespaces, which are broken down by related functionality.

The System.Drawing Namespaces

The array of functionality exposed by the *System.Drawing* namespaces is vast. Although an exhaustive dissection of these namespaces and their associated functionality is beyond the scope of this book, this lesson will familiarize you with the techniques involved in accessing the classes and methods they contain.

Table 7.1 summarizes the *System.Drawing* namespaces and the general functions of the classes they contain.

Table 7.1 *The System.Drawing Namespaces*

Namespace	Contains
System.Drawing	Most of the classes actually involved in rendering graphical content to the screen. This namespace is the primary namespace used for graphics programming.
System.Drawing.Design	Classes that are designed to extend the design time user interface.

This document is created with the unregistered version of CHM2PDF Pilot

System.Drawing.2D	Classes that are designed to render advanced visual effects.
System.Drawing.Imaging	Classes that allow advanced manipulation of image files.
System.Drawing.Printing	Classes that facilitate printing content.
System.Drawing.Text	Classes that facilitate advanced manipulation of fonts.

Almost all of the classes you will need to render graphics are provided in the *System.Drawing* namespace. For advanced rendering, you might need some classes in the *System.Drawing.2D* namespace, and to enable printing, you will need classes in the *System.Drawing.Printing* namespace.

The Graphics Object

The principal object used in rendering graphics is the *Graphics* object. The *Graphics* object is located in the *System.Drawing* namespace. A *Graphics* object represents the drawing surface of a visual element, such as a form, a control, or an *Image* object. Thus, a form has an associated *Graphics* object that can be used to draw inside the form; a control has an associated *Graphics* object that is used to draw inside the control, and so on. All of the actual rendering of visual elements is done by the *Graphics* object.

Because each *Graphics* object must be associated with a visual element, you cannot directly instantiate one with a call to the constructor. Instead, you must create a *Graphics* object directly from the visual element. Classes that inherit from *Control* (including *Form*) expose a *CreateGraphics* method that allows you to get a reference to the *Graphics* object associated with that control. The following example demonstrates how to access the *Graphics* object of a *Form* called *myForm*:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 2: Authoring Controls

Windows Forms programming is based on controls. Controls allow you to encapsulate discrete units of functionality and provide a graphical representation of that functionality to the user. The .NET Framework provides a variety of tools and techniques to help you create your own custom controls for your applications. In this lesson, you will learn about the different types of controls you can create and the tools available to help you create them.

After this lesson, you will be able to

- Describe the different types of user-authored controls
- Explain how each type of user-authored control is created
- Describe how to add properties, methods, and events to your controls
- Describe how to create an inherited control
- Describe how to create a user control
- Describe how to create a custom control

Estimate lesson time: 45 minutes

In the .NET Framework, controls are specialized classes that include code for rendering a graphical interface. There are several possible sources for the controls you use in your applications. The controls included with Microsoft's Visual Studio .NET provide a broad range of functionality and allow development of client applications with a rich array of features. You can also purchase controls from third-party developers and use them in your applications or use existing ActiveX controls in your applications. Third-party controls can provide functionality not found in the .NET Framework set of controls. If you need a control with a set of features not available in either the .NET Framework controls or the third-party controls, you can create your own controls.

Overview of Control Authoring

All controls in the .NET Framework inherit either directly or indirectly from the base class *Control*. This class provides all of the basic, low-level functionality that a control needs. For example, the *Control* class provides logic for handling user input through keyboard and mouse interaction; it provides logic for interacting with Windows; and it provides a set of properties, methods, and events that are common to all controls. It does not include any logic that creates the unique functionality of the control, nor does it contain any code for graphically rendering the control.

When creating a control, there are three basic techniques you can use:

- You can inherit from an existing control.
- You can create a user control.
- You can create a custom control.

Inheriting from Existing Controls

The simplest way to create a new control is to inherit from a preexisting control. When inheriting from an existing

This document is created with the unregistered version of CHM2PDF Pilot

control, the new control contains all of the functionality represented by the base control, but can also serve as a base for the addition of new functionality.

In addition to retaining all of the functionality of the base control, an inherited control also inherits the visual representation of the base control. Thus, a control that extends *Button* will look just like a regular *Button*. You do not need to create additional code to render the new control, but you can implement code to alter its appearance if you want.

Most Windows Forms controls can be used as a base class for inheritance unless they are specified as *NotInheritable* (*sealed*). For example, you could create a *TextBox* control with built-in validation or a *PictureBox* that allows the user to select filters to apply to the images it contains. You can also use inheritance to create controls that have the same functionality as the base control but have a different appearance. An example of this might be a *Button* that is round instead of square.

Inheriting from an existing control is the easiest and least time-consuming way to develop a new control. You should choose this technique if you need to replicate most or all of the functionality of an existing Windows Forms control but want to add custom functionality as well. You should also use this technique if you want to retain the functionality of an existing Windows Forms control but want to provide a new look and feel. You should not inherit from an existing Windows control if you need radically different functionality.

Inheriting from UserControl

Sometimes, a single control does not contain all of the functionality you need. For instance, you might want a control that you can bind to a data source to display a first name, last name, and phone number, each in a separate *TextBox*. Although it is possible to implement this logic on the form itself, it might be more efficient to create a single control that contains multiple text boxes, especially if this configuration is needed in many different applications. Controls that contain multiple Windows Forms controls bound together as a single unit are called user controls.

You can create a user control (sometimes called a composite control) by inheriting from the *UserControl* class. The *UserControl* class provides a base of functionality that you can build on by adding properties, method, events, and other controls. The UserControl designer allows you to add other Windows Forms controls to the design surface and create code to implement custom functionality. The result is a user-designed control made up of multiple Windows Forms controls that act as a single coherent unit.

Some limited design of the user control visual interface is possible, but most of the visual modifications you can make consist of simply configuring and positioning the constituent controls.

Creating a user control can be a relatively simple task or a complex one, depending on the level of complexity the control must expose. You should consider creating a user control if you need to combine the functionality of multiple existing controls and add custom logic to the unit.

Inheriting from Control

If you require a richer visual interface or functionality that is not achievable with a user control or an inherited control, you can create a custom control. Custom controls derive from *Control*, the base class for all controls. The *Control* class provides a lot of the functionality common to all controls, such as interacting with the mouse, and common events, such as *Click*. The control class also provides a set of properties that are useful for controls, such as *Font*, *ForeColor*, *BackColor*, *Visible*, and so on.

The *Control* class does not, however, provide any specific control functionality. All control logic specific to the control functionality must be programmed by the developer. Additionally, although the *Control* class provides many

This document is created with the unregistered version of CHM2PDF Pilot

properties that support a visual depiction, the developer must program all of the code for creating the visual representation of the control.

Creating a custom control is the most time-consuming approach to designing your control. Not only must you code all of the specific functionality for your control, but you must also write code to render the control, which can be time-consuming. You should consider creating a custom control when you need a richness of form or functionality that cannot be supplied by an inherited control or a user control.

Adding Members to Controls

You can add properties, methods, fields, and events to your controls in the same manner in which you would add members to a class. You can create member declarations with any appropriate access level, and the form that hosts the control will be able to call any public members of that control.

Public properties that are added to controls are automatically displayed in the Properties window at design time. Thus, they can be edited and configured by any user who is using your application. If you do not want a property to be displayed in the Properties window, you must mark the property with the *Browsable* attribute and specify false. The following example shows how to mark a property so that it will not be displayed in the Properties window at design time:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 3: Common Tasks Using Controls

There are a variety of tasks that are common to each type of control you develop. In this lesson, you will learn about some of the tasks common to control development.

After this lesson, you will be able to

- Add your control to the Toolbox
- Provide a Toolbox bitmap for your control
- Debug your control
- Manage control licensing
- View your control in Microsoft Internet Explorer

Estimated lesson time: 20 minutes

Adding Your Control to the Toolbox

Once you have created a control, you will want to be able to work with it in your development projects. You can make your controls accessible and easy to work with by adding them to the Toolbox. This allows you to add them to your projects at design time.

You can add new controls to the Toolbox by using the Customize Toolbox option. You can access this option by right-clicking on the Toolbox and choosing Customize Toolbox, which launches the Customize Toolbox dialog box. From there, you can browse to the file that contains your control and add it.

To add your control to the Toolbox

1. Right-click the Toolbox and choose Customize Toolbox. The Customize Toolbox dialog box opens.
2. Choose the .NET Framework Components tab and click the Browse button to open the File dialog box.
3. Browse to the folder that contains the DLL or EXE file that contains your control. Select the file and click Open.
4. Verify that your control has been added to the list displayed in the Customize Toolbox dialog box, and click OK.

   Your control is added to the Toolbox.

Providing a Toolbox Bitmap for Your Control

This document is created with the unregistered version of CHM2PDF Pilot

Visual Studio .NET provides a default icon that appears next to your controls in the Toolbox. You might, however, want to specify your own Toolbox bitmap. The following procedure details how to specify a bitmap to be used in the Toolbox with your control.

You specify a Toolbox bitmap by using the *ToolboxBitmapAttribute* class. This class is an attribute a specialized class that provides metadata about your control. Using the *ToolboxBitmapAttribute*, you can specify either a 16-by-16 pixel bitmap file to be used as your Toolbox bitmap, or you can specify a *Type* (*type*). If you specify a *Type* (*type*), your control will have the same Toolbox bitmap as that of the *Type* (*type*) you specify.

The *ToolboxBitmapAttribute* is attached to the class declaration of the control. In Microsoft Visual Basic .NET, it appears in angle brackets (<>) before the class declaration on the same line. In Microsoft Visual C#, it should go within braces ([]) on the line immediately preceding the class declaration for the control.

To provide a Toolbox bitmap for your control by specifying a file

Use the *ToolboxBitmapAttribute* to specify the file that contains the bitmap, as follows:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lab 7: Creating a Custom Control

In this lab, you will create a custom control called *PrettyClock*. This control will display the current time on the form in a graphically rich format. You will write code to render the control, add logic to implement the functionality, add code to the constructor to set the initial properties, and test your control. The solution to this lab is available in the \Labs\Ch07\Solution folder on the Supplemental Course Materials CD-ROM.

Before You Begin

There are no prerequisites to this lab.

Estimated lab time: 30 minutes

Exercise 7.1: Creating the Control

In this exercise, you will create a custom control called *PrettyClock*. You will inherit from the control class and add logic to paint the control and provide properties that control its appearance. Finally, you will add a timer component to provide the timekeeping functionality.

To create the project

1. From the File menu, choose New, and then choose Project.

   The New Project dialog box opens.
2. In the New Project dialog box, choose Windows Application. Name your project **ControlTest.vb** or **.cs** as appropriate, and click OK.
3. From the Project menu, choose Add New Item.

   The Add New Item dialog box opens.
4. In the Add New Item dialog box, choose Custom Control, and name your new item **PrettyClock.vb** or **.cs** as appropriate. Click Open to continue.

This document is created with the unregistered version of CHM2PDF Pilot

A new custom control is added to your project.

5. In Solution Explorer, right-click *PrettyClock* and choose View Code.

   The Code window opens. Note that an override of the *OnPaint* method has been stubbed out, and comments have been added to indicate that you need to add rendering code here.

You will now add code to create the control. Your control will render the current time to the screen, surrounded by a rectangle. You will use a *LinearGradientBrush* to paint the control to the screen and produce interesting graphic effects. You will also add properties to your control to allow developers to set the control's visual properties at run time.

1. At the top of the code editor, add the following lines to import the *System.Drawing* and *System.Drawing.Drawing2D* namespaces into your code:

   Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Review

These questions are intended to reinforce key concepts presented in the chapter. If you are unable to answer a question, review the relevant section and try again. Answers can be found in Appendix A.

1. Briefly describe the three types of user-developed controls and how they differ.
2. Describe the roles of *Graphics*, *Brush*, *Pen*, and *GraphicsPath* objects in graphics rendering.
3. Describe the general procedure for rendering text to a drawing surface.
4. Describe the role of the *LicenseProvider* in control licensing.
5. Describe how to create a form or control with a nonrectangular shape.

This document is created with the unregistered version of CHM2PDF Pilot

Chapter 8

Advanced .NET Framework Topics

About This Chapter

In this chapter, you will explore assorted topics in Microsoft .NET Framework programming. You will learn to implement print support, access external components, implement accessibility, provide help in your application, and implement globalization for your application.

Before You Begin

To complete the lessons in this chapter, you must have completed Chapters 1-7.

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 1: Implementing Print Functionality

Printing data and documents is a common task in any business setting. The .NET Framework provides the *PrintDocument* component to facilitate printing and a host of other classes to support and configure your printer settings. In this lesson, you will learn how to create and configure a *PrintDocument* object, and how to use it to print text, text files, and graphics.

After this lesson, you will be able to

- Describe how to create and configure a *PrintDocument* object
- Explain how to print text
- Describe how to print graphics
- Describe how to use the *PrintPreview* control and the Print Preview dialog box
- Explain how to configure page settings with the *PageSetupDialog* control
- Describe how to implement printer selection with the *PrintDialog* control

Estimated lesson time: 45 minutes

Printing will undoubtedly play a major part in your business applications. Although electronic file and document storage has become increasingly popular, printed material will not be replaced in the near future. Thus, you must be able to implement print capability in your applications. The .NET Framework provides classes and components that allow you to quickly and easily implement print support for your programs.

The PrintDocument Component

A printed document is represented by the *PrintDocument* component in the .NET Framework. Although it is not a control with a visual component, the *PrintDocument* component can be found on the Windows Forms tab of the Toolbox, from where it can be added to your application in the designer.

The *PrintDocument* object encapsulates all of the information necessary to print a page. It exposes a *PrinterSettings* property that contains information about the capabilities and settings of the printers; a *DefaultPageSettings* property that encapsulates the configuration for printing each printed page; and a *PrintController* property that describes how each page is guided through the printing process. Through these properties, it is possible to attain a very exacting level of control over the print process.

Creating a PrintDocument Object

You can create a *PrintDocument* object either in the designer at design time or in code at run time. You can drag an instance of the *PrintDocument* object directly from the Toolbox to the designer. The new instance appears in the

This document is created with the unregistered version of CHM2PDF Pilot

component tray and is automatically configured to work with the system's default printer.

You can also create an instance in code, as follows:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 2: Accessing and Invoking Components

Although the .NET Framework provides a vast array of built-in functionality, you might want to access external components from time to time. You might, for instance, have built an assembly yourself using the .NET Framework or want to access an older COM component. You might also need to make a call to a Web Service or even interact directly with the Microsoft Windows Application Programming Interface (API). Visual Studio .NET allows you to incorporate these entities into your application with a minimum of effort.

After this lesson, you will be able to

- Explain how to access a .NET Framework assembly
- Describe how to access a COM component or ActiveX control
- Describe how to access a Web Service
- Explain how to make a call to the Windows API

Estimated lesson time: 30 minutes

You will undoubtedly need to access external code at some time. Whether reusing components that you have previously developed, accessing Web Services, or making calls to the Windows API, Visual Studio .NET makes it easy to create references to external components and call them in your code.

Accessing .NET and COM Type Libraries

You can access any .NET or COM library on your system without much difficulty. These components might represent other development projects you have created, legacy COM components, or business logic that you need to incorporate into your application. The generalized scheme for accessing .NET or COM components is to create a reference to the type library, *Import* (*use*) the type library in your application, and then declare and instantiate the class you want to use.

You can obtain a list of available type libraries in the Add Reference dialog box. To display the Add Reference dialog box, right-click References in the Solution Explorer under your project, and choose Add Reference from the shortcut menu. The Add Reference dialog box is shown in Figure 8.4.

The Add Reference dialog box has three tabs. Choosing the .NET tab displays available .NET assemblies. Choosing the COM tab displays the available COM type libraries, and choosing the Projects tab displays available projects. If the item you want to add a reference to is not listed, you can browse to the file location by clicking the Browse button. Once you have located the reference you would like to add, you can select it with the Select button. The reference is added to your project in the Solution Explorer.

This document is created with the unregistered version of CHM2PDF Pilot

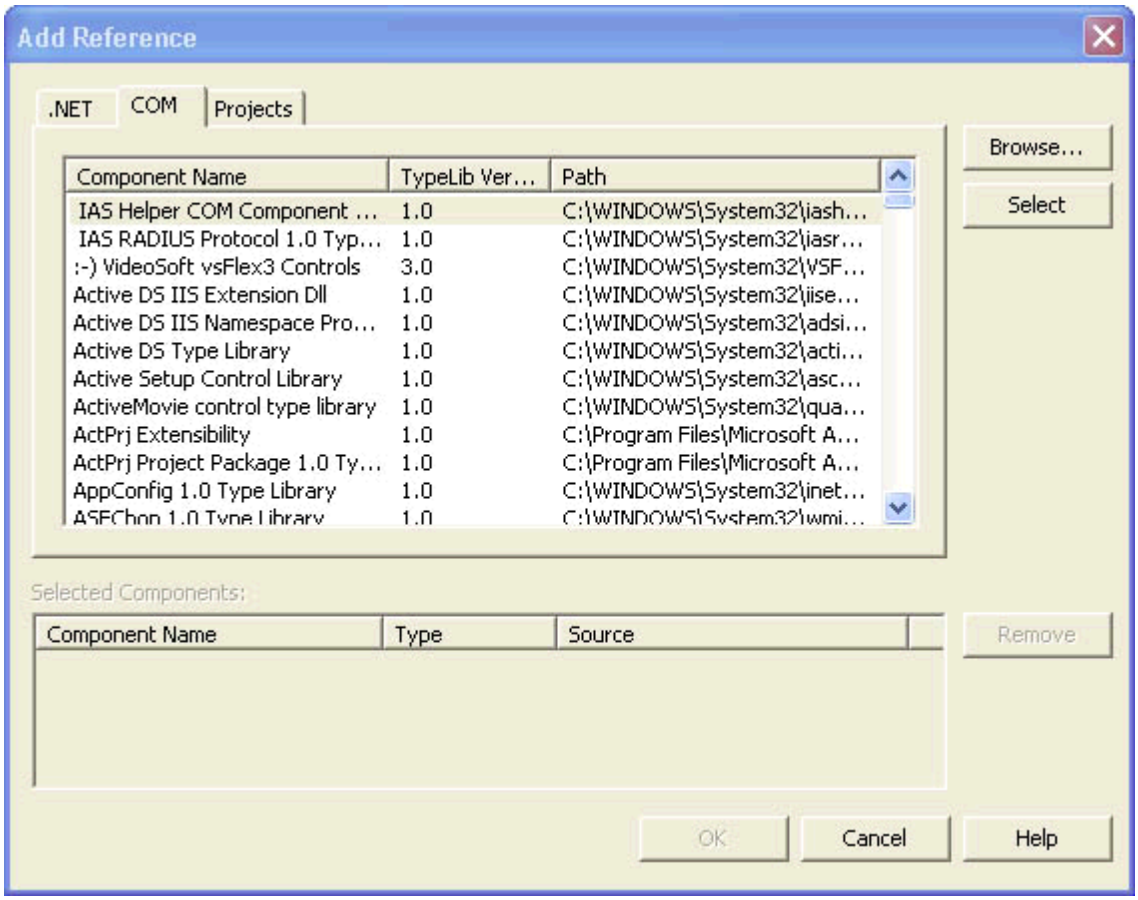


Figure 8.4 *The Add Reference dialog box*

Once you have added a reference to an assembly or COM type library, you can use the types contained in that reference in your application by using the fully qualified name of the type. If you want to obviate the need to use the fully qualified name, you can use the *Imports* (Visual Basic .NET) or *using* (Visual C#) keyword to make members accessible. Chapter 1 describes how to use the *Imports* and *using* statements.

Once a reference has been established, you can declare and instantiate the types contained in the type library as you would any other type.

To access a .NET assembly or COM type library

1. In Solution Explorer, right-click the References node under your project, and choose Add Reference.

   The Add Reference dialog box opens.
2. Select the reference you want to add. COM type libraries are listed under the COM tab, .NET assemblies are listed under the .NET tab, and projects can be found under the Projects tab.
3. Use the *Imports* (Visual Basic .NET) or *using* (Visual C#) keyword to import the reference into your application.
4. Declare and instantiate the types exposed by your reference as normal.

Instantiating ActiveX Controls

ActiveX is an implementation of COM. As such, you can add a reference to an ActiveX control type library in much the same way as you would add a reference to a COM library. You can then add the ActiveX control to the Toolbox and add it to your applications at design time.
Although ActiveX controls can be hosted in Windows Forms, there are some significant performance drawbacks to doing so. You should use ActiveX controls only when a .NET control with the same functionality is not available.

This document is created with the unregistered version of CHM2PDF Pilot

To instantiate an ActiveX control

1. Add a reference to the type library that contains the ActiveX control, as described previously in this lesson.
2. Add the ActiveX control to the Toolbox by customizing the Toolbox. This is described in greater detail in Chapter 7.
3. Drag an instance of the control from the Toolbox to the designer to add it to your project at design time. Alternatively, you can declare and instantiate the control in code and add it to your application dynamically at run time, as described in Chapter 2.

Accessing a Web Service

An integral feature of the .NET Framework is the Web Service. A Web Service is a type of class that is hosted on the Internet. You can declare an instance of the Web Service in your application, and then call its methods either synchronously or asynchronously.

You can create a reference to a Web Service in the Solution Explorer. You can display the Add Web Reference window by right-clicking References under your project and choosing Add Web Reference. This displays a screen that allows you to navigate to the URL of a Web reference or to search online if you do not know the URL. The Add Web Reference window is shown in Figure 8.5.

If you do not know the address of the Web Service you would like to use, the window displays directories in which to search. The Universal Description Discovery Integration (UDDI) directory allows you to search the Web for companies that provide Web Services. When a company is selected, the Web Services it offers will be displayed. Choosing a Web Service will display the XML contract in the left page of the window and the name of the Web Service in the right window. You can also view a more reader friendly description of the Web Service by removing "?WSDL" from the address displayed in the status bar and reloading the page. Once you have verified the correct Web reference to add, click the Add Reference button to add the Web reference to your project.

This document is created with the unregistered version of CHM2PDF Pilot

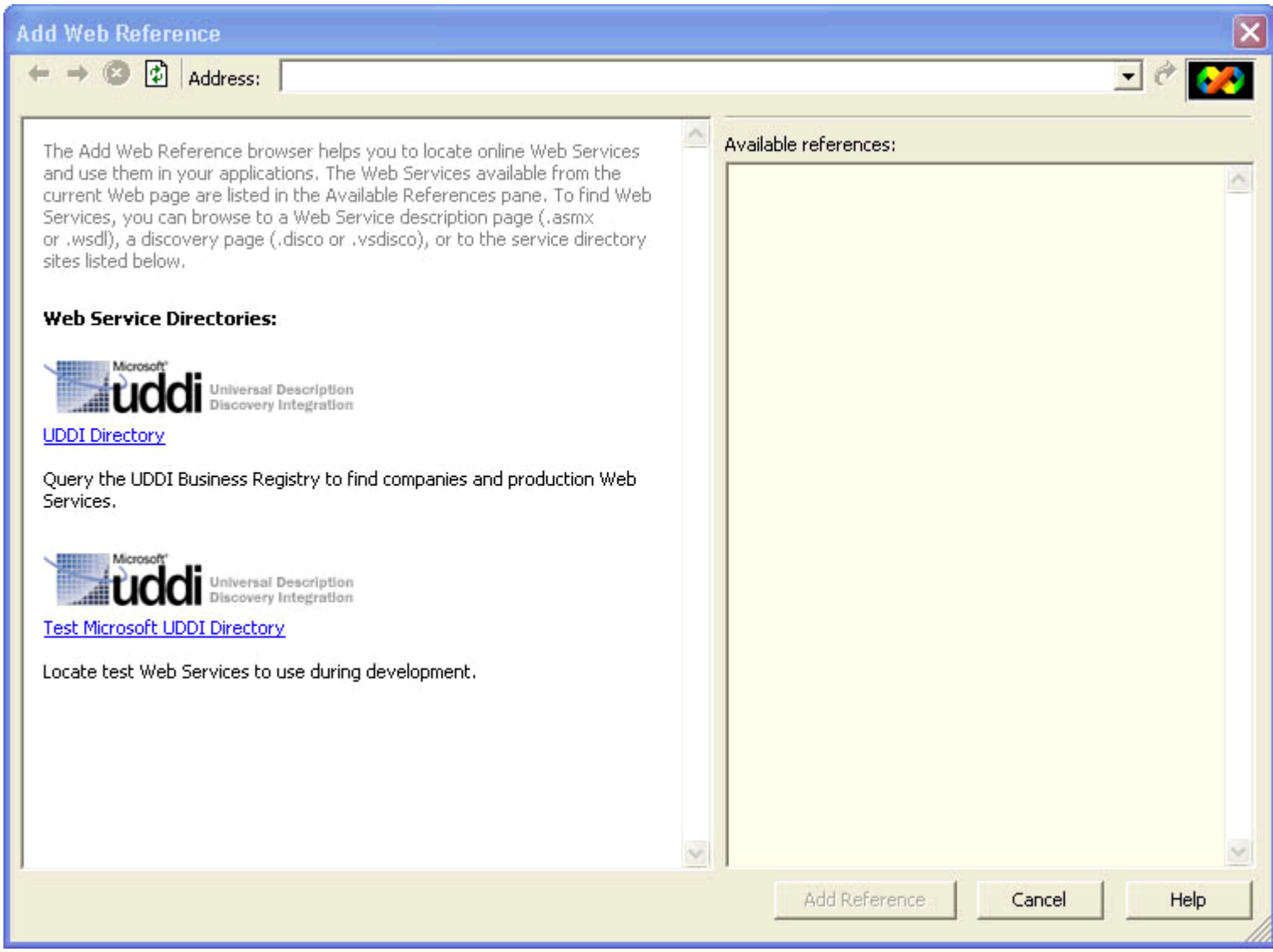


Figure 8.5 *The Add Web Reference window*

When an instance of a Web Service is instantiated, the .NET Framework actually creates an instance of a proxy class that represents the Web Service. This class resides in your application and exposes all of the methods provided by the Web Service. When one of these methods is called, the call is relayed to the Web Service at the address specified on the Internet. You can instantiate a Web Service just like any other component. The following example demonstrates how to instantiate a Web Service called *WebService1*:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 3: Implementing Accessibility

When designing your applications, you must take all potential users into account. For many applications, business and otherwise, this includes people that have difficulty using the standard user interfaces provided by most Windows Forms applications. By designing your applications to be accessible, you can increase the user base and usefulness of your programs.

After this lesson, you will be able to

- Describe the basic principles of designing for accessibility
- Explain the accessibility requirements for the Certified for Windows logo program
- Describe the accessibility properties of standard Windows Forms controls and their roles

Estimated lesson time: 20 minutes

People with accessibility requirements are a significant part of the workforce. In order to meet the needs of today's business application programming, applications must be accessible to people with all kinds of different needs.

Accessibility is facilitated in a variety of ways. Microsoft Windows XP provides a variety of accessibility aids that can interact with your applications. For example, SoundSentry allows the user to specify a visual cue whenever the system emits a sound. These accessibility aids are designed to integrate seamlessly with your applications to provide an accessible user experience. Additionally, there are accessibility principles you can design into your application.

Accessibility Design

An application begins to be accessible in the design phase. By planning ahead, you can create an application that integrates the principles of accessible design into the user interface. Some of these principles are

- Flexibility
- Choice of input methods
- Choice of output methods
- Consistency
- Compatibility with accessibility aids

Flexibility is a key principle in accessibility design. Users should be able to customize the user interface to their specific needs. They should also be able to choose from a variety of input methods, such as pointing devices or keyboards. The application should provide keyboard access to the important features of the application and mouse click access for common tasks. Likewise, users should have a choice of output methods and be able to choose between sound, visual cues, graphics, and text. Your application should interact in a consistent manner with other applications and within its own operation. In addition, your application should be compatible with accessibility aids.

This document is created with the unregistered version of CHM2PDF Pilot

Accessibility and the Certified for Windows Program

In order to provide a standard set of guidelines for designing accessible applications, Microsoft has incorporated five requirements for applications to be certified in the Certified for Windows program. This section discusses the requirements for the Certified for Windows program and how to implement them.

Support Standard System Settings

The first requirement states that applications must support standard system settings for size, color, font, and input. This ensures that all of a user's applications have a consistent user interface that will inherit the look and feel provided by the system settings. Thus, users with accessibility needs can configure their systems once, and all of their applications will automatically adopt a consistent appearance.

You can implement this requirement in your applications by using *System* objects exposed through the .NET Framework. The .NET Framework provides a palette of system colors that can be used to set the colors of your application. When the system settings change, the colors of your application's user interface will change. Similarly, you can leave the font determination up to the system by not explicitly setting the *Font* property for your controls. If the *Font* property is not changed from its default value, it will conform to the system font settings. The exception to this is the *Font*.*Size* property, which can be changed while maintaining accessibility requirements. Your application should be designed so that it can easily be used on a screen with 640 x 480 resolution and should not require any special methods of input beyond the keyboard and pointing devices.

Ensure Compatibility Using the High Constrast Option

Users requiring a high degree of legibility will select the High Contrast option. This option sets the Windows color scheme to a scheme that provides maximum contrast between light and dark portions of the user interface.

You can ensure that your application conforms to the High Contrast option requirement by only using system colors or colors chosen by the user. Thus, when the High Contrast option is chosen, your application will conform. You should also avoid using background images for your forms because they decrease contrast and legibility.

Provide Documented Keyboard Access to All Features

Your application should implement keyboard access for all features, and this access should be documented and discoverable. The use of shortcut keys to access controls and menu items, as well as setting the Tab order, allows you to implement keyboard access for your user interface features. The importance of documentation for these features cannot be overlooked. A user must have some means of discovering keyboard access for features and this should be provided through careful, complete, and well-organized documentation.

Provide Notification of the Keyboard Focus Location

The location of the keyboard focus is used by accessibility aids such as Magnifier and Narrator. Thus, it is important that the user and the application always have a clear understanding of where the focus is. For the most part, this is

This document is created with the unregistered version of CHM2PDF Pilot

facilitated by the .NET Framework. When designing your program flow, however, you should always incorporate code to set the focus to the first control on a form when the form is first displayed, and the Tab Order should be set to provide logical program flow.

Convey No Information by Sound Alone

Your applications should never rely on sound to convey important information. Although sound is an important cue for some users, other users might be unable to perceive an alert that is communicated by an audio cue. When using sound to convey information, you should combine the notification with visual cues, such as flashing the background color of the active window, displaying a messagebox notification, or using a nonmodal screen that displays information while the application continues to execute.

Accessibility Properties of Windows Forms Controls

In addition to properties that affect the visual interface of a control, each Windows Forms control has five accessibility-related properties that determine how a control interacts with accessibility aids. These properties are summarized in Table 8.2.

Table 8.2 *Accessibility Properties of Windows Forms Controls*

Property	Description
AccessibleDescription	Contains the description that is reported to accessibility aids.
AccessibleName	Contains the name that is reported to accessibility aids.
AccessibleRole	Contains the role of the control that is reported to accessibility aids.
AccessibilityObject	Contains an instance of *AccessibleObject*, which provides information about the control to usability aids. This property is read-only and is set by the designer.
AccessibleDefaultActionDescription	Contains a description of the default action of a control. This property cannot be set at design time and must be set in code.

These properties provide information to accessibility aids about the role of the control in the application. Accessibility aids can then present this information to the user or use this information to make decisions on how to display the control.

The *AccessibilityObject* property contains the *AccessibleObject* class instance that provides information to accessibility aids. The *AccessibleName*, *AccessibleDescription*, and *AccessibleDefaultActionDescription* properties contain information that accessibility aids can use to describe your user interface. The *AccessibleRole* property contains an enum value that determines how an accessibility aid treats your control. This property is normally set to (Default), which indicates that each control is acting in its usual manner. It is important to set this property, however, when a control is acting in a manner more consistent with another type of control, or for custom controls. For example, if you have a custom control that displays data in a chart form, you would set the

This document is created with the unregistered version of CHM2PDF Pilot

AccessibleRole property value to Chart. The accessibility aids would then treat the control as a chart.

Lesson Summary

- When designing for accessibility, keep the following principles in mind:
  - Flexibility
  - Choice of input methods
  - Choice of output methods
  - Consistency
  - Compatibility with accessibility aids
- The Certified for Windows logo program has the following accessibility requirements for applications:
  - Support standard system settings
  - Be compatible with High Contrast mode
  - Provide documented keyboard access for all UI features
  - Provide notification of the focus location
  - Convey no information by sound alone
- Windows Forms controls expose the following accessibility-related properties:
  - *AccessibleDescription*
  - *AccessibleName*
  - *AccessibleRole*
  - *AccessibilityObject*
  - *AccessibleDefaultActionDescription*

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 4: Implementing Help in Your Application

Too often, the importance of proper program documentation is overlooked. Providing help in your application allows users to learn it more quickly, thereby increasing productivity and saving money. In this lesson, you will learn how to display help in your applications.

After this lesson, you will be able to

- Describe how to use the *Help* class to display help
- Explain how to use the *HelpProvider* component

Estimated lesson time: 20 minutes

For complex or difficult applications, you will want to provide built-in help. You can provide help in the form of HTML files that contain a set of linked help topics or as Compressed HTML (CHM) files that you create with Microsoft HTML Help Workshop. Both of these approaches allow you to display dynamically linked help topics that you can show to your users.

The Help Class

Your application can display HTM or CHM files to users by using the *Help* class. The *Help* class encapsulates the HTML Help 1.0 engine and provides static methods that allow you to display your help files. The two methods exposed by the *Help* class are *ShowHelp* and *ShowHelpIndex*.

The *ShowHelp* method is used to display a help file for a particular control. This method requires the control that is the parent of the *HelpDialog* box to be displayed as well as the URL of the help file. The URL specified can either be a file URL (C:\myHelp.htm) or an HTTP URL (HTTP://myHelp.htm). Because the *ShowHelp* method is *Shared* (*static*), you do not need to create an instance of the *Help* class to use it. In fact, the *Help* class cannot be instantiated. The following example shows how to display help with the *ShowHelp* method:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 5: Globalization and Localization

The .NET Framework provides unprecedented developer support for globally accessible applications. You can create applications that adapt to different languages, currency formats, date/time formats, and other culture-sensitive information. In this lesson, you will learn how to create globalization and localization in your application.

After this lesson, you will be able to

- Explain what is meant by globalization and what is meant by localization
- Explain how to implement globalization and localization in the user interface
- Describe how to prepare culture-specific formatting
- Describe how to implement right-to-left formatting
- Explain how to convert existing text encoding schemes
- Describe how to validate non-Latin input

Estimated lesson time: 30 minutes

If your company does business in the global marketplace, you must design your applications to accommodate users from a variety of cultures. Users in other parts of the world might be unfamiliar with United States formatting standards for currency or dates, or might be unable to understand English altogether. By designing international support into your applications, you increase the user base of your application and enable its global use.

Globalization and Localization

Globalization and localization are two different but related processes. Globalization involves applying culture-based formatting to existing data, and localization involves retrieving the appropriate piece of data based on the culture. The following examples illustrate these differences:

- **Globalization.** Currency is formatted in some countries using a period (.) as a thousand separator and a comma as a decimal separator. A globalized application will take the existing data for currency and format it appropriately based on location.
- **Localization.** The title of a form might be displayed in English or French depending on the country in which it is deployed. A localized application will retrieve the appropriate string, in English or French, and display it according to the location.

Culture

In your application, culture refers to the cultural information about the region in which the application is being used. In the .NET Framework, cultures are identified using a culture code. The culture code is a code that represents to the framework what the current language is. A culture code can also specify information about the region in which an

This document is created with the unregistered version of CHM2PDF Pilot

application is being used. Generally, the culture code is either a two letter code that only specifies the language that is spoken or a two letter language code followed by a dash and another two letter code that specifies the region. Culture codes that only specify the language are called *neutral cultures*, whereas cultures that specify the language and the region are called *specific cultures*. The following list shows some examples of culture codes:

- **En.** Specifies English language, no region.
- **en-CA.** Specifies English language, Canada.
- **af-ZA.** Specifies Afrikaans language, South Africa.
- **eu.** Specifies Basque language, no region.
- **kn-IN.** Specifies Kannada language, India.
- **tr.** Specifies Turkish language, no region.

A complete list of culture codes can be found in the CultureInfo Class reference topic in the .NET Framework reference documentation.

Although most culture codes follow this format, it is important to note that there are some exceptions. For example:

- **Cy-uz-UZ.** Specifies Uzbek language, Uzbekistan, Cyrillic alphabet.
- **Lt-uz-UZ.** Specifies Uzbek language, Uzbekistan, Latin alphabet.
- **zh-CHT.** Specifies traditional Chinese language, no region.
- **zh-CHS.** Specifies simplified Chinese language, no region.

Getting and Setting the Current Culture

The application automatically reads the system culture settings and implements them. You can change the current culture of your application programmatically by setting the current culture to a new instance of the *CultureInfo* class. This class holds all of the information about a culture and how it should interact with your application. For example, the *CultureInfo* class contains information on which calendar is used, which type of currency formatting is used, which type of date formatting is used, and so on. You can set the culture of an application programmatically by setting the *CurrentThread*.*CurrentCulture* property to a new instance of the *CultureInfo* class. The *CultureInfo* class takes a string that represents the appropriate culture code as a parameter. The following example demonstrates how to set the current culture to French-Canadian:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lab 8: Creating a Localized Form with Print Support

In this lab, you will create an application for international use. The application will be a very simple forms-based application that accepts orders from the user and adds them to a list box. You will implement globalization for your form and create a localized version for use in Italy. You will also add print support to your form. The solution to this lab is available in the \Labs\Ch08\Solution folder on the Supplemental Course Materials CD-ROM.

Before You Begin

There are no prerequisites to this lab.

Estimated lab time: 60 minutes

Exercise 8.1: Creating the Form

Business is booming! In this exercise, you will create an application to handle orders. The application will allow users to enter values for quantity and price, and add a summary of the order to a list box.

To create the form

1. Create a new Windows Forms project. Add the controls listed in Table 8.4 to the form, and set the properties as indicated.
   Table 8.4 *Controls and Properties for the Project*

Control	Property	Value
Label1	*Name* *Text*	lblQuantity Quantity
Label2	*Name* *Text*	lblPrice Price

This document is created with the unregistered version of CHM2PDF Pilot

Label3	*Name* *Text*	lblTotal Total
TextBox1	*Name* *Text*	txtQuantity (empty)
TextBox2	*Name* *Text*	txtPrice (empty)
TextBox3	*Name* *Text* *ReadOnly*	txtTotal (empty) True
Button1	*Name* *Text*	btnAdd Add Order
Button2	*Name* *Text*	btnClear Clear Order
Button3	*Name* *Text*	btnClearAll Clear All Orders
Button4	*Name* *Text*	btnPrint Print
ListBox1	*Name*	lstOrders

After adding controls, your form should look Figure 8.9.

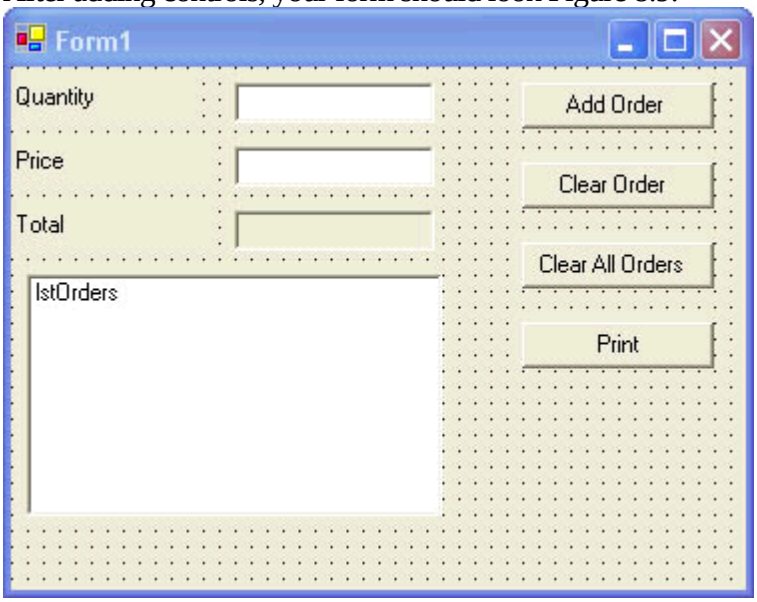


Figure 8.9 *Lab Form1*

2. Implement handlers for the *KeyPress* events of *txtPrice* and *txtQuantity*. They should verify that the keystroke represents a number. The following example demonstrates an appropriate event handler for *txtPrice*:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Review

These questions are provided as a review for this chapter. If you are unable to answer any of these questions, review the relevant section and try again. Answers to these questions can be found in Appendix A.

1. Briefly describe how to use the *PrintDocument* component to print a document. Discuss maintaining correct line spacing and multipage documents.
2. Explain how to use the *Begin* and *End* methods on a Web Service to make an asynchronous method call.
3. Briefly describe the five accessibility requirements of the Certified for Windows logo program.
4. Explain how to use the *HelpProvider* component to provide help for UI elements.
5. Describe how to create localized versions of a form.
6. Explain how to convert data in legacy code page formats to the Unicode format.
7. Explain the difference between Globalization and Localization.

This document is created with the unregistered version of CHM2PDF Pilot

Chapter 9

Assemblies, Configuration, and Security

About This Chapter

In this chapter, you will learn about creating, configuring, and securing assemblies. You will learn how to create private, shared, and resource-only assemblies. You will learn how to use the .config file to configure your application and to optimize performance in your code. Finally, you will learn to secure your application against unauthorized users and to protect your code from unauthorized use.

Before You Begin

To complete the lessons in this chapter, you must have completed Chapters 1-8.

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 1: Assemblies and Resources

Assemblies are the primary units of deployment in a Microsoft .NET Framework application. Assemblies are collections of types and resources that are bound together to create a logical unit of functionality. Assemblies are also self-describing and contain all of the information required by the run time to interpret their contents and configure execution. In this lesson, you will learn how to create an assembly, how to create resource files and retrieve resources dynamically, and how to share your assembly between applications.

After this lesson, you will be able to

- Describe the parts of an assembly
- Describe how to set identity information for an assembly
- Explain how to create a DLL assembly
- Describe how to retrieve resources programmatically
- Describe how to create a resource-only assembly
- Explain how to apply a strong name to your assembly
- Describe how to install your assembly to the Global Assembly Cache

Estimated lesson time: 45 minutes

Assemblies

Assemblies are the fundamental building blocks of a .NET Framework application. They contain the types and resources that make up an application and describe those contained types to the common language runtime. Assemblies enable code reuse, version control, security, and deployment.

Put simply, an assembly is a project that compiles to an EXE or a DLL file. Although .NET EXE and DLL files resemble their predecessors externally, the internal structure of an assembly is quite different from that of an EXE or DLL created with earlier development tools. An assembly consists of four internal parts:

- The assembly manifest, or metadata. This contains information about the assembly that the common language runtime uses to obtain information about the assembly.
- The type metadata, which exposes information about the types contained within the assembly.
- The intermediate language code for your assembly.
- Resource files, which are nonexecutable bits of data, such as strings or images for a specific culture.

The assembly manifest contains the metadata that describes the assembly to the common language runtime. The common language runtime then uses the information in the assembly manifest to make decisions about the assembly's execution. An assembly manifest contains the following information:

- **Identity.** It contains the name and version number of the assembly, and can contain optional information such as locale and signature information.
- **Types and resources.** It contains a list of all the types that will be exposed to the common language runtime

This document is created with the unregistered version of CHM2PDF Pilot

as well as information about how those types can be accessed.

- **Files.** It contains a list of all files in the assembly as well as dependency information for those files.
- **Security permissions.** The manifest describes the security permissions required by the assembly. If the permissions required conflict with the local security policy, the assembly will fail to execute.

For the most part, the developer does not have to be concerned with the contents of the assembly manifest. It is compiled and presented to the common language runtime automatically. The developer does, however, need to explicitly set the metadata that describes the identity of the assembly.

The identity of the assembly is contained in the AssemblyInfo.vb or .cs file for your project. You can set identity information for your assembly by right-clicking the AssemblyInfo icon and choosing View Code from the drop-down menu. The code window will open to the AssemblyInfo code page, which contains default null values for several assembly identity attributes. The following code example shows an excerpt from the AssemblyInfo file. Note that Microsoft Visual Basic .NET and Microsoft Visual C# include slightly different attributes by default.

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 2: Configuring and Optimizing Your Application

The .NET Framework provides tools that you can use to configure and optimize your application after it has been deployed. The configuration file allows you to specify location or version information for dependent assemblies and allows you to configure application properties as well. You can use the performance monitor utility to identify bottlenecks in your application and optimize your code for future builds. In this section, you will learn to create and use a configuration file, to configure your application with dynamic properties, and to use the performance monitor to identify bottlenecks in your code.

After this lesson, you will be able to

- Explain how to create a .config file
- Describe the basic schema of a .config file
- Explain how to set dynamic properties in your application
- Describe how to configure dynamic properties for your application
- Explain how to use the performance monitor to diagnose bottlenecks in your application

Estimated lesson time: 30 minutes

You can provide a configuration file for your application. A configuration file allows you to configure application properties after the application has been deployed without recompiling your code. The configuration file is an XML file that contains information about how your application should be configured. You learned how to use .config files to configure Trace switches in Chapter 5. You can also use configuration files to configure other aspects of your program.

Creating the Configuration File

A configuration file is simply an XML file with the appropriate tags and an appropriate name. A configuration file for an application will have the name <*name*>.<*extension*>.config, where <*name*> is the name of the application, and <*extension*> is the extension of the application (such as .exe). Thus, the .config file for an application named myApplication.exe would be myApplication.exe.config. A configuration file must be located in the same folder as the application assembly that it configures.

The content of a configuration file must be configured in the .config file schema. The basic structure of a configuration file is as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <! configured elements go here  >
```

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 3: Securing Your Application

When writing an application, it is important to protect your code. Hackers and malicious users cause untold amounts of damage to businesses every year. As a developer, it is your responsibility to ensure that your code is secure from attack. In addition, sensitive information such as company databases and personal information must be kept secure from unauthorized users. The .NET Framework provides ample tools to secure your applications from unauthorized use. In this lesson, you will learn how to use imperative and declarative security checks to secure your application. You will learn how to protect against unauthorized users, unauthorized code groups, and unidentified code. You will also learn to implement custom security checks.

After this lesson, you will be able to

- Explain what a permission is and how it is used
- Describe the difference between declarative and imperative security
- Explain how to configure role-based security
- Explain how to configure code access security

Estimated lesson time: 40 minutes

Security is about protection. You can use the security features provided by the .NET Framework to protect your code from unauthorized users and to protect the system from unauthorized use of your code. Overall system security policy is set by the system administrator. The administrator decides what kind of code the machine will be allowed to execute, whether or not a particular assembly is trusted, and if so, what kind of trust is extended to the assembly. The security policy set by the system administrator cannot be overridden by your code: it is the highest level of security on a given machine.

You use security to further protect your application within the bounds set by the system administrator. You can use role-based security to authorize users and code access security to protect your code from being misused by unauthorized callers. Security authorizations can be either imperative or declarative. In imperative security, permission to execute is demanded at run time. Declarative security, on the other hand, specifies permissions required by the assembly in the assembly manifest. If the required permissions are in conflict with the security policy, the assembly will not be allowed to execute. *Permission* objects are the central objects in code security.
Unless otherwise specified, all of the code examples in this lesson use *Imports System.Security.Permissions* for Visual Basic .NET and use *using System.Security.Permissions* for Visual C#.

Permissions

Permissions are the primary objects in security. A permission is a code object that represents a user, an identity, or a code resource. You use *Permission* objects for a variety of security-related functions. *Permission* objects are used to represent security clearances as well as to enforce security policy.

This document is created with the unregistered version of CHM2PDF Pilot

The IPermission Interface

All permissions implement the IPermission interface. Although you will very rarely, if ever, need to implement this interface, an examination of the methods described by the IPermission interface will give insight into how the objects that implement this interface work. The member methods of the IPermission interface are described in Table 9.2.

Table 9.2 *Methods of the IPermission Interface*

Method	Description
Copy	Creates and returns an identical copy of the permission.
Demand	Walks the call stack and throws a *SecurityException* if all callers in the call stack have not been granted the permission.
Intersect	Creates and returns a permission that is the intersection of two permissions.
IsSubsetOf	Determines whether the current permission is a subset of a specified permission.
Union	Creates a permission that is the union of the current permission and the specified permission.

All security permissions in the .NET Framework must implement the IPermission interface, which provides a common level of functionality for all security objects. Permissions can use the *Demand* method to enforce security. The method requires that callers must have been granted the appropriate permission to access the protected code. If the appropriate permission has not been granted, a *SecurityException* is thrown. The *Copy* method creates an identical copy of the permission, and the *IsSubsetOf* method determines whether the current permission is a subset of a specified permission. The *Union* and *Intersect* methods are used to create a new permission from two specified permissions of the same type. The permission returned by a *Union* method represents the sum of the two permissions specified, and the permission returned by the *Intersect* method represents only those resources that both permissions have in common. In the following sections, you will learn how to use these methods as implemented in the *Permission* objects provided by the .NET Framework.

Configuring Role-Based Authorization

Role-based security is security that grants or denies access to an application or resource based on the identity and role of the user. For example, suppose you have an application that is used by both managers and clerks. You might want to allow everyone to access some parts of the application, but make sensitive parts of the application, such as payroll or personal information, available to managers only. You would use role-based authorization to implement this kind of security.

All of the code examples in the role-based security section use *Imports System.Security.Principal* for Visual Basic .NET and use *using System.Security.Principal* for Visual C#.

The Principal

This document is created with the unregistered version of CHM2PDF Pilot

In the .NET Framework, authenticated users are represented by a *Principal* object. The *Principal* object contains information about a user's identity and role, and can be used to validate identity against a *PrincipalPermission* object, which is used to protect sensitive parts of an application from unauthenticated users.

When writing a Windows application, you can use the built-in security in Windows to verify the identity and role of the current user. You can associate the *WindowsPrincipal* object that represents the current user with your application by setting the principal policy for the current application domain as follows:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lab 9: Configuring and Securing an Application

In this lab, you will modify the international application you created in Lab 8. You will add code to allow the user to set the culture by editing the configuration file. Then, you will configure role-based security for your application to prevent unauthorized users from using it. Finally, you will configure code access security to prevent unauthorized printing. The solution to this lab is available in the \Labs\Ch09\Solution folder on the Supplemental Course Materials CD-ROM.

Before You Begin

Before you begin this lab, you must have either completed the Chapter 8 lab or loaded the Chapter 8 solution from the CD-ROM. Additionally, you must be working on a computer where you are a member of the Windows Administrators group.

Estimated lab time: 45 minutes

Exercise 9.1: Adding the Configuration File

In this exercise, you will add a configuration file to your application, and then add code to your application that reads a culture value from the configuration file and sets the *CurrentCulture* and the *CurrentUICulture* to that value. This exercise begins with the Chapter 8 solution already loaded into Visual Studio .NET.

To add the configuration file

1. In Solution Explorer, right-click *frmLanguage* and choose Delete. Click OK in response to the dialog box.

   frmLanguage is deleted from the project.
2. In Solution Explorer, right-click Form1 and choose View Code.

   The code window for Form1 opens.
3. Locate the constructor for Form1. In the constructor, delete the following two lines of code:

This document is created with the unregistered version of CHM2PDF Pilot

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Review

The following questions are intended to reinforce key concepts and information presented in this chapter. If you are unable to answer a question, return to the appropriate lesson to review, and then try the question again. Answers to the questions can be found in Appendix A.

1. Describe how to sign your assembly with a strong name. Why would you want to do this?
2. Describe how to use code to retrieve resources at run time.
3. Explain how to retrieve information from the configuration file at run time. How would you store information in the configuration file at design time?
4. You are creating a solution that must be accessed by members of a group called Developers and Administrators on the local machine. Describe a plan to implement this security scheme.
5. Briefly highlight the differences between imperative and declarative security as they pertain to code access security.

This document is created with the unregistered version of CHM2PDF Pilot

Chapter 10

Deploying Your Application

About This Chapter

In this chapter, you will learn how to plan and execute the deployment of your project. You will learn how to plan deployments from the network, from the Internet, and from removable media. In addition, you will learn how to create a setup project for your application and deploy your application to a host computer.

Before You Begin

To complete the lessons in this chapter, you must have completed Chapters 1-9.

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 1: Planning the Deployment of Your Project

For an application to be successful, it must reach the intended audience. Microsoft Visual Studio .NET provides many different options for deploying your project, from simple XCOPY deployment to fully configurable Microsoft Windows Installer options. In this lesson, you will learn how to choose the best deployment option for your project.

After this lesson, you will be able to

- Explain how to use XCOPY to deploy a Microsoft Windows Forms application
- Describe how to create and configure a setup project for your application using the Setup Project wizard
- Explain how to plan a deployment from removable media
- Explain how to plan a deployment from a network share
- Explain how to plan a deployment from a Web server

Estimated lesson time: 45 minutes

Once an application has been developed, tested, and debugged, it is ready for deployment to client machines. The goal of a deployment is the simple and easy installation of application files and any other required files to a client machine. For simple programs, the Microsoft .NET Framework allows you to deploy your application by copying the application directory to the target machine. For more complex applications, Visual Studio .NET provides Windows Installer technology, which allows you to make fully configurable setup projects for your application.

XCOPY Deployment

XCOPY deployment is a simple and straightforward deployment scenario. The name is derived from the DOS XCOPY command, which copies the contents of a directory (including subdirectories) to a target directory. In certain circumstances, you can use this very simple method to deploy your applications.

XCOPY deployment is accomplished from the command prompt. You use the XCOPY command to specify the source directory and the target directory. The /s flag is used to indicate that subdirectories are to be copied as well. For example, to copy the MyApplication directory (including all subdirectories) from drive D: to drive C:, the following command-line command is used:

This document is created with the unregistered version of CHM2PDF Pilot

Lesson 2: Configuring Your Setup Project

Windows Installer setup projects are highly configurable. You can specify locations for files on target computers, create file associations, check for preexisting conditions, or execute custom actions. In this lesson, you will learn how to create a fully configured Windows Installer setup application.

After this lesson, you will be able to

- Explain how to use the Setup Project properties to provide information about your application
- Describe how to configure the directory structure of the deployed application with the File System Editor
- Explain how to create registry keys upon installation with the Registry Editor
- Explain how to create file associations with the File Types Editor
- Describe how to edit the deployment user interface with the User Interface Editor
- Explain how to create a custom action for your setup project
- Describe how to specify a launch condition for your setup project
- Explain how to use Ngen.exe to create a native image of your application on the client machine
- Describe how to verify the security policies for an installed application

Estimated lesson time: 45 minutes

You can create, compile, and distribute a setup project using only what you have learned in Lesson 1. For many applications, this will be adequate. For most applications, however, you will want to provide some degree of customization for your setup project. Visual Studio .NET provides a series of editors that allow you to set installation properties and actions that can be taken by the setup program. After your application has been installed, you can use command-line utilities to create a precompiled image of your application in the native image cache and to verify the security policies for an installed application.

Setting the Setup Project Properties

You use the Setup Project properties to provide information about your project and set actions relating to versions of your project. The Setup Project properties can be set in the Properties window. Note that this is the Properties window in the IDE, not the Properties page that was covered in the last lesson. Many of these properties can provide descriptive information about your application. These properties include

- **AddRemoveProgramsIcon.** Allows you to specify an icon for the Add/Remove Programs dialog box on the client computer.
- **Author.** Contains information about the author of the program.
- **Description.** Contains a description of the application.
- **Keywords.** Contains keywords to be associated with the application.
- **Localization.** Provides the locale information for the application.
- **Manufacturer.** Contains information about the manufacturer of the application.

This document is created with the unregistered version of CHM2PDF Pilot

- **ManufacturerURL.** Contains the URL of the manufacturer's Web site.
- **ProductName.** Contains the name of the product.
- **Subject.** Contains information about the subject of the application.
- **SupportPhone.** Provides a phone number for support for the application.
- **SupportURL.** Contains a URL for support for the application.
- **Title.** Contains the title of the application.

Other properties of the setup project are used to determine the behavior of the setup project at install time. These properties include

- **DetectNewerInstall.** When true, the setup project looks for a more recent version of the application on the target computer and aborts the installation if one is found.
- **RemovePreviousVersion.** When true, the setup project looks for earlier versions of the application and uninstalls them in favor of the new version if found.
- **Version.** This property holds the information used by the previous two properties to determine versioning.

There are two additional properties: ProductCode and UpgradeCode. These are used by the setup program and should never be altered manually.

These properties are editable in the Properties window at design time. You can also set properties for the individual files in your setup project.

Registering Components at Installation

If your setup project includes a Font, COM component, or any other component that requires registration, you can register it upon installation by setting the *Register* property of the file in the Properties window. The possible settings for this property are

- **vsdrpDoNotRegister.** Indicates that this object requires no registration.
- **vsdrpCOM.** Indicates that this object will be registered as a COM object.
- **vsdrpCOMRelativePath.** Indicates that this object will be registered as an isolated COM object.
- **vsdrpCOMSelfReg.** Indicates that this item will be self-registered as a COM object when installed.
- **vsdrpFont.** Indicates that this item will be registered as a Font upon installation.

For .NET assemblies and components, no registration is required, and this property should be set to vsdrpDoNotRegister. Fonts should be set to vsdrpFont, and COM files with OleSelfRegister in their version information will default to vsdrpCOMSelfReg.

To register a component upon installation

In Solution Explorer, select your component. In the Properties window, set the *Register* property to the appropriate value.

The Installation Editors

Visual Studio .NET provides six installation editors that allow you to impart a high degree of configuration to your setup project. The editors are

- **File System Editor.** Allows you to configure the installation of your application to the file system of the target computer.
- **Registry Editor.** Allows you to write entries to the registry upon installation.
- **File Types Editor.** Allows you to set associations between applications and file types.

This document is created with the unregistered version of CHM2PDF Pilot

- **User Interface Editor.** Allows you to edit the user interface seen during installation for both regular installation and administrative installation.
- **Custom Actions Editor.** Allows you to define custom actions to be performed during installation.
- **Launch Conditions Editor.** Allows you to set conditions for launching the installation of your setup project.

Each of these editors is discussed in further detail in the following sections.

File System Editor

The File System Editor is used to manipulate the file system on the target computer. You can add output files to various directories, create new directories on the target machine, or create and add shortcuts to the target machine. You can access the File System Editor by selecting your project in Solution Explorer and choosing View, Editor, File System Editor.

The File System Editor window is split into two panes, as shown in Figure 10.8.

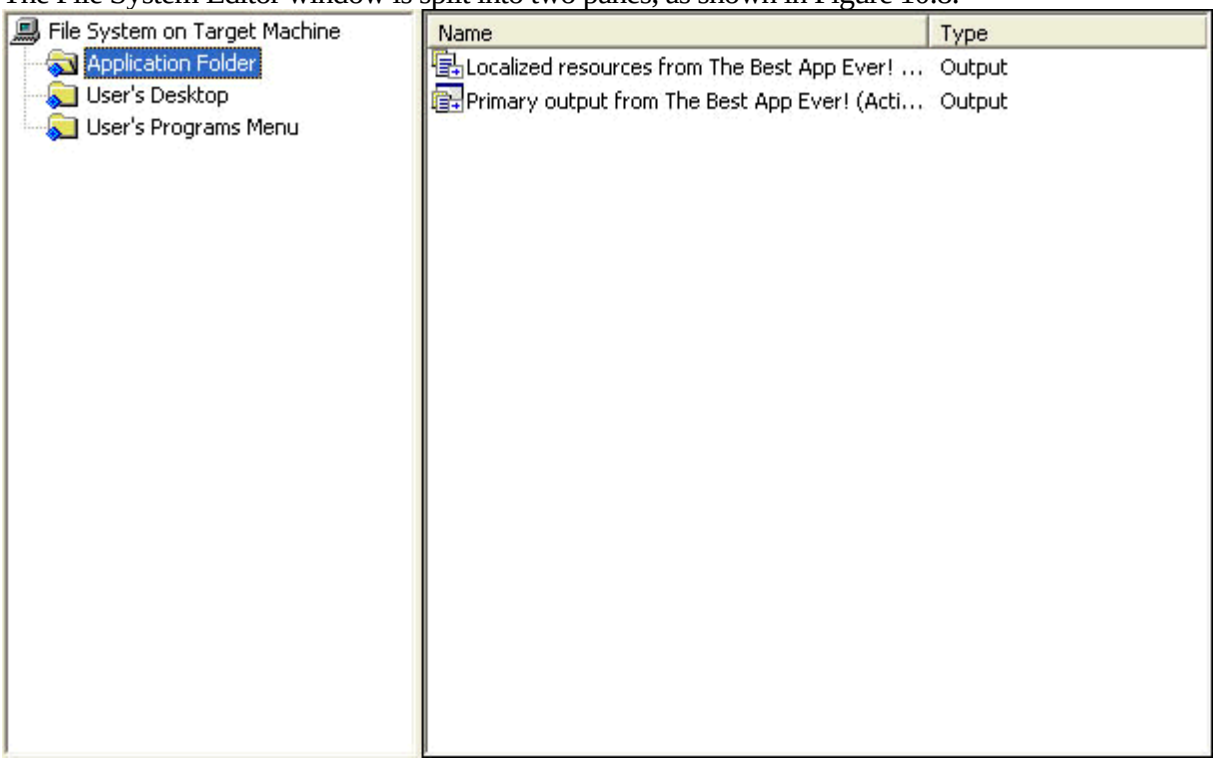
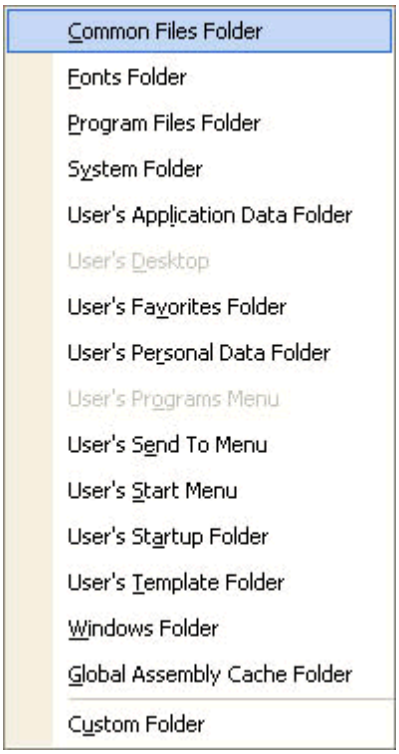


Figure 10.8 *The File System Editor*

The right pane of the File System Editor lists all of the output files in your setup project. The left pane represents the directory structure of the target computer. Initially, the File System Editor consists of three folders: the Application Folder, the User's Desktop, and the User's Program Menu. By default, the output files are initially stored in the Application Folder. You can change the folder for a particular file by selecting the file in the right pane and dragging it to the appropriate folder.

You can add additional folders to the File System Editor by right-clicking the left pane and choosing Add Special

This document is created with the unregistered version of CHM2PDF Pilot



Folder. The shortcut menu pictured in Figure 10.9 appears.

Figure 10.9 *The Special Folders shortcut menu*

Using this menu, you can add a special folder to the File System Editor or create your own custom folder. If you choose a custom folder, this folder will be created in the target computer's File System upon installation.

Adding a Loose File to Your Setup Application Using the File System Editor

You can use the File System Editor to add loose files and assemblies to your setup project. These files will be incorporated into the setup project and installed upon execution of the setup project, but will not be compressed and packaged with the rest of the project output.

To add a loose file or assembly to your setup project

1. If the folder you want your file to be added to is not present in the left pane of the File System Editor, right-click the left pane and choose the appropriate folder from the shortcut menu. You can also create subfolders under existing folders by right-clicking on an existing folder and choosingAdd, then Folder.
2. In the left pane of the File System Editor, right-click the folder you want the file to be added to upon installation and choose File. If you are adding an assembly, choose Assembly.
3. Browse to the item you want to add and select it. Click OK (for assemblies) or Open (for files).

   The file or assembly is added to your setup project and will be installed to the specified folder at installation.

Installing an Assembly to the Global Assembly Cache upon Installation

You can use the File System Editor to install an assembly to the Global Assembly Cache upon installation. To do so, the assembly must be incorporated in the setup project as a loose file and not compressed with the other project output. To install the assembly to the Global Assembly Cache, you must first sign your assembly with a strong name. See Chapter 9 for more information on strongly naming your assembly. Once your assembly has been strongly named, all you need to do is add a Global Assembly Cache folder to the File System Editor and add your assembly to that folder. Upon installation, your assembly will be added to the Global Assembly Cache.

To install an assembly to the Global Assembly Cache on installation

This document is created with the unregistered version of CHM2PDF Pilot

1. Sign your assembly with a strong name. For more information, see Chapter 9.
2. After creating your setup project, open the File System Editor.
3. Right-click the left pane and choose Add Special Folder. In the shortcut menu, choose the Global Assembly Cache Folder.
4. In the left pane, right-click the Global Assembly Cache Folder. Choose Add, and then choose Assembly from the shortcut menu. Choose the appropriate assembly or browse to its location and click OK to add the assembly.

   The assembly will be added to the Global Assembly Cache upon installation. If the assembly was not already a part of your setup project, it is added to it.

Adding a Shortcut Using the File System Editor

You might want to configure your setup application to add a shortcut to the installed application on the target machine's desktop or another location. You can use the File System Editor to create shortcuts to files and add them to folders on the client machine.

To create a shortcut and add it to the target computer

1. In Solution Explorer, right-click the file you want to create a shortcut to. Choose Find In Editor.

   The right pane of the File System Editor switches to the folder that contains that file.
2. In the right pane of the File System Editor, right-click the file you want to create a shortcut to and choose Create Shortcut.

   A shortcut to the file is created and added to the pane.
3. Drag the shortcut from the right pane to the appropriate folder in the left pane.

Registry Editor

You can use the Registry Editor to write registry entries upon installation. You can either create new registry key entries or write new values to existing registry key entries. The Registry Editor is shown in Figure 10.10.

This document is created with the unregistered version of CHM2PDF Pilot

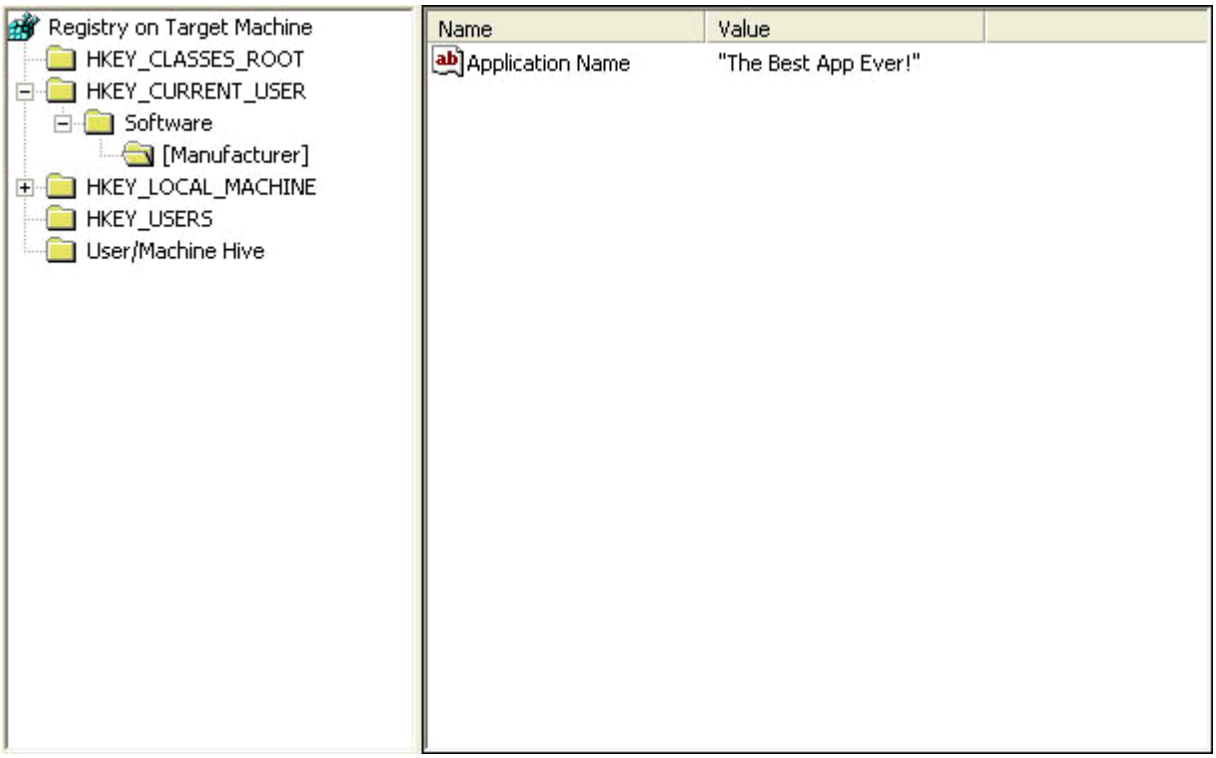


Figure 10.10 *The Registry Editor*

Like the File System Editor, the Registry Editor window is divided into two panes. The left pane represents the registry on the target computer, and the right pane represents registry key values.

To add a new registry key

Right-click the key that you want to contain the new registry key and choose New Key. A new subkey is added to that registry key. Name the key appropriately.

To add a new registry key value

1. In the left pane, right-click the registry key you want to add a value to, choose New, and then choose String Value, Environment String Value, Binary Value, or DWORD Value, depending on the type of key value you want to add.

   A new key value entry is added to the right pane of the Registry Editor.
2. In the Properties window, set the *Value* property of the new key value to an appropriate value.

File Types Editor

The File Types Editor allows you to create associations between files with a particular extension and your application. An example of a file association might be the association between files with the .txt extension and the Notepad application. When a .txt file is double-clicked, it is automatically opened with Notepad. The File Types Editor (shown in Figure 10.11) allows you to create these associations.

This document is created with the unregistered version of CHM2PDF Pilot

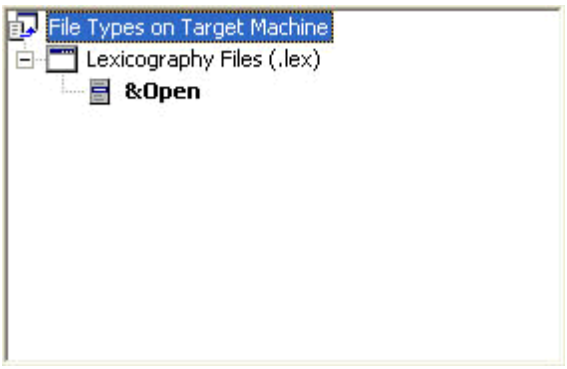


Figure 10.11 *The File Types Editor*

You can create a new file association by right-clicking the File Types Editor and choosing Add File Type. This causes a new blank file type to be added to the window. Beneath the new file type, *&Open* is added as an action. You can add additional actions by right-clicking the file type and choosing Add Action.

You can configure the file association in the Properties window for the file type and the action. The *Name* property of the file association is the name of the file types that it represents. You can add file types to this file association by setting the *Extensions* property to the extension of the file type you want to add. If you want to associate more than one file type with your application, you can add multiple file types separated by semicolons to this property. It is not necessary to prefix file extensions with a period.

The *Command* property for the association represents the application to launch when an action is taken. For example, if you are creating an association between your application and a file with the extension .tew, set the *Command* property to your application. Whenever an action is invoked on a .tew file, your application launches and performs the appropriate action.

Actions associated with a file type have three properties. The *Name* property is the text that appears in the context menu when a file with this extension is right-clicked. The *Verb* property specifies the verb that is used to invoke an action for the file type. The *Arguments* property allows you to supply any required command-line arguments to the application.

When a document that has a file association with your application is right-clicked, a context menu appears presenting the *Name* property of any defined actions. When one of the actions is chosen, the executable represented by the action is launched. The executable receives the path of the chosen document and any other parameters specified in the *Arguments* property as command-line parameters. The path of the document is always the first member of the argument array. These arguments are passed to the *Main* method of the application. You must add any appropriate program logic to cause the application to perform the desired task. The following example demonstrates how you would use the command-line argument to open a file with an application's *Open* command:

Visual Basic .NET

This document is created with the unregistered version of CHM2PDF Pilot

Lab 10: Creating an Installer Application

In this lab, you will create a setup project for the application you created in Chapters 8 and 9. You will use the Setup Project wizard to create a setup project and configure the project with the Property Pages. You will add a shortcut to the desktop of the target computer and add a launch condition to verify that Windows 2000 or a later version is the operating system. In addition, you will build your setup project and install your application to your computer. Once installed, you will verify the security permissions for your application. The solution to this lab is available in the \Labs\Ch10\Solution folder on the Supplemental Course Materials CD-ROM.

Before You Begin

Before you begin this lab, you must have either completed the Chapter 8 and Chapter 9 labs, or loaded the Chapter 10 partial solution from the \Labs\Ch10\Partial folder on the CD-ROM.

Estimated lab time: 30 minutes

Exercise 10.1: Creating the Setup Project

In this exercise, you will use the Setup Project wizard to create a setup project for your application.

To create the setup project

1. From the File menu, choose Add Project, and then choose New Project.

   The Add New Project dialog box opens.
2. In the left pane of the Add New Project dialog box, choose Setup And Deployment Projects. In the right pane, choose Setup Wizard. Name your project **InternationalSales**, and click OK to continue.
3. In the Setup wizard, click Next to go to the Choose A Project Type page.
4. Choose Create A Setup For A Windows Application. Click Next to go to the Choose Project Outputs To Include page.
5. Choose Primary Output, Localized Output, and Content Files for inclusion in the setup project. Click Next to

This document is created with the unregistered version of CHM2PDF Pilot

continue.

6. Click Next in the fourth step, and click Finish to create your setup project.
7. In Solution Explorer, right-click InternationalSales and choose Properties.

   The InternationalSales Property Page opens.
8. In the Bootstrapper drop-down menu, choose None to skip creating a bootstrapper application.
   If you are using a Windows 2000 machine, you should choose a Windows bootstrapper application instead of None.
9. Click OK to close the Property page.
10. From the File menu, choose Save All to save your work.

Exercise 10.2: Configuring Your Application

In this exercise, you will create a shortcut for your application and add a launch condition to verify that the target machine is running Windows 2000 or a later version.

To add a shortcut to your application

1. From the View menu, choose Editors, File System to open the File System Editor.
2. In the left pane of the File System Editor, click Application Folder to open the Application Folder. The output from your setup project should appear in this folder, in the right pane of the File System Editor.
3. In the right pane, right-click Primary Output and choose Create Shortcut. A new shortcut is created. Name the shortcut **International Sales**.
4. Drag the shortcut to the User's Desktop Folder.
5. From the View menu, choose Editors, Launch Conditions Editor to open the Launch Conditions Editor.
6. Right-click Launch Conditions, and choose Add Launch Condition to add a new launch condition. Name the Launch Condition **Windows2000**.
7. In the Properties window, set the Condition property to **VersionNT>=500** to specify Windows 2000 or later. Set the Message property to **You must be running Windows 2000 or later to install this application**.
8. From the Build menu, choose Build Solution to build your solution.
9. From the File menu, choose Save all.
10. Close Visual Studio .NET

Exercise 10.3: Installing Your Application

In this exercise, you will install your application to your computer. You will test the shortcut added by your setup application, and you will verify the security policy for your assembly.

To install your application

This document is created with the unregistered version of CHM2PDF Pilot

1. In Windows, browse to the folder containing your setup project. It should be in the Debug folder of your Setup Project folder.
2. Double-click the InternationalSales icon, and follow the directions provided by the Installer.
3. After the Installer has finished, locate the shortcut named International Sales on your desktop. Double-click the shortcut to open your application.
4. Close your application.
5. Open the Visual Studio .NET Command Prompt. Use the DOS cd command to change the current directory to the directory your application is installed in.
6. Use the Permview.exe utility to view the security policy for your assembly. Use the /DECL flag to view all security declarations in the assembly. An example follows:

This document is created with the unregistered version of CHM2PDF Pilot

Review

The following questions are intended to reinforce key concepts and information presented in this chapter. If you are unable to answer a question, return to the appropriate lesson to review, and then try the question again. Answers to the questions can be found in Appendix A.

1. Describe XCOPY deployment. Under what conditions is it useful? When can it not be used?
2. You have created an application for a client who wants to distribute your application to his workforce via a network share. He wants to ensure that everyone who downloads the application will download it to the same folder. Describe a plan that would accomplish this goal.
3. You have written documentation for your application and have provided it in the form of several HTML files. Describe two ways you could include this content in your setup project.
4. What is a native image? How do you create one?
5. What is the purpose of a bootstrapper application? When do you not need to create one?

This document is created with the unregistered version of CHM2PDF Pilot

Appendix A

Questions and Answers

Chapter 1 Introduction to the .NET Framework

Review Questions

1. Briefly describe the major components of the .NET Framework and describe what each component does.

   The .NET Framework consists of two primary parts: the common language runtime, which manages application execution, enforces type safety, and manages memory reclamation, and the .NET base class library, which consists of thousands of predeveloped classes that can be used to build applications.
2. Briefly explain what is meant by a reference type and a value type.

   A value type holds all of the data represented by the variable within the variable itself. A reference type contains a reference to a memory address that holds the data instead of the actual data itself.
3. How do you enable your application to use .NET base class library members without referencing their fully qualified names?

   Use the *Imports* keyword (Visual Basic .NET) or the *using* keyword (Visual C#) to make a .NET Framework namespace visible to your application.
4. Briefly describe how garbage collection works.

   The garbage collector is a thread that runs in the background of managed .NET applications. It constantly traces the reference tree and attempts to find objects that are no longer referenced. When a nonreferenced object is found, its memory is reclaimed for later use.
5. Briefly describe what members are, and list the four types of members.

   Members are the parts of a class or a structure that hold data or implement functionality. The primary member types are fields, properties, methods, and events.
6. Explain what constructors and destructors are and describe what they are used for.

   The constructor is the method that initializes a class or structure and is run when a type is first instantiated. It is used to set default values and perform other tasks required by the class. A destructor is the method that is run as the object is being reclaimed by garbage collection. It contains any code that is required for cleanup of the object.
7. Briefly explain the difference between *Public (public)*, *Friend (internal)*, and *Private (private)* access levels as they apply to user-defined types and members.

This document is created with the unregistered version of CHM2PDF Pilot

In user-defined types, Public (*public*) classes can be instantiated by any element of the application. Friend (*internal*) classes can be instantiated only by members of the same assembly, and Private (*private*) classes can be instantiated only by themselves or types they are nested in. Likewise, a Public (*public*) member can be accessed by any client in the applica-tion, a Friend (*internal*) member can be accessed only from members of the same assembly, and Private (*private*) members can be accessed only from within the type.

8. Do you need to instantiate a class before accessing a *Shared* (*static*) member? Why or why not?

Because a Shared (*static*) member belongs to the type rather than to any instance of the type, you can access the member without first creating an instance of the type.

This document is created with the unregistered version of CHM2PDF Pilot

Appendix A

Questions and Answers

Chapter 1 Introduction to the .NET Framework

Review Questions

1. Briefly describe the major components of the .NET Framework and describe what each component does.

   The .NET Framework consists of two primary parts: the common language runtime, which manages application execution, enforces type safety, and manages memory reclamation, and the .NET base class library, which consists of thousands of predeveloped classes that can be used to build applications.
2. Briefly explain what is meant by a reference type and a value type.

   A value type holds all of the data represented by the variable within the variable itself. A reference type contains a reference to a memory address that holds the data instead of the actual data itself.
3. How do you enable your application to use .NET base class library members without referencing their fully qualified names?

   Use the *Imports* keyword (Visual Basic .NET) or the *using* keyword (Visual C#) to make a .NET Framework namespace visible to your application.
4. Briefly describe how garbage collection works.

   The garbage collector is a thread that runs in the background of managed .NET applications. It constantly traces the reference tree and attempts to find objects that are no longer referenced. When a nonreferenced object is found, its memory is reclaimed for later use.
5. Briefly describe what members are, and list the four types of members.

   Members are the parts of a class or a structure that hold data or implement functionality. The primary member types are fields, properties, methods, and events.
6. Explain what constructors and destructors are and describe what they are used for.

   The constructor is the method that initializes a class or structure and is run when a type is first instantiated. It is used to set default values and perform other tasks required by the class. A destructor is the method that is run as the object is being reclaimed by garbage collection. It contains any code that is required for cleanup of the object.
7. Briefly explain the difference between *Public (public)*, *Friend (internal)*, and *Private (private)* access levels as they apply to user-defined types and members.

This document is created with the unregistered version of CHM2PDF Pilot

In user-defined types, Public (*public*) classes can be instantiated by any element of the application. Friend (*internal*) classes can be instantiated only by members of the same assembly, and Private (*private*) classes can be instantiated only by themselves or types they are nested in. Likewise, a Public (*public*) member can be accessed by any client in the applica-tion, a Friend (*internal*) member can be accessed only from members of the same assembly, and Private (*private*) members can be accessed only from within the type.

8. Do you need to instantiate a class before accessing a *Shared* (*static*) member? Why or why not?

Because a Shared (*static*) member belongs to the type rather than to any instance of the type, you can access the member without first creating an instance of the type.

This document is created with the unregistered version of CHM2PDF Pilot

Chapter 2 Creating the User Interface

Review Questions

1. You are creating an application for a major bank. The application should integrate seamlessly with Microsoft Office XP, be easy to learn, and instill a sense of corporate pride in the users. Name two ways you might approach these goals in the user interface.

   Design the user interface to mimic the look and feel of Microsoft Office XP, which will allow users of Office XP to immediately feel comfortable with the new application. Integration of the corporate logo and other visual elements associated with the company will aid in identification of the program with the company.
2. You are writing an application that needs to display a common set of controls on several different forms. What is the fastest way to approach this problem?

   Create a single form that incorporates the common controls, and use visual inheritance to create derived forms.
3. If you wanted to prompt a user for input every time a form received the focus, what would be the best strategy for implementing this functionality?

   Write an event handler for the *Activated* event that implements the relevant functionality.
4. Describe two ways to set the tab order of controls on your form.

   You can set the tab order in Visual Studio by choosing Tab Index from the View menu and clicking each control in the order you desire. Alternatively, you can set the *TabIndex* property either in code or in the Properties window.
5. What is an extender provider, and what does one do?

   Extender providers are components that provide additional properties to controls on a form. Examples include the *ErrorProvider*, *HelpProvider*, and *ToolTip* components. They can be used to provide additional information about particular controls to the user in the user interface.
6. Explain when you might implement a shortcut menu instead of a main menu.

   If every possible option is exposed on a main menu, the menu can become busy and hard to use. Shortcut menus allow less frequently used options to be exposed only in situations where they are likely to be used.
7. Describe what is meant by field-level validation and form-level validation.

   Field-level validation is the process of validating each individual field as it is entered into a form. Form-level validation describes the process of validating all of the data on a form before submitting the form.

This document is created with the unregistered version of CHM2PDF Pilot

Chapter 3 Types and Members

Review Questions

1. Explain when a type conversion will undergo an implicit cast and when you must perform an explicit cast. What are the dangers associated with explicit casts?

   Types can be implicitly converted when the conversion can always take place without any potential loss of data. When a potential loss of data is possible, an explicit cast is required. If an explicit cast is improperly performed, a loss of data precision can result, or an exception can be thrown.
2. Explain why you might use enums and constants instead of their associated literal values.

   Enums and constants make code easier to read and maintain by substituting human-legible tokens for frequently used constant values.
3. Briefly summarize the similarities and differences between arrays and collections.

   Arrays and collections allow you to manage groups of objects. You can access a particular object by index in both arrays and collections, and you can use *For Each...Next* (*foreach*) syntax to iterate through the members of arrays and most collections. Arrays are fixed in length, and members must be initialized before use. Members of collections must be declared and initialized outside of the collection, and then added to the collection. Collections provided in the *System.Collections* namespace can grow or shrink dynamically, and items can be added or removed at run time.
4. Explain how properties differ from fields. Why would you expose public data through properties instead of fields?

   Properties allow validation code to execute when values are accessed or changed. This allows you to impose some measure of control over when and how values are read or changed. Fields cannot perform validation when being read or set.
5. Explain what a delegate is and how one works.

   A delegate acts like a strongly typed function pointer. Delegates can invoke the methods that they reference without making explicit calls to those methods.

This document is created with the unregistered version of CHM2PDF Pilot

Chapter 4 Object-Oriented Programming and Polymorphism

Review Questions

1. Briefly explain encapsulation and why it is important in object-oriented programming.

   Encapsulation is the principle that all of the data and functionality required by an object be contained by that object. This allows objects to exist as independent, interchangeable units of functionality without maintaining dependencies on other units of code.
2. What is method overloading, and when is it useful?

   Method overloading allows you to create several methods with the same name but different signatures. Overloading is useful when you want to provide the same or similar functionality to different sets of parameters.
3. You need to create several unrelated classes that each exposes a common set of methods. Briefly outline a strategy that will allow these classes to polymorphically expose that functionality to other classes.

   Factor the common set of methods into an interface, and then implement that interface in each class. Each class can then be implicitly cast to that interface and can polymorphically interact with other classes.
4. You need to create several classes that provide a core set of functionality but each must be able to interact with a different set of objects. Outline a strategy for developing these classes with the least development time.

   Create a single class that implements all of the common functionality required by these classes. Then, use inheritance to create derived classes that are specific for each individual case.
5. Describe an abstract class and explain when one might be useful.

   An abstract class is a class that cannot be instantiated but must be inherited. It can contain both implemented methods and abstract methods, which must be implemented in an inheriting class. Thus, it can define common functionality for some methods, a common interface for other methods, and leave more detailed implementation up to the inheriting class.

This document is created with the unregistered version of CHM2PDF Pilot

Chapter 5 Testing and Debugging Your Application

Review Questions

1. Describe Break mode and some of the available methods for navigating in Break mode.

   Break mode allows you to observe program execution on a line-by-line basis. You can navigate program execution in Break mode by using Step Into, Step Over, Step Out, Run To Cursor, and Set Next Statement.
2. When would you use the Watch window?

   You would use the Watch window to observe the values of application variables while in Break mode.
3. You are deploying a beta version of a large application and want to collect performance data in text files while the application is in use. Briefly describe a strategy for enabling this scenario.

   Place Trace statements that report the data of interest throughout the application. Create a *TextWriterTraceListener* and add it to the Listeners collection. Create *Trace* switches that control when *Trace* statements are executed. Configure the *TextWriterTraceListener* to write output to a text file. Then, compile and deploy the application with *Trace* defined, and enable the *Trace* switches in the application .config file.
4. When testing a method, should you test data that is known to be outside of the bounds of normal operation? Why or why not?

   Yes. In addition to testing normal data operation, you must test known bad input to ensure that your application can recover from input errors without a catastrophic application failure.
5. Briefly explain what each segment of a *Try*...*Catch*...*Finally* (*try*...*catch*...*finally*) block does.

   The *Try* (*try*) block encloses code that is to be executed. If an exception is thrown, it can be caught in an appropriate *Catch* (*catch*) block where code that will allow the application to handle the execution will be executed. The *Finally* (*finally*) block contains any code that must be executed whether or not the exception is handled.

This document is created with the unregistered version of CHM2PDF Pilot

Chapter 6 Data Access with ADO.NET

Review Questions

1. What are the major components of a Data Provider, and what function does each fulfill?

   An ADO.NET Data Provider is a suite of components designed to facilitate data access. Every Data Provider minimally includes a *Connection* object that provides the actual connection to the data source, a Command object that represents a direct command to the data source, a *DataReader* object that provides connected, forward-only, read-only access to a database, and a *DataAdapter* that facilitates disconnected data access.
2. Briefly contrast connected and disconnected data access in ADO.NET.

   In ADO.NET, connected data access is available through the *DataReader*, which is a lightweight class designed to provide very fast and efficient data access. It is severely limited, however, in that it can only provide forward-only data access, it does not allow editing, and it requires the exclusive use of a *Connection* object. In contrast, disconnected data access is facilitated by a *DataAdapter*, which manages the commands required for selecting and updating data. The *DataAdapter* executes a SELECT command against a database, opening a data connection just long enough to retrieve the data, and loads the data into a *DataSet*, which is an in-memory copy of the data. When the data is ready to be updated, the Data Provider manages the updates in the same way, generating the appropriate commands to update the database and keeping the connection open just long enough to execute those commands.
3. What are the three possible settings for the *CommandType* property of a *SqlCommand* object or an *OleDbCommand* object, and what does each mean?

   A *Command* object can have a *CommandType* property setting of Text, StoredProcedure, or TableDirect. When set to Text, the command executes the SQL string that is stored in the *Command* object's *CommandText* property. When set to StoredProcedure, the command accesses a procedure stored on the database and returns the results. A *CommandText* setting of TableDirect indicates that the command should return the entire contents of the table indicated by the *CommandText* property.
4. How could you execute DDL commands, such as ALTER or CREATE TABLE, against a database with ADO.NET?

   You must use a *Command* object to execute DDL commands. You can set the *CommandType* property to Text and enter the appropriate DDL command in the CommandText property. Then call *Command.ExecuteNonQuery* to execute the command.
5. Briefly discuss the advantages and disadvantages of using typed *DataSet* objects.

   Typed *DataSet* objects allow you to work with data that is represented as members of the .NET common type system. This allows your applications to be aware of the types of data returned in a *DataSet* and serves to eliminate errors resulting from invalid casts, as any type mismatches are caught at compile time. Untyped *DataSet* objects, however, are useful if you do not know the structure of your data, and can be used with any data source.

This document is created with the unregistered version of CHM2PDF Pilot

6. How can you manage data currency on a form with several bound controls?

   Every data source on a form has an associated *CurrencyManager* object that keeps that of the "current" record with respect to bound controls. For convenience, all of the *CurrencyManager* objects represented on a form are exposed through the form's *BindingContext* property. The *Position* of the *CurrencyManager* can be changed, allowing navigation through the records.
7. Describe how to use a *DataView* to filter or sort data.

   You can apply sort criteria to a *DataView* by setting the Sort property to the name of a column or columns to be sorted by. The data represented in a *DataView* object can be filtered by setting the *RowFilter* property to a valid filter expression.
8. Briefly describe an *XmlDataDocument* and how it relates to a *DataSet*.

   An *XmlDataDocument* is an in-memory representation of data in a hierarchical Xml format. Each *XmlDataDocument* is synchronized with a DataSet. Whenever changes are made to one object, the other is instantly updated. Thus, you can use the *XmlDataDocument* to perform XML manipulations on a *DataSet*.

This document is created with the unregistered version of CHM2PDF Pilot

Chapter 7 Creating Controls Using the .NET Framework

Review Questions

1. Briefly describe the three types of user-developed controls and how they differ.

   The three types of user-developed controls are inherited controls, user controls, and custom controls. An inherited control derives from a standard Windows Forms control and inherits the look, feel, and functionality of that control. User controls allow you to combine standard Windows Forms controls and bind them together with common functionality. Custom controls inherit from *Control* and are the most development-intensive kind of control. Custom controls must implement all their own code for painting and inherit only generic control functionality. All specific functionality must be implemented by the developer.
2. Describe the roles of *Graphics*, *Brush*, *Pen*, and *GraphicsPath* objects in graphics rendering.

   The *Graphics* object represents a drawing surface and encapsulates methods that allow graphics to be rendered to that surface. A *Brush* is an object that is used to fill solid shapes, and a *Pen* is used to render lines. A *GraphicsPath* object represents a complex shape that can be rendered by a *Graphics* object.
3. Describe the general procedure for rendering text to a drawing surface.

   You must first obtain a reference to a *Graphics* object. Next, create an instance of a *GraphicsPath* object. Use the *GraphicsPath.AddString* method to add text to the *GraphicsPath*. Then, call the *Graphics.DrawPath* or *Graphics.FillPath* to render the text.
4. Describe the role of the *LicenseProvider* in control licensing.

   The *LicenseProvider* controls license validation and grants run-time licenses to validly licensed components. The *LicenseManager.Validate* method checks for an available license file and checks against the validation logic provided by the specific implementation of *LicenseProvider*. You specify which *LicenseProvider* to use by applying the *LicenseProviderAttribute*.
5. Describe how to create a form or control with a nonrectangular shape.

   Set the *Region* property of the form or control to a *Region* object that contains the irregular shape. You can create a *Region* object from a *GraphicsPath* object.

This document is created with the unregistered version of CHM2PDF Pilot

Chapter 8 Advanced .NET Framework Topics

Review Questions

1. Briefly describe how to use the *PrintDocument* component to print a document. Discuss maintaining correct line spacing and multipage documents.

   The *PrintDocument* class exposes the *Print* method, which raises the *PrintPage* event. Code to render printed items to the printer should be placed in the *PrintPage* event handler. The *PrintPage* event handler provides the objects required to render to the printer in an instance of the *PagePrintEventArgs* class. Content is rendered to the printer using the Graphics object provided by *PagePrintEventArgs*. You can calculate correct line spacing by dividing the height of the *MarginBounds* property by the height of the font you are rendering. If your document has multiple pages, you must set the *PagePrintEventArgs.HasMorePages* property to true, which causes the *PrintPage* event to fire again. Because the *PrintPage* event handler retains no inherent memory of how many pages have been printed, you must incorporate all logic for printing multiple pages into your event handler.
2. Explain how to use the *Begin* and *End* methods on a Web Service to make an asynchronous method call.

   Every public Web method on a Web Service can be called either synchronously or asynchronously. To make an asynchronous call to a Web method, you call the method named Begin<*webmethod*>, where <*webmethod*> is the name of the method. This method requires a delegate to an appropriate callback method and returns a value of IAsyncResult. This value is returned as a parameter in the callback method. To retrieve the data returned by the Web method, call *End*<*webmethod*>, supplying a reference to the IAsyncResult returned by *Begin*<*webmethod*>. This will allow you to retrieve the actual data returned by the Web method.
3. Briefly describe the five accessibility requirements of the Certified for Windows logo program.

   The five requirements are
   - **Support standard system settings. This requires your application to be able to conform to system settings for colors, fonts, and other UI elements.**
   - **Be compatible with High Contrast mode. This requirement can be met by using only the System palette for UI colors.**
   - **Provide documented keyboard access for all UI features. Key points in this requirement are shortcut keys and accessible documentation.**
   - **Provide notification of the focus location. This requirement is handled primarily by the .NET Framework.**
   - **Convey no information by sound alone. This requirement can be met by providing redundant means of conveying information.**
4. Explain how to use the *HelpProvider* component to provide help for UI elements.

   You can provide either a *HelpString* or a help topic for UI elements with the *HelpProvider*. The *HelpProvider* provides *HelpString*, *HelpKeyWord*, and *HelpNavigator* properties for each control on the form. If no value for the *HelpProvider.HelpNameSpace* is set, the *HelpString* will be provided as help. If the *HelpNameSpace* is set, the *HelpProvider* will display the appropriate help

This document is created with the unregistered version of CHM2PDF Pilot

topic as configured by the *HelpKeyWord* and *HelpNavigator* properties. Help for a particular element is displayed when the element has the focus and the F1 key is pressed.

5. Describe how to create localized versions of a form.

   To create a localized version of a form, set the *Localizable* property to true. Then set the *Language* property to the language/region for which you want to create the localized form. Make any localization-related changes in the UI. The changed property values will automatically be stored in resource files and loaded when the *CurrentUICulture* is set to the appropriate *CultureInfo*.

6. Explain how to convert data in legacy code page formats to the Unicode format.

   You can use the *Encoding.Convert* method to convert data between encoding types. This method requires instances of both encoding types and an array of bytes that represents the data to be converted. It returns an array of bytes in the target format. You can convert a string or array of chars to an array of bytes with the *Encoding.GetBytes* method and can convert an array of bytes back to chars with the *Encoding.GetChars* method.

7. Explain the difference between Globalization and Localization.

   Globalization refers to the application of culture-specific format to existing data. Localization refers to providing new culture-specific resources and retrieving the appropriate resources based on the culture setting.

This document is created with the unregistered version of CHM2PDF Pilot

Chapter 9 Assemblies, Configuration, and Security

Review Questions

1. Describe how to sign your assembly with a strong name. Why would you want to do this?

   To sign your assembly with a strong name, you must have access to a key file or create one with the strong name utility (sn.exe). You then specify the key file in the AssemblyInfo file and verify that the version number is correct. The assembly will be signed with a strong name when built. In addition to identifying your assembly and ensuring version identity, a strong name is required if you want to install your assembly to the Global Assembly Cache.
2. Describe how to use code to retrieve resources at run time.

   You must first create an instance of the *ResourceManager* class that is associated with the assembly that contains the desired resource. You can then use the *GetString* method to retrieve string resources or the *GetObject* method to retrieve object resources.
3. Explain how to retrieve information from the configuration file at run time. How would you store information in the configuration file at design time?

   You must first create an instance of *AppSettingsReader* to read the configuration file. You can then call the *GetValue* method to read values represented in the configuration file. To add configuration file entries, you should create <add> elements in the <appSettings> node of the configuration file. In the <add> element, you should specify a value for the entry and a key that can be used to retrieve the entry. The value can be changed between executions of the application.
4. You are creating a solution that must be accessed by members of a group called Developers and Administrators on the local machine. Describe a plan to implement this security scheme.

   Create one *PrincipalPermission* that represents the Developers group and another *PrincipalPermission* that represents the BUILTIN\Administrators group. Then, create a third permission that represents the union of the two by calling the *Union* method and demand that permission.
5. Briefly highlight the differences between imperative and declarative security as they pertain to code access security.

   Imperative security is implemented by calling methods of *Permission* objects in code at run time. Declarative security is configured by attaching attributes representing permissions to classes and methods. Imperative security allows a finer control over the point in execution where permissions are demanded, but declarative security is emitted into metadata, and required permissions can be discovered through the classes in the *System.Reflection* namespace. Additionally, you can request assembly-wide permissions using the *Assembly* (*assembly*) directive with declarative security.

This document is created with the unregistered version of CHM2PDF Pilot

Chapter 10 Deploying Your Application

Review Questions

1. Describe XCOPY deployment. Under what conditions is it useful? When can it not be used?

   XCOPY deployment is a simple method of deployment where the DOS command XCOPY is used to copy the application directory and any subdirectories to the target machine. You can use XCOPY deployment if your application has no dependency on shared files and requires no special actions to be taken upon deployment. If an application requires a more complex deployment or references shared assemblies, you cannot use XCOPY.
2. You have created an application for a client who wants to distribute your application to his workforce via a network share. He wants to ensure that everyone who downloads the application will download it to the same folder. Describe a plan that would accomplish this goal.

   Create a setup project for the application. Using the User Interface Editor, provide a dialog box that allows the file directory to be set during administrative installation, but removes this box from regular installation. Use administrative installation to install the application to the network share.
3. You have written documentation for your application and have provided it in the form of several HTML files. Describe two ways you could include this content in your setup project.

   You can include loose HTML files along with your application by either including them when you create the setup project with the Setup wizard or by adding them to the project after creation with the File System Editor.
4. What is a native image? How do you create one?

   A native image is a precompiled version of a .NET assembly. You can create a native image of your application by using the Ngen.exe utility.
5. What is the purpose of a bootstrapper application? When do you not need to create one?

   A bootstrapper application automatically detects if Windows Installer is installed on the target machine. If Windows Installer is not present, it installs Windows Installer before proceeding with the rest of the installation. You should create a bootstrapper application unless all of your target machines are running Windows XP (which has Microsoft Windows Installer 1.5 already installed) or have had Microsoft Installer 1.5 installed previously.

This document is created with the unregistered version of CHM2PDF Pilot

Glossary

A

abstract class

A class that cannot be instantiated but is used as a base from which other classes can be derived. In Microsoft Visual Basic .NET, abstract classes are declared using the *MustInherit* keyword. In Microsoft Visual C#, abstract classes are declared using the *abstract* keyword.

abstraction

Representing real-world objects as programmatic constructs.

abstract members

A member of a base class that cannot be invoked, but instead provides a template for members of a derived class. In Visual Basic .NET, abstract members are declared using the *MustOverride* keyword. In Visual C#, abstract members are declared using the *abstract* keyword.

ADO.NET

The data access architecture for the Microsoft .NET Framework. ADO.NET is built around a disconnected data access model that uses a set of classes called a Data Provider to retrieve data from a data source.

alias

The name you use to identify resources on the Web. Aliases represent physical resources on the Web server, such as a Web form, an HTML page, or a graphic.

anchoring

In Windows Forms controls, the resizing and repositioning of a control when its parent form is resized. You can anchor a control by setting the *Anchor* property.

This document is created with the unregistered version of CHM2PDF Pilot

assembly

The primary unit of deployment for a .NET application. An assembly is either an executable application (EXE) or a class library (DLL).

assembly manifest

The metadata for the assembly. It contains all of the information needed to describe the assembly to the common language runtime.

authentication

The process of determining the identity of a user. In effect, authentication validates that the users are who they say they are. *See also* authorization.

authorization

The process of allowing access to an application or resource based on credentials supplied by the user. *See also* authentication.

B

base class

A class that provides properties and methods as a foundation for a derived class. In objected-oriented programming, one class can be based on another through inheritance. Using this technique, the base class provides characteristics (such as properties and methods) to a derived class. The derived class can reuse, modify, or add to the members of the base class.

Bezier

A curve specified by four points. Two points specify the ends of the curve, and the other two points influence the shape of the curve between the end points.

Boolean

A data type that can represent a value of either true or false.

This document is created with the unregistered version of CHM2PDF Pilot

bootstrapper

In deployment, an application that installs Microsoft Windows Installer on machines that do not currently have it installed.

boundary condition

In testing, test conditions at the limit of normal operation and just over or under those limits.

boxing

The act of wrapping a Value type, such as an Integer, inside an object so that it can be treated like a Reference type.

Break mode

In debugging, a state of suspended execution wherein the application can be executed in a line-by-line manner, and individual program variables can be examined.

breakpoint

In debugging, a predetermined point where execution of an application will halt and enter Break mode.

C

cast

An explicit conversion from one type to another.

circular reference

A condition that occurs when two objects refer to each other but do not have an external reference. Circular references are automatically detected and disposed of by the garbage collector.

class

A user-defined Reference type that serves as a template for an object of that type.

This document is created with the unregistered version of CHM2PDF Pilot

CLR

See common language runtime (CLR).

code access security

Verification that all callers to code have appropriate permission to access it.

code tuning

The act of examining application code and optimizing for faster execution. Code tuning is usually an iterative process that involves several successive rounds of planning, improvement, and assessment of the results.

common language runtime (CLR)

The environment in which managed code executes. The common language runtime provides just-in-time compilation, enforces type safety, and manages memory through garbage collection.

Common Type System

A set of types that are used by all .NET languages, thus ensuring .NET language type compatibility.

complex binding

In data binding, binding more than one column to a single control. The *DataGrid* is an example of a control that can be complexly bound.

component

Strictly speaking, a class that implements *IComponent* or inherits *System*.*Component*. More generally, any class that provides a reusable set of functionality.

connected data access

Data access that requires a data connection that is always open.

Constant

This document is created with the unregistered version of CHM2PDF Pilot

A type that represents a value that does not change in the course of an application's execution and cannot be changed.

constraint

In ADO.NET, an object that represents a rule that enforces referential integrity, such as a constraint that specifies a primary key.

constructor

The method that executes when a class is instantiated. The constructor usually contains initialization code to set member variables to initial values.

container controls

Controls that can host other controls. Examples include *Forms*, *Panels*, and *TabPages*.

control

A class that inherits *System*.*Control*. Controls usually have a visual representation, interact with the user as a part of the user interface, and are hosted in a container control.

controls collection

The collection of a container control that manages the controls it hosts.

custom control

A user-designed control that directly inherits *System*.*Control*.

D

data binding

A way to link data (such as a data set) in your application to the properties of a control.

data consumer

This document is created with the unregistered version of CHM2PDF Pilot

A control that is bound to a data source.

Data Provider

A set of classes that work together to provide managed data access.

DataSet

A class that contains an application copy of data that has been retrieved from a data source.

delegate

An object that acts as a type-safe function pointer. Delegates can be used to invoke the method they point to.

deploying

Installing an application on the computer where it will run.

derived class

A class that is based on another class (called a base class) through inheritance. A derived class inherits the members of its base class and can override or shadow those members.

design time

Represents the state of the application while being designed. Applications at design time consist of noncompiled code that can be manipulated in the code editor or designer window.

destructor

The method that is run just before a class is removed from memory. Generally, a destructor contains any code required to "clean up" after the class. *See also* finalizer.

disconnected data access

Data access that uses a data connection only when executing database operations. This is the default kind of data access implemented by ADO.NET.

This document is created with the unregistered version of CHM2PDF Pilot

docking

In Windows Forms, attaching a control to the edge of the form. You can dock a control by setting the *Dock* property.

E

encapsulation

In component programming, separating the implementation of a component from the interface. Only the public interface of a component is made accessible to the rest of the application. Component data should never be accessible to outside callers.

event

A notification raised by a control or component that a specified condition has occurred.

event handler

A method that is executed when a particular event is raised.

exception

Any class that derives from *System.Exception*. Exceptions are thrown in response to run-time errors. *See also* throwing.

exception handling

The process of dealing with unusual occurrences within code so that it does not cause the program to crash or lose data.

explicit conversion

A conversion of one type to another that cannot be performed automatically. An explicit conversion usually presents some danger of a failed conversion or a loss of data.

extender provider

This document is created with the unregistered version of CHM2PDF Pilot

A specialized type of control that provides design time properties to other controls. *ErrorProvider* and *HelpProvider* are examples of extender providers.

F

field

A member variable of a class or structure.

finalizer

In Visual Basic .NET, the method that is called just before an instance of a class is destroyed. *See also* destructor.

floating-point numbers

A number that has a decimal component. *See also* floating-point types.

floating-point types

Types capable of representing floating-point numbers. Floating-point types include Single (float) and Double (double). *See also* floating-point numbers.

focus

When referring to a control, the condition of being able to receive user input through the keyboard or other input devices. A control that is capable of receiving input is said to have the focus.

foreign key

In a data table, a column that contains a key value that uniquely identifies a record in a related table.

form

The primary unit of the user interface in a Windows Forms application. A form serves as a nexus for data display and user input, and can host controls.

This document is created with the unregistered version of CHM2PDF Pilot

G

GAC

See Global Assembly Cache (GAC).

garbage collection

Automatic memory management provided by the common language runtime. Unused memory is automatically reclaimed by garbage collection without interaction with the application.

GDI+

See Graphic Device Interface (GDI).

getter

The code segment of a property that returns the value the property contains.

Global Assembly Cache (GAC)

A machine-wide cache that stores assemblies that have been specifically designated to be shared between several applications.

globalization

Applying culture-based formatting to existing data, for example, using culture-specific currency symbols and formatting.

Graphic Device Interface (GDI)

In Windows, a graphics display system used by applications to display or print bitmapped text (TrueType fonts), images, and other graphical elements. GDI+ is the name given to the .NET Framework's managed implementation of the GDI.

I

This document is created with the unregistered version of CHM2PDF Pilot

implicit conversion

A conversion between types that can be performed automatically by the common language runtime. An implicit conversion will always succeed and never pose a danger of data loss.

inheritance

In object-oriented programming, creating a new class that incorporates and extends all of the characteristics and functionality of a base class.

inherited control

A user-created control that inherits an existing control. An inherited control inherits all of the functionality of the base control and can incorporate custom functionality as well.

interface

A contract for behavior that can be implemented by a class. Classes that implement the same interface can be expected to provide the members defined by the interface.

L

localization

Providing multiple sets of data specific to different cultures and retrieving the data appropriate to the current culture setting.

logical error

An error that compiles and executes correctly but produces unexpected results. These can be the most difficult errors to detect in application testing.

M

This document is created with the unregistered version of CHM2PDF Pilot

managed code

Code that runs under the common language runtime. The common language runtime handles many tasks that would formerly have been handled in the application's executable. Managed code solves the Windows programming problems of component registration and versioning (sometimes called DLL Hell) because managed code contains all the versioning and type information that the common language runtime needs to run the application. The common language runtime handles registration dynamically at run time rather than statically through the system registry, as is done with applications based on the Component Object Model (COM).

metadata

Data about data. For example, the name, version, security information, and size of a file constitute metadata about the file.

Microsoft Intermediate Language (MSIL)

A low-level language that is just-in-time compiled to native code at run time. All .NET assemblies are represented in the MSIL.

MSIL

See Microsoft Intermediate Language (MSIL).

multidimensional array

An array that contains more than one dimension.

N

namespace

A logical organization of types that perform related functions.

.NET Framework

This document is created with the unregistered version of CHM2PDF Pilot

A managed environment for the development and execution of code. It consists of the common language runtime and .NET Framework base class library.

.NET Framework base class library

A collection of object-oriented types and interfaces that provides object models to assist in the development of your applications.

O

object

The base class for all types in the .NET Framework. More generically, any instance of a class or structure.

object-oriented programming

The process of creating applications by using programmatic constructs to represent real-world objects.

operator overloading

In Visual C#, defining custom behavior for an operator when interacting with custom types.

optimization

The process of examining your application and modifying it to be more efficient in terms of either speed or size of the deployed application.

overloading

Creating multiple methods with the same name but different signatures.

overriding

When inheriting from a base class, providing a new method for one of the base class methods.

P

This document is created with the unregistered version of CHM2PDF Pilot

parent-child relationship

In data access, a relationship between two tables wherein a record from the parent table can be used to retrieve one or more related records from a child table. A parent-child relationship can be defined by using a *DataRelation* object.

polymorphism

The ability of classes to provide different implementations of the same public interface. Thus, two classes that exhibit polymorphism might contain different implementations, but because the interfaces are identical, they can be treated the same in code.

primary key

In a data table, the column that uniquely identifies a record in the table.

private assembly

An assembly that is used by only one application. In .NET Framework programming, most assemblies are private assemblies.

project

The collection of Visual Studio .NET source files that make up an application.

R

Reference type

A type, such as a class, whose instance data is allocated on the heap. Variables that represent Reference types contain a reference to the object on the heap as opposed to the object's data itself. *See also* Value type.

registry

The system repository for information about a computer's settings. The registry can contain configuration information

This document is created with the unregistered version of CHM2PDF Pilot

about programs and system components.

resource

Nonexecutable data, such as a string or image, that is deployed in an application.

role-based security

Code that ensures that users of your application have appropriate permission to access your application based on the identity of the user.

run time

Represents the state of the application while executing. At run time, the application consists of executing code.

run-time error

An error that causes an invalid result or attempts an invalid operation at run time.

S

satellite assembly

When localizing an application, assemblies that contain alternate sets of resources to be used in the application for different cultures.

setter

The code segment of a property that sets the property's value.

shadowing

Hiding a member in a base class with a member in an inherited class that has a different signature.

shared assembly

This document is created with the unregistered version of CHM2PDF Pilot

An assembly that is referenced by several applications on a single machine. A shared assembly must be installed to the Global Assembly Cache (GAC).

shared members

In Visual Basic .NET, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Shared members are called static members in Visual C#.

simple binding

In data binding, binding a control to a single data column. A *TextBox* is an example of a control that can be simply bound.

solution

A group of Visual Studio .NET projects that make up a single functional unit.

SQL

See Structured Query Language (SQL).

static members

In Visual C#, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Static members are called shared members in Visual Basic .NET.

Step Into

Moving from a calling procedure to a called procedure during debugging. Also used in reference to setting breakpoints to stop execution in a specific procedure during debugging.

Step Out

In Break mode, resuming execution of a function call until the call returns.

Step Over

In Break mode, executing a procedure call as a single statement during debugging.

This document is created with the unregistered version of CHM2PDF Pilot

strong name

A name that uniquely identifies an assembly. It consists of the assembly's name, version, and culture information as well as a public key encrypted by a private key. An assembly must have a strong name to be installed to the Global Assembly Cache (GAC).

strongly typed

Refers to types for all variables being explicitly declared. Strong typing eliminates the possibility of type mismatch errors, which are discovered at compile time. Visual C# is inherently strongly typed, and Visual Basic .NET is strongly typed when Option Strict is set to On.

Struct

In Visual C#, a user-defined Value type that serves as a template for an instance of that type. *See also* Structure.

Structure

In Visual Basic .NET, a user-defined Value type that serves as a template for an instance of that type. *See also* Struct .

structured exception handling

Using *Try...Catch...Finally* blocks to handle errors that might occur during code execution. When exceptions are thrown, they are handled in the *Catch* block before code in the *Finally* block is executed.

Structured Query Language (SQL)

The standard language used for issuing commands to databases.

syntax error

An error that occurs because of typographical errors or nonsensical code. In the .NET Framework, syntax errors prevent compilation and are thus discovered at compile time.

T

target audience

This document is created with the unregistered version of CHM2PDF Pilot

The group of end users for whom an application is designed.

threads

The basic unit to which the server allocates processor time. A single process can have multiple threads.

throwing

The act of passing an exception to the next caller in the call stack. *See also* exception.

tracing

A programming technique for recording events, such as exceptions, in an application. Tracing is used during debugging and in the testing phase of application deployment.

transaction

A group of commands (treated as a single unit) that change the data stored in a database. The transaction ensures that the commands are handled in an all-or-nothing fashion if one of the commands fails, all of the commands fail, and any data that was written to the database by the commands is backed out. In this way, transactions maintain the integrity of data in a database.

tuning

The process of making adjustments to a deployed application that do not affect code.

U

Unicode

The universal character encoding scheme for characters and text. Each character is represented by a unique numeric value.

unmanaged code

Code that is not managed by the common language runtime. Unmanaged code is not checked for type-safety and

This document is created with the unregistered version of CHM2PDF Pilot

must be used with extreme care.

user control

A user-defined control that encapsulates other Windows Forms controls with custom functionality in a common container.

V

Value type

A type, such as an Integer, Decimal, or Structure, whose instance data is allocated on the stack. Variables that represent Value types contain the object's instance data itself as opposed to a reference to that data. *See also* Reference type.

W

Watch window

When debugging, the window that allows you to examine the values of predetermined watch variables.

Web Service

A service or component that is implemented by a Web server and can be instantiated and accessed programmatically as a component over the World Wide Web.

X

XML Schema

A description of the data elements contained in an XML file. The XML Schema provides the names of the elements, their types, whether or not they are key fields, and other information.

This document is created with the unregistered version of CHM2PDF Pilot

Glossary

A

abstract class

A class that cannot be instantiated but is used as a base from which other classes can be derived. In Microsoft Visual Basic .NET, abstract classes are declared using the *MustInherit* keyword. In Microsoft Visual C#, abstract classes are declared using the *abstract* keyword.

abstraction

Representing real-world objects as programmatic constructs.

abstract members

A member of a base class that cannot be invoked, but instead provides a template for members of a derived class. In Visual Basic .NET, abstract members are declared using the *MustOverride* keyword. In Visual C#, abstract members are declared using the *abstract* keyword.

ADO.NET

The data access architecture for the Microsoft .NET Framework. ADO.NET is built around a disconnected data access model that uses a set of classes called a Data Provider to retrieve data from a data source.

alias

The name you use to identify resources on the Web. Aliases represent physical resources on the Web server, such as a Web form, an HTML page, or a graphic.

anchoring

In Windows Forms controls, the resizing and repositioning of a control when its parent form is resized. You can anchor a control by setting the *Anchor* property.

This document is created with the unregistered version of CHM2PDF Pilot

assembly

The primary unit of deployment for a .NET application. An assembly is either an executable application (EXE) or a class library (DLL).

assembly manifest

The metadata for the assembly. It contains all of the information needed to describe the assembly to the common language runtime.

authentication

The process of determining the identity of a user. In effect, authentication validates that the users are who they say they are. *See also* authorization.

authorization

The process of allowing access to an application or resource based on credentials supplied by the user. *See also* authentication.

B

base class

A class that provides properties and methods as a foundation for a derived class. In objected-oriented programming, one class can be based on another through inheritance. Using this technique, the base class provides characteristics (such as properties and methods) to a derived class. The derived class can reuse, modify, or add to the members of the base class.

Bezier

A curve specified by four points. Two points specify the ends of the curve, and the other two points influence the shape of the curve between the end points.

Boolean

A data type that can represent a value of either true or false.

This document is created with the unregistered version of CHM2PDF Pilot

bootstrapper

In deployment, an application that installs Microsoft Windows Installer on machines that do not currently have it installed.

boundary condition

In testing, test conditions at the limit of normal operation and just over or under those limits.

boxing

The act of wrapping a Value type, such as an Integer, inside an object so that it can be treated like a Reference type.

Break mode

In debugging, a state of suspended execution wherein the application can be executed in a line-by-line manner, and individual program variables can be examined.

breakpoint

In debugging, a predetermined point where execution of an application will halt and enter Break mode.

C

cast

An explicit conversion from one type to another.

circular reference

A condition that occurs when two objects refer to each other but do not have an external reference. Circular references are automatically detected and disposed of by the garbage collector.

class

A user-defined Reference type that serves as a template for an object of that type.

This document is created with the unregistered version of CHM2PDF Pilot

CLR

See common language runtime (CLR).

code access security

Verification that all callers to code have appropriate permission to access it.

code tuning

The act of examining application code and optimizing for faster execution. Code tuning is usually an iterative process that involves several successive rounds of planning, improvement, and assessment of the results.

common language runtime (CLR)

The environment in which managed code executes. The common language runtime provides just-in-time compilation, enforces type safety, and manages memory through garbage collection.

Common Type System

A set of types that are used by all .NET languages, thus ensuring .NET language type compatibility.

complex binding

In data binding, binding more than one column to a single control. The *DataGrid* is an example of a control that can be complexly bound.

component

Strictly speaking, a class that implements *IComponent* or inherits *System*.*Component*. More generally, any class that provides a reusable set of functionality.

connected data access

Data access that requires a data connection that is always open.

Constant

This document is created with the unregistered version of CHM2PDF Pilot

A type that represents a value that does not change in the course of an application's execution and cannot be changed.

constraint

In ADO.NET, an object that represents a rule that enforces referential integrity, such as a constraint that specifies a primary key.

constructor

The method that executes when a class is instantiated. The constructor usually contains initialization code to set member variables to initial values.

container controls

Controls that can host other controls. Examples include *Forms*, *Panels*, and *TabPages*.

control

A class that inherits *System*.*Control*. Controls usually have a visual representation, interact with the user as a part of the user interface, and are hosted in a container control.

controls collection

The collection of a container control that manages the controls it hosts.

custom control

A user-designed control that directly inherits *System*.*Control*.

D

data binding

A way to link data (such as a data set) in your application to the properties of a control.

data consumer

This document is created with the unregistered version of CHM2PDF Pilot

A control that is bound to a data source.

Data Provider

A set of classes that work together to provide managed data access.

DataSet

A class that contains an application copy of data that has been retrieved from a data source.

delegate

An object that acts as a type-safe function pointer. Delegates can be used to invoke the method they point to.

deploying

Installing an application on the computer where it will run.

derived class

A class that is based on another class (called a base class) through inheritance. A derived class inherits the members of its base class and can override or shadow those members.

design time

Represents the state of the application while being designed. Applications at design time consist of noncompiled code that can be manipulated in the code editor or designer window.

destructor

The method that is run just before a class is removed from memory. Generally, a destructor contains any code required to "clean up" after the class. *See also* finalizer.

disconnected data access

Data access that uses a data connection only when executing database operations. This is the default kind of data access implemented by ADO.NET.

This document is created with the unregistered version of CHM2PDF Pilot

docking

In Windows Forms, attaching a control to the edge of the form. You can dock a control by setting the *Dock* property.

E

encapsulation

In component programming, separating the implementation of a component from the interface. Only the public interface of a component is made accessible to the rest of the application. Component data should never be accessible to outside callers.

event

A notification raised by a control or component that a specified condition has occurred.

event handler

A method that is executed when a particular event is raised.

exception

Any class that derives from *System.Exception*. Exceptions are thrown in response to run-time errors. *See also* throwing.

exception handling

The process of dealing with unusual occurrences within code so that it does not cause the program to crash or lose data.

explicit conversion

A conversion of one type to another that cannot be performed automatically. An explicit conversion usually presents some danger of a failed conversion or a loss of data.

extender provider

This document is created with the unregistered version of CHM2PDF Pilot

A specialized type of control that provides design time properties to other controls. *ErrorProvider* and *HelpProvider* are examples of extender providers.

F

field

A member variable of a class or structure.

finalizer

In Visual Basic .NET, the method that is called just before an instance of a class is destroyed. *See also* destructor.

floating-point numbers

A number that has a decimal component. *See also* floating-point types.

floating-point types

Types capable of representing floating-point numbers. Floating-point types include Single (float) and Double (double). *See also* floating-point numbers.

focus

When referring to a control, the condition of being able to receive user input through the keyboard or other input devices. A control that is capable of receiving input is said to have the focus.

foreign key

In a data table, a column that contains a key value that uniquely identifies a record in a related table.

form

The primary unit of the user interface in a Windows Forms application. A form serves as a nexus for data display and user input, and can host controls.

This document is created with the unregistered version of CHM2PDF Pilot

G

GAC

See Global Assembly Cache (GAC).

garbage collection

Automatic memory management provided by the common language runtime. Unused memory is automatically reclaimed by garbage collection without interaction with the application.

GDI+

See Graphic Device Interface (GDI).

getter

The code segment of a property that returns the value the property contains.

Global Assembly Cache (GAC)

A machine-wide cache that stores assemblies that have been specifically designated to be shared between several applications.

globalization

Applying culture-based formatting to existing data, for example, using culture-specific currency symbols and formatting.

Graphic Device Interface (GDI)

In Windows, a graphics display system used by applications to display or print bitmapped text (TrueType fonts), images, and other graphical elements. GDI+ is the name given to the .NET Framework's managed implementation of the GDI.

I

This document is created with the unregistered version of CHM2PDF Pilot

implicit conversion

A conversion between types that can be performed automatically by the common language runtime. An implicit conversion will always succeed and never pose a danger of data loss.

inheritance

In object-oriented programming, creating a new class that incorporates and extends all of the characteristics and functionality of a base class.

inherited control

A user-created control that inherits an existing control. An inherited control inherits all of the functionality of the base control and can incorporate custom functionality as well.

interface

A contract for behavior that can be implemented by a class. Classes that implement the same interface can be expected to provide the members defined by the interface.

L

localization

Providing multiple sets of data specific to different cultures and retrieving the data appropriate to the current culture setting.

logical error

An error that compiles and executes correctly but produces unexpected results. These can be the most difficult errors to detect in application testing.

M

This document is created with the unregistered version of CHM2PDF Pilot

managed code

Code that runs under the common language runtime. The common language runtime handles many tasks that would formerly have been handled in the application's executable. Managed code solves the Windows programming problems of component registration and versioning (sometimes called DLL Hell) because managed code contains all the versioning and type information that the common language runtime needs to run the application. The common language runtime handles registration dynamically at run time rather than statically through the system registry, as is done with applications based on the Component Object Model (COM).

metadata

Data about data. For example, the name, version, security information, and size of a file constitute metadata about the file.

Microsoft Intermediate Language (MSIL)

A low-level language that is just-in-time compiled to native code at run time. All .NET assemblies are represented in the MSIL.

MSIL

See Microsoft Intermediate Language (MSIL).

multidimensional array

An array that contains more than one dimension.

N

namespace

A logical organization of types that perform related functions.

.NET Framework

This document is created with the unregistered version of CHM2PDF Pilot

A managed environment for the development and execution of code. It consists of the common language runtime and .NET Framework base class library.

.NET Framework base class library

A collection of object-oriented types and interfaces that provides object models to assist in the development of your applications.

O

object

The base class for all types in the .NET Framework. More generically, any instance of a class or structure.

object-oriented programming

The process of creating applications by using programmatic constructs to represent real-world objects.

operator overloading

In Visual C#, defining custom behavior for an operator when interacting with custom types.

optimization

The process of examining your application and modifying it to be more efficient in terms of either speed or size of the deployed application.

overloading

Creating multiple methods with the same name but different signatures.

overriding

When inheriting from a base class, providing a new method for one of the base class methods.

P

This document is created with the unregistered version of CHM2PDF Pilot

parent-child relationship

In data access, a relationship between two tables wherein a record from the parent table can be used to retrieve one or more related records from a child table. A parent-child relationship can be defined by using a *DataRelation* object.

polymorphism

The ability of classes to provide different implementations of the same public interface. Thus, two classes that exhibit polymorphism might contain different implementations, but because the interfaces are identical, they can be treated the same in code.

primary key

In a data table, the column that uniquely identifies a record in the table.

private assembly

An assembly that is used by only one application. In .NET Framework programming, most assemblies are private assemblies.

project

The collection of Visual Studio .NET source files that make up an application.

R

Reference type

A type, such as a class, whose instance data is allocated on the heap. Variables that represent Reference types contain a reference to the object on the heap as opposed to the object's data itself. *See also* Value type.

registry

The system repository for information about a computer's settings. The registry can contain configuration information

This document is created with the unregistered version of CHM2PDF Pilot

about programs and system components.

resource

Nonexecutable data, such as a string or image, that is deployed in an application.

role-based security

Code that ensures that users of your application have appropriate permission to access your application based on the identity of the user.

run time

Represents the state of the application while executing. At run time, the application consists of executing code.

run-time error

An error that causes an invalid result or attempts an invalid operation at run time.

S

satellite assembly

When localizing an application, assemblies that contain alternate sets of resources to be used in the application for different cultures.

setter

The code segment of a property that sets the property's value.

shadowing

Hiding a member in a base class with a member in an inherited class that has a different signature.

shared assembly

This document is created with the unregistered version of CHM2PDF Pilot

An assembly that is referenced by several applications on a single machine. A shared assembly must be installed to the Global Assembly Cache (GAC).

shared members

In Visual Basic .NET, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Shared members are called static members in Visual C#.

simple binding

In data binding, binding a control to a single data column. A *TextBox* is an example of a control that can be simply bound.

solution

A group of Visual Studio .NET projects that make up a single functional unit.

SQL

See Structured Query Language (SQL).

static members

In Visual C#, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Static members are called shared members in Visual Basic .NET.

Step Into

Moving from a calling procedure to a called procedure during debugging. Also used in reference to setting breakpoints to stop execution in a specific procedure during debugging.

Step Out

In Break mode, resuming execution of a function call until the call returns.

Step Over

In Break mode, executing a procedure call as a single statement during debugging.

This document is created with the unregistered version of CHM2PDF Pilot

strong name

A name that uniquely identifies an assembly. It consists of the assembly's name, version, and culture information as well as a public key encrypted by a private key. An assembly must have a strong name to be installed to the Global Assembly Cache (GAC).

strongly typed

Refers to types for all variables being explicitly declared. Strong typing eliminates the possibility of type mismatch errors, which are discovered at compile time. Visual C# is inherently strongly typed, and Visual Basic .NET is strongly typed when Option Strict is set to On.

Struct

In Visual C#, a user-defined Value type that serves as a template for an instance of that type. *See also* Structure.

Structure

In Visual Basic .NET, a user-defined Value type that serves as a template for an instance of that type. *See also* Struct .

structured exception handling

Using *Try...Catch...Finally* blocks to handle errors that might occur during code execution. When exceptions are thrown, they are handled in the *Catch* block before code in the *Finally* block is executed.

Structured Query Language (SQL)

The standard language used for issuing commands to databases.

syntax error

An error that occurs because of typographical errors or nonsensical code. In the .NET Framework, syntax errors prevent compilation and are thus discovered at compile time.

T

target audience

This document is created with the unregistered version of CHM2PDF Pilot

The group of end users for whom an application is designed.

threads

The basic unit to which the server allocates processor time. A single process can have multiple threads.

throwing

The act of passing an exception to the next caller in the call stack. *See also* exception.

tracing

A programming technique for recording events, such as exceptions, in an application. Tracing is used during debugging and in the testing phase of application deployment.

transaction

A group of commands (treated as a single unit) that change the data stored in a database. The transaction ensures that the commands are handled in an all-or-nothing fashion if one of the commands fails, all of the commands fail, and any data that was written to the database by the commands is backed out. In this way, transactions maintain the integrity of data in a database.

tuning

The process of making adjustments to a deployed application that do not affect code.

U

Unicode

The universal character encoding scheme for characters and text. Each character is represented by a unique numeric value.

unmanaged code

Code that is not managed by the common language runtime. Unmanaged code is not checked for type-safety and

This document is created with the unregistered version of CHM2PDF Pilot

must be used with extreme care.

user control

A user-defined control that encapsulates other Windows Forms controls with custom functionality in a common container.

V

Value type

A type, such as an Integer, Decimal, or Structure, whose instance data is allocated on the stack. Variables that represent Value types contain the object's instance data itself as opposed to a reference to that data. *See also* Reference type.

W

Watch window

When debugging, the window that allows you to examine the values of predetermined watch variables.

Web Service

A service or component that is implemented by a Web server and can be instantiated and accessed programmatically as a component over the World Wide Web.

X

XML Schema

A description of the data elements contained in an XML file. The XML Schema provides the names of the elements, their types, whether or not they are key fields, and other information.

This document is created with the unregistered version of CHM2PDF Pilot

Glossary

A

abstract class

A class that cannot be instantiated but is used as a base from which other classes can be derived. In Microsoft Visual Basic .NET, abstract classes are declared using the *MustInherit* keyword. In Microsoft Visual C#, abstract classes are declared using the *abstract* keyword.

abstraction

Representing real-world objects as programmatic constructs.

abstract members

A member of a base class that cannot be invoked, but instead provides a template for members of a derived class. In Visual Basic .NET, abstract members are declared using the *MustOverride* keyword. In Visual C#, abstract members are declared using the *abstract* keyword.

ADO.NET

The data access architecture for the Microsoft .NET Framework. ADO.NET is built around a disconnected data access model that uses a set of classes called a Data Provider to retrieve data from a data source.

alias

The name you use to identify resources on the Web. Aliases represent physical resources on the Web server, such as a Web form, an HTML page, or a graphic.

anchoring

In Windows Forms controls, the resizing and repositioning of a control when its parent form is resized. You can anchor a control by setting the *Anchor* property.

This document is created with the unregistered version of CHM2PDF Pilot

assembly

The primary unit of deployment for a .NET application. An assembly is either an executable application (EXE) or a class library (DLL).

assembly manifest

The metadata for the assembly. It contains all of the information needed to describe the assembly to the common language runtime.

authentication

The process of determining the identity of a user. In effect, authentication validates that the users are who they say they are. *See also* authorization.

authorization

The process of allowing access to an application or resource based on credentials supplied by the user. *See also* authentication.

B

base class

A class that provides properties and methods as a foundation for a derived class. In objected-oriented programming, one class can be based on another through inheritance. Using this technique, the base class provides characteristics (such as properties and methods) to a derived class. The derived class can reuse, modify, or add to the members of the base class.

Bezier

A curve specified by four points. Two points specify the ends of the curve, and the other two points influence the shape of the curve between the end points.

Boolean

A data type that can represent a value of either true or false.

This document is created with the unregistered version of CHM2PDF Pilot

bootstrapper

In deployment, an application that installs Microsoft Windows Installer on machines that do not currently have it installed.

boundary condition

In testing, test conditions at the limit of normal operation and just over or under those limits.

boxing

The act of wrapping a Value type, such as an Integer, inside an object so that it can be treated like a Reference type.

Break mode

In debugging, a state of suspended execution wherein the application can be executed in a line-by-line manner, and individual program variables can be examined.

breakpoint

In debugging, a predetermined point where execution of an application will halt and enter Break mode.

C

cast

An explicit conversion from one type to another.

circular reference

A condition that occurs when two objects refer to each other but do not have an external reference. Circular references are automatically detected and disposed of by the garbage collector.

class

A user-defined Reference type that serves as a template for an object of that type.

This document is created with the unregistered version of CHM2PDF Pilot

CLR

See common language runtime (CLR).

code access security

Verification that all callers to code have appropriate permission to access it.

code tuning

The act of examining application code and optimizing for faster execution. Code tuning is usually an iterative process that involves several successive rounds of planning, improvement, and assessment of the results.

common language runtime (CLR)

The environment in which managed code executes. The common language runtime provides just-in-time compilation, enforces type safety, and manages memory through garbage collection.

Common Type System

A set of types that are used by all .NET languages, thus ensuring .NET language type compatibility.

complex binding

In data binding, binding more than one column to a single control. The *DataGrid* is an example of a control that can be complexly bound.

component

Strictly speaking, a class that implements *IComponent* or inherits *System*.*Component*. More generally, any class that provides a reusable set of functionality.

connected data access

Data access that requires a data connection that is always open.

Constant

This document is created with the unregistered version of CHM2PDF Pilot

A type that represents a value that does not change in the course of an application's execution and cannot be changed.

constraint

In ADO.NET, an object that represents a rule that enforces referential integrity, such as a constraint that specifies a primary key.

constructor

The method that executes when a class is instantiated. The constructor usually contains initialization code to set member variables to initial values.

container controls

Controls that can host other controls. Examples include *Forms*, *Panels*, and *TabPages*.

control

A class that inherits *System*.*Control*. Controls usually have a visual representation, interact with the user as a part of the user interface, and are hosted in a container control.

controls collection

The collection of a container control that manages the controls it hosts.

custom control

A user-designed control that directly inherits *System*.*Control*.

D

data binding

A way to link data (such as a data set) in your application to the properties of a control.

data consumer

This document is created with the unregistered version of CHM2PDF Pilot

A control that is bound to a data source.

Data Provider

A set of classes that work together to provide managed data access.

DataSet

A class that contains an application copy of data that has been retrieved from a data source.

delegate

An object that acts as a type-safe function pointer. Delegates can be used to invoke the method they point to.

deploying

Installing an application on the computer where it will run.

derived class

A class that is based on another class (called a base class) through inheritance. A derived class inherits the members of its base class and can override or shadow those members.

design time

Represents the state of the application while being designed. Applications at design time consist of noncompiled code that can be manipulated in the code editor or designer window.

destructor

The method that is run just before a class is removed from memory. Generally, a destructor contains any code required to "clean up" after the class. *See also* finalizer.

disconnected data access

Data access that uses a data connection only when executing database operations. This is the default kind of data access implemented by ADO.NET.

This document is created with the unregistered version of CHM2PDF Pilot

docking

In Windows Forms, attaching a control to the edge of the form. You can dock a control by setting the *Dock* property.

E

encapsulation

In component programming, separating the implementation of a component from the interface. Only the public interface of a component is made accessible to the rest of the application. Component data should never be accessible to outside callers.

event

A notification raised by a control or component that a specified condition has occurred.

event handler

A method that is executed when a particular event is raised.

exception

Any class that derives from *System.Exception*. Exceptions are thrown in response to run-time errors. *See also* throwing.

exception handling

The process of dealing with unusual occurrences within code so that it does not cause the program to crash or lose data.

explicit conversion

A conversion of one type to another that cannot be performed automatically. An explicit conversion usually presents some danger of a failed conversion or a loss of data.

extender provider

This document is created with the unregistered version of CHM2PDF Pilot

A specialized type of control that provides design time properties to other controls. *ErrorProvider* and *HelpProvider* are examples of extender providers.

F

field

A member variable of a class or structure.

finalizer

In Visual Basic .NET, the method that is called just before an instance of a class is destroyed. *See also* destructor.

floating-point numbers

A number that has a decimal component. *See also* floating-point types.

floating-point types

Types capable of representing floating-point numbers. Floating-point types include Single (float) and Double (double). *See also* floating-point numbers.

focus

When referring to a control, the condition of being able to receive user input through the keyboard or other input devices. A control that is capable of receiving input is said to have the focus.

foreign key

In a data table, a column that contains a key value that uniquely identifies a record in a related table.

form

The primary unit of the user interface in a Windows Forms application. A form serves as a nexus for data display and user input, and can host controls.

This document is created with the unregistered version of CHM2PDF Pilot

G

GAC

See Global Assembly Cache (GAC).

garbage collection

Automatic memory management provided by the common language runtime. Unused memory is automatically reclaimed by garbage collection without interaction with the application.

GDI+

See Graphic Device Interface (GDI).

getter

The code segment of a property that returns the value the property contains.

Global Assembly Cache (GAC)

A machine-wide cache that stores assemblies that have been specifically designated to be shared between several applications.

globalization

Applying culture-based formatting to existing data, for example, using culture-specific currency symbols and formatting.

Graphic Device Interface (GDI)

In Windows, a graphics display system used by applications to display or print bitmapped text (TrueType fonts), images, and other graphical elements. GDI+ is the name given to the .NET Framework's managed implementation of the GDI.

I

This document is created with the unregistered version of CHM2PDF Pilot

implicit conversion

A conversion between types that can be performed automatically by the common language runtime. An implicit conversion will always succeed and never pose a danger of data loss.

inheritance

In object-oriented programming, creating a new class that incorporates and extends all of the characteristics and functionality of a base class.

inherited control

A user-created control that inherits an existing control. An inherited control inherits all of the functionality of the base control and can incorporate custom functionality as well.

interface

A contract for behavior that can be implemented by a class. Classes that implement the same interface can be expected to provide the members defined by the interface.

L

localization

Providing multiple sets of data specific to different cultures and retrieving the data appropriate to the current culture setting.

logical error

An error that compiles and executes correctly but produces unexpected results. These can be the most difficult errors to detect in application testing.

M

This document is created with the unregistered version of CHM2PDF Pilot

managed code

Code that runs under the common language runtime. The common language runtime handles many tasks that would formerly have been handled in the application's executable. Managed code solves the Windows programming problems of component registration and versioning (sometimes called DLL Hell) because managed code contains all the versioning and type information that the common language runtime needs to run the application. The common language runtime handles registration dynamically at run time rather than statically through the system registry, as is done with applications based on the Component Object Model (COM).

metadata

Data about data. For example, the name, version, security information, and size of a file constitute metadata about the file.

Microsoft Intermediate Language (MSIL)

A low-level language that is just-in-time compiled to native code at run time. All .NET assemblies are represented in the MSIL.

MSIL

See Microsoft Intermediate Language (MSIL).

multidimensional array

An array that contains more than one dimension.

N

namespace

A logical organization of types that perform related functions.

.NET Framework

This document is created with the unregistered version of CHM2PDF Pilot

A managed environment for the development and execution of code. It consists of the common language runtime and .NET Framework base class library.

.NET Framework base class library

A collection of object-oriented types and interfaces that provides object models to assist in the development of your applications.

O

object

The base class for all types in the .NET Framework. More generically, any instance of a class or structure.

object-oriented programming

The process of creating applications by using programmatic constructs to represent real-world objects.

operator overloading

In Visual C#, defining custom behavior for an operator when interacting with custom types.

optimization

The process of examining your application and modifying it to be more efficient in terms of either speed or size of the deployed application.

overloading

Creating multiple methods with the same name but different signatures.

overriding

When inheriting from a base class, providing a new method for one of the base class methods.

P

This document is created with the unregistered version of CHM2PDF Pilot

parent-child relationship

In data access, a relationship between two tables wherein a record from the parent table can be used to retrieve one or more related records from a child table. A parent-child relationship can be defined by using a *DataRelation* object.

polymorphism

The ability of classes to provide different implementations of the same public interface. Thus, two classes that exhibit polymorphism might contain different implementations, but because the interfaces are identical, they can be treated the same in code.

primary key

In a data table, the column that uniquely identifies a record in the table.

private assembly

An assembly that is used by only one application. In .NET Framework programming, most assemblies are private assemblies.

project

The collection of Visual Studio .NET source files that make up an application.

R

Reference type

A type, such as a class, whose instance data is allocated on the heap. Variables that represent Reference types contain a reference to the object on the heap as opposed to the object's data itself. *See also* Value type.

registry

The system repository for information about a computer's settings. The registry can contain configuration information

This document is created with the unregistered version of CHM2PDF Pilot

about programs and system components.

resource

Nonexecutable data, such as a string or image, that is deployed in an application.

role-based security

Code that ensures that users of your application have appropriate permission to access your application based on the identity of the user.

run time

Represents the state of the application while executing. At run time, the application consists of executing code.

run-time error

An error that causes an invalid result or attempts an invalid operation at run time.

S

satellite assembly

When localizing an application, assemblies that contain alternate sets of resources to be used in the application for different cultures.

setter

The code segment of a property that sets the property's value.

shadowing

Hiding a member in a base class with a member in an inherited class that has a different signature.

shared assembly

This document is created with the unregistered version of CHM2PDF Pilot

An assembly that is referenced by several applications on a single machine. A shared assembly must be installed to the Global Assembly Cache (GAC).

shared members

In Visual Basic .NET, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Shared members are called static members in Visual C#.

simple binding

In data binding, binding a control to a single data column. A *TextBox* is an example of a control that can be simply bound.

solution

A group of Visual Studio .NET projects that make up a single functional unit.

SQL

See Structured Query Language (SQL).

static members

In Visual C#, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Static members are called shared members in Visual Basic .NET.

Step Into

Moving from a calling procedure to a called procedure during debugging. Also used in reference to setting breakpoints to stop execution in a specific procedure during debugging.

Step Out

In Break mode, resuming execution of a function call until the call returns.

Step Over

In Break mode, executing a procedure call as a single statement during debugging.

This document is created with the unregistered version of CHM2PDF Pilot

strong name

A name that uniquely identifies an assembly. It consists of the assembly's name, version, and culture information as well as a public key encrypted by a private key. An assembly must have a strong name to be installed to the Global Assembly Cache (GAC).

strongly typed

Refers to types for all variables being explicitly declared. Strong typing eliminates the possibility of type mismatch errors, which are discovered at compile time. Visual C# is inherently strongly typed, and Visual Basic .NET is strongly typed when Option Strict is set to On.

Struct

In Visual C#, a user-defined Value type that serves as a template for an instance of that type. *See also* Structure.

Structure

In Visual Basic .NET, a user-defined Value type that serves as a template for an instance of that type. *See also* Struct .

structured exception handling

Using *Try...Catch...Finally* blocks to handle errors that might occur during code execution. When exceptions are thrown, they are handled in the *Catch* block before code in the *Finally* block is executed.

Structured Query Language (SQL)

The standard language used for issuing commands to databases.

syntax error

An error that occurs because of typographical errors or nonsensical code. In the .NET Framework, syntax errors prevent compilation and are thus discovered at compile time.

T

target audience

This document is created with the unregistered version of CHM2PDF Pilot

The group of end users for whom an application is designed.

threads

The basic unit to which the server allocates processor time. A single process can have multiple threads.

throwing

The act of passing an exception to the next caller in the call stack. *See also* exception.

tracing

A programming technique for recording events, such as exceptions, in an application. Tracing is used during debugging and in the testing phase of application deployment.

transaction

A group of commands (treated as a single unit) that change the data stored in a database. The transaction ensures that the commands are handled in an all-or-nothing fashion if one of the commands fails, all of the commands fail, and any data that was written to the database by the commands is backed out. In this way, transactions maintain the integrity of data in a database.

tuning

The process of making adjustments to a deployed application that do not affect code.

U

Unicode

The universal character encoding scheme for characters and text. Each character is represented by a unique numeric value.

unmanaged code

Code that is not managed by the common language runtime. Unmanaged code is not checked for type-safety and

This document is created with the unregistered version of CHM2PDF Pilot

must be used with extreme care.

user control

A user-defined control that encapsulates other Windows Forms controls with custom functionality in a common container.

V

Value type

A type, such as an Integer, Decimal, or Structure, whose instance data is allocated on the stack. Variables that represent Value types contain the object's instance data itself as opposed to a reference to that data. *See also* Reference type.

W

Watch window

When debugging, the window that allows you to examine the values of predetermined watch variables.

Web Service

A service or component that is implemented by a Web server and can be instantiated and accessed programmatically as a component over the World Wide Web.

X

XML Schema

A description of the data elements contained in an XML file. The XML Schema provides the names of the elements, their types, whether or not they are key fields, and other information.

This document is created with the unregistered version of CHM2PDF Pilot

Glossary

A

abstract class

A class that cannot be instantiated but is used as a base from which other classes can be derived. In Microsoft Visual Basic .NET, abstract classes are declared using the *MustInherit* keyword. In Microsoft Visual C#, abstract classes are declared using the *abstract* keyword.

abstraction

Representing real-world objects as programmatic constructs.

abstract members

A member of a base class that cannot be invoked, but instead provides a template for members of a derived class. In Visual Basic .NET, abstract members are declared using the *MustOverride* keyword. In Visual C#, abstract members are declared using the *abstract* keyword.

ADO.NET

The data access architecture for the Microsoft .NET Framework. ADO.NET is built around a disconnected data access model that uses a set of classes called a Data Provider to retrieve data from a data source.

alias

The name you use to identify resources on the Web. Aliases represent physical resources on the Web server, such as a Web form, an HTML page, or a graphic.

anchoring

In Windows Forms controls, the resizing and repositioning of a control when its parent form is resized. You can anchor a control by setting the *Anchor* property.

This document is created with the unregistered version of CHM2PDF Pilot

assembly

The primary unit of deployment for a .NET application. An assembly is either an executable application (EXE) or a class library (DLL).

assembly manifest

The metadata for the assembly. It contains all of the information needed to describe the assembly to the common language runtime.

authentication

The process of determining the identity of a user. In effect, authentication validates that the users are who they say they are. *See also* authorization.

authorization

The process of allowing access to an application or resource based on credentials supplied by the user. *See also* authentication.

B

base class

A class that provides properties and methods as a foundation for a derived class. In objected-oriented programming, one class can be based on another through inheritance. Using this technique, the base class provides characteristics (such as properties and methods) to a derived class. The derived class can reuse, modify, or add to the members of the base class.

Bezier

A curve specified by four points. Two points specify the ends of the curve, and the other two points influence the shape of the curve between the end points.

Boolean

A data type that can represent a value of either true or false.

This document is created with the unregistered version of CHM2PDF Pilot

bootstrapper

In deployment, an application that installs Microsoft Windows Installer on machines that do not currently have it installed.

boundary condition

In testing, test conditions at the limit of normal operation and just over or under those limits.

boxing

The act of wrapping a Value type, such as an Integer, inside an object so that it can be treated like a Reference type.

Break mode

In debugging, a state of suspended execution wherein the application can be executed in a line-by-line manner, and individual program variables can be examined.

breakpoint

In debugging, a predetermined point where execution of an application will halt and enter Break mode.

C

cast

An explicit conversion from one type to another.

circular reference

A condition that occurs when two objects refer to each other but do not have an external reference. Circular references are automatically detected and disposed of by the garbage collector.

class

A user-defined Reference type that serves as a template for an object of that type.

This document is created with the unregistered version of CHM2PDF Pilot

CLR

See common language runtime (CLR).

code access security

Verification that all callers to code have appropriate permission to access it.

code tuning

The act of examining application code and optimizing for faster execution. Code tuning is usually an iterative process that involves several successive rounds of planning, improvement, and assessment of the results.

common language runtime (CLR)

The environment in which managed code executes. The common language runtime provides just-in-time compilation, enforces type safety, and manages memory through garbage collection.

Common Type System

A set of types that are used by all .NET languages, thus ensuring .NET language type compatibility.

complex binding

In data binding, binding more than one column to a single control. The *DataGrid* is an example of a control that can be complexly bound.

component

Strictly speaking, a class that implements *IComponent* or inherits *System*.*Component*. More generally, any class that provides a reusable set of functionality.

connected data access

Data access that requires a data connection that is always open.

Constant

This document is created with the unregistered version of CHM2PDF Pilot

A type that represents a value that does not change in the course of an application's execution and cannot be changed.

constraint

In ADO.NET, an object that represents a rule that enforces referential integrity, such as a constraint that specifies a primary key.

constructor

The method that executes when a class is instantiated. The constructor usually contains initialization code to set member variables to initial values.

container controls

Controls that can host other controls. Examples include *Forms*, *Panels*, and *TabPages*.

control

A class that inherits *System*.*Control*. Controls usually have a visual representation, interact with the user as a part of the user interface, and are hosted in a container control.

controls collection

The collection of a container control that manages the controls it hosts.

custom control

A user-designed control that directly inherits *System*.*Control*.

D

data binding

A way to link data (such as a data set) in your application to the properties of a control.

data consumer

This document is created with the unregistered version of CHM2PDF Pilot

A control that is bound to a data source.

Data Provider

A set of classes that work together to provide managed data access.

DataSet

A class that contains an application copy of data that has been retrieved from a data source.

delegate

An object that acts as a type-safe function pointer. Delegates can be used to invoke the method they point to.

deploying

Installing an application on the computer where it will run.

derived class

A class that is based on another class (called a base class) through inheritance. A derived class inherits the members of its base class and can override or shadow those members.

design time

Represents the state of the application while being designed. Applications at design time consist of noncompiled code that can be manipulated in the code editor or designer window.

destructor

The method that is run just before a class is removed from memory. Generally, a destructor contains any code required to "clean up" after the class. *See also* finalizer.

disconnected data access

Data access that uses a data connection only when executing database operations. This is the default kind of data access implemented by ADO.NET.

This document is created with the unregistered version of CHM2PDF Pilot

docking

In Windows Forms, attaching a control to the edge of the form. You can dock a control by setting the *Dock* property.

E

encapsulation

In component programming, separating the implementation of a component from the interface. Only the public interface of a component is made accessible to the rest of the application. Component data should never be accessible to outside callers.

event

A notification raised by a control or component that a specified condition has occurred.

event handler

A method that is executed when a particular event is raised.

exception

Any class that derives from *System.Exception*. Exceptions are thrown in response to run-time errors. *See also* throwing.

exception handling

The process of dealing with unusual occurrences within code so that it does not cause the program to crash or lose data.

explicit conversion

A conversion of one type to another that cannot be performed automatically. An explicit conversion usually presents some danger of a failed conversion or a loss of data.

extender provider

This document is created with the unregistered version of CHM2PDF Pilot

A specialized type of control that provides design time properties to other controls. *ErrorProvider* and *HelpProvider* are examples of extender providers.

F

field

A member variable of a class or structure.

finalizer

In Visual Basic .NET, the method that is called just before an instance of a class is destroyed. *See also* destructor.

floating-point numbers

A number that has a decimal component. *See also* floating-point types.

floating-point types

Types capable of representing floating-point numbers. Floating-point types include Single (float) and Double (double). *See also* floating-point numbers.

focus

When referring to a control, the condition of being able to receive user input through the keyboard or other input devices. A control that is capable of receiving input is said to have the focus.

foreign key

In a data table, a column that contains a key value that uniquely identifies a record in a related table.

form

The primary unit of the user interface in a Windows Forms application. A form serves as a nexus for data display and user input, and can host controls.

This document is created with the unregistered version of CHM2PDF Pilot

G

GAC

See Global Assembly Cache (GAC).

garbage collection

Automatic memory management provided by the common language runtime. Unused memory is automatically reclaimed by garbage collection without interaction with the application.

GDI+

See Graphic Device Interface (GDI).

getter

The code segment of a property that returns the value the property contains.

Global Assembly Cache (GAC)

A machine-wide cache that stores assemblies that have been specifically designated to be shared between several applications.

globalization

Applying culture-based formatting to existing data, for example, using culture-specific currency symbols and formatting.

Graphic Device Interface (GDI)

In Windows, a graphics display system used by applications to display or print bitmapped text (TrueType fonts), images, and other graphical elements. GDI+ is the name given to the .NET Framework's managed implementation of the GDI.

I

This document is created with the unregistered version of CHM2PDF Pilot

implicit conversion

A conversion between types that can be performed automatically by the common language runtime. An implicit conversion will always succeed and never pose a danger of data loss.

inheritance

In object-oriented programming, creating a new class that incorporates and extends all of the characteristics and functionality of a base class.

inherited control

A user-created control that inherits an existing control. An inherited control inherits all of the functionality of the base control and can incorporate custom functionality as well.

interface

A contract for behavior that can be implemented by a class. Classes that implement the same interface can be expected to provide the members defined by the interface.

L

localization

Providing multiple sets of data specific to different cultures and retrieving the data appropriate to the current culture setting.

logical error

An error that compiles and executes correctly but produces unexpected results. These can be the most difficult errors to detect in application testing.

M

This document is created with the unregistered version of CHM2PDF Pilot

managed code

Code that runs under the common language runtime. The common language runtime handles many tasks that would formerly have been handled in the application's executable. Managed code solves the Windows programming problems of component registration and versioning (sometimes called DLL Hell) because managed code contains all the versioning and type information that the common language runtime needs to run the application. The common language runtime handles registration dynamically at run time rather than statically through the system registry, as is done with applications based on the Component Object Model (COM).

metadata

Data about data. For example, the name, version, security information, and size of a file constitute metadata about the file.

Microsoft Intermediate Language (MSIL)

A low-level language that is just-in-time compiled to native code at run time. All .NET assemblies are represented in the MSIL.

MSIL

See Microsoft Intermediate Language (MSIL).

multidimensional array

An array that contains more than one dimension.

N

namespace

A logical organization of types that perform related functions.

.NET Framework

This document is created with the unregistered version of CHM2PDF Pilot

A managed environment for the development and execution of code. It consists of the common language runtime and .NET Framework base class library.

.NET Framework base class library

A collection of object-oriented types and interfaces that provides object models to assist in the development of your applications.

O

object

The base class for all types in the .NET Framework. More generically, any instance of a class or structure.

object-oriented programming

The process of creating applications by using programmatic constructs to represent real-world objects.

operator overloading

In Visual C#, defining custom behavior for an operator when interacting with custom types.

optimization

The process of examining your application and modifying it to be more efficient in terms of either speed or size of the deployed application.

overloading

Creating multiple methods with the same name but different signatures.

overriding

When inheriting from a base class, providing a new method for one of the base class methods.

P

This document is created with the unregistered version of CHM2PDF Pilot

parent-child relationship

In data access, a relationship between two tables wherein a record from the parent table can be used to retrieve one or more related records from a child table. A parent-child relationship can be defined by using a *DataRelation* object.

polymorphism

The ability of classes to provide different implementations of the same public interface. Thus, two classes that exhibit polymorphism might contain different implementations, but because the interfaces are identical, they can be treated the same in code.

primary key

In a data table, the column that uniquely identifies a record in the table.

private assembly

An assembly that is used by only one application. In .NET Framework programming, most assemblies are private assemblies.

project

The collection of Visual Studio .NET source files that make up an application.

R

Reference type

A type, such as a class, whose instance data is allocated on the heap. Variables that represent Reference types contain a reference to the object on the heap as opposed to the object's data itself. *See also* Value type.

registry

The system repository for information about a computer's settings. The registry can contain configuration information

This document is created with the unregistered version of CHM2PDF Pilot

about programs and system components.

resource

Nonexecutable data, such as a string or image, that is deployed in an application.

role-based security

Code that ensures that users of your application have appropriate permission to access your application based on the identity of the user.

run time

Represents the state of the application while executing. At run time, the application consists of executing code.

run-time error

An error that causes an invalid result or attempts an invalid operation at run time.

S

satellite assembly

When localizing an application, assemblies that contain alternate sets of resources to be used in the application for different cultures.

setter

The code segment of a property that sets the property's value.

shadowing

Hiding a member in a base class with a member in an inherited class that has a different signature.

shared assembly

This document is created with the unregistered version of CHM2PDF Pilot

An assembly that is referenced by several applications on a single machine. A shared assembly must be installed to the Global Assembly Cache (GAC).

shared members

In Visual Basic .NET, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Shared members are called static members in Visual C#.

simple binding

In data binding, binding a control to a single data column. A *TextBox* is an example of a control that can be simply bound.

solution

A group of Visual Studio .NET projects that make up a single functional unit.

SQL

See Structured Query Language (SQL).

static members

In Visual C#, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Static members are called shared members in Visual Basic .NET.

Step Into

Moving from a calling procedure to a called procedure during debugging. Also used in reference to setting breakpoints to stop execution in a specific procedure during debugging.

Step Out

In Break mode, resuming execution of a function call until the call returns.

Step Over

In Break mode, executing a procedure call as a single statement during debugging.

This document is created with the unregistered version of CHM2PDF Pilot

strong name

A name that uniquely identifies an assembly. It consists of the assembly's name, version, and culture information as well as a public key encrypted by a private key. An assembly must have a strong name to be installed to the Global Assembly Cache (GAC).

strongly typed

Refers to types for all variables being explicitly declared. Strong typing eliminates the possibility of type mismatch errors, which are discovered at compile time. Visual C# is inherently strongly typed, and Visual Basic .NET is strongly typed when Option Strict is set to On.

Struct

In Visual C#, a user-defined Value type that serves as a template for an instance of that type. *See also* Structure.

Structure

In Visual Basic .NET, a user-defined Value type that serves as a template for an instance of that type. *See also* Struct .

structured exception handling

Using *Try...Catch...Finally* blocks to handle errors that might occur during code execution. When exceptions are thrown, they are handled in the *Catch* block before code in the *Finally* block is executed.

Structured Query Language (SQL)

The standard language used for issuing commands to databases.

syntax error

An error that occurs because of typographical errors or nonsensical code. In the .NET Framework, syntax errors prevent compilation and are thus discovered at compile time.

T

target audience

This document is created with the unregistered version of CHM2PDF Pilot

The group of end users for whom an application is designed.

threads

The basic unit to which the server allocates processor time. A single process can have multiple threads.

throwing

The act of passing an exception to the next caller in the call stack. *See also* exception.

tracing

A programming technique for recording events, such as exceptions, in an application. Tracing is used during debugging and in the testing phase of application deployment.

transaction

A group of commands (treated as a single unit) that change the data stored in a database. The transaction ensures that the commands are handled in an all-or-nothing fashion if one of the commands fails, all of the commands fail, and any data that was written to the database by the commands is backed out. In this way, transactions maintain the integrity of data in a database.

tuning

The process of making adjustments to a deployed application that do not affect code.

U

Unicode

The universal character encoding scheme for characters and text. Each character is represented by a unique numeric value.

unmanaged code

Code that is not managed by the common language runtime. Unmanaged code is not checked for type-safety and

This document is created with the unregistered version of CHM2PDF Pilot

must be used with extreme care.

user control

A user-defined control that encapsulates other Windows Forms controls with custom functionality in a common container.

V

Value type

A type, such as an Integer, Decimal, or Structure, whose instance data is allocated on the stack. Variables that represent Value types contain the object's instance data itself as opposed to a reference to that data. *See also* Reference type.

W

Watch window

When debugging, the window that allows you to examine the values of predetermined watch variables.

Web Service

A service or component that is implemented by a Web server and can be instantiated and accessed programmatically as a component over the World Wide Web.

X

XML Schema

A description of the data elements contained in an XML file. The XML Schema provides the names of the elements, their types, whether or not they are key fields, and other information.

This document is created with the unregistered version of CHM2PDF Pilot

Glossary

A

abstract class

A class that cannot be instantiated but is used as a base from which other classes can be derived. In Microsoft Visual Basic .NET, abstract classes are declared using the *MustInherit* keyword. In Microsoft Visual C#, abstract classes are declared using the *abstract* keyword.

abstraction

Representing real-world objects as programmatic constructs.

abstract members

A member of a base class that cannot be invoked, but instead provides a template for members of a derived class. In Visual Basic .NET, abstract members are declared using the *MustOverride* keyword. In Visual C#, abstract members are declared using the *abstract* keyword.

ADO.NET

The data access architecture for the Microsoft .NET Framework. ADO.NET is built around a disconnected data access model that uses a set of classes called a Data Provider to retrieve data from a data source.

alias

The name you use to identify resources on the Web. Aliases represent physical resources on the Web server, such as a Web form, an HTML page, or a graphic.

anchoring

In Windows Forms controls, the resizing and repositioning of a control when its parent form is resized. You can anchor a control by setting the *Anchor* property.

This document is created with the unregistered version of CHM2PDF Pilot

assembly

The primary unit of deployment for a .NET application. An assembly is either an executable application (EXE) or a class library (DLL).

assembly manifest

The metadata for the assembly. It contains all of the information needed to describe the assembly to the common language runtime.

authentication

The process of determining the identity of a user. In effect, authentication validates that the users are who they say they are. *See also* authorization.

authorization

The process of allowing access to an application or resource based on credentials supplied by the user. *See also* authentication.

B

base class

A class that provides properties and methods as a foundation for a derived class. In objected-oriented programming, one class can be based on another through inheritance. Using this technique, the base class provides characteristics (such as properties and methods) to a derived class. The derived class can reuse, modify, or add to the members of the base class.

Bezier

A curve specified by four points. Two points specify the ends of the curve, and the other two points influence the shape of the curve between the end points.

Boolean

A data type that can represent a value of either true or false.

This document is created with the unregistered version of CHM2PDF Pilot

bootstrapper

In deployment, an application that installs Microsoft Windows Installer on machines that do not currently have it installed.

boundary condition

In testing, test conditions at the limit of normal operation and just over or under those limits.

boxing

The act of wrapping a Value type, such as an Integer, inside an object so that it can be treated like a Reference type.

Break mode

In debugging, a state of suspended execution wherein the application can be executed in a line-by-line manner, and individual program variables can be examined.

breakpoint

In debugging, a predetermined point where execution of an application will halt and enter Break mode.

C

cast

An explicit conversion from one type to another.

circular reference

A condition that occurs when two objects refer to each other but do not have an external reference. Circular references are automatically detected and disposed of by the garbage collector.

class

A user-defined Reference type that serves as a template for an object of that type.

This document is created with the unregistered version of CHM2PDF Pilot

CLR

See common language runtime (CLR).

code access security

Verification that all callers to code have appropriate permission to access it.

code tuning

The act of examining application code and optimizing for faster execution. Code tuning is usually an iterative process that involves several successive rounds of planning, improvement, and assessment of the results.

common language runtime (CLR)

The environment in which managed code executes. The common language runtime provides just-in-time compilation, enforces type safety, and manages memory through garbage collection.

Common Type System

A set of types that are used by all .NET languages, thus ensuring .NET language type compatibility.

complex binding

In data binding, binding more than one column to a single control. The *DataGrid* is an example of a control that can be complexly bound.

component

Strictly speaking, a class that implements *IComponent* or inherits *System*.*Component*. More generally, any class that provides a reusable set of functionality.

connected data access

Data access that requires a data connection that is always open.

Constant

This document is created with the unregistered version of CHM2PDF Pilot

A type that represents a value that does not change in the course of an application's execution and cannot be changed.

constraint

In ADO.NET, an object that represents a rule that enforces referential integrity, such as a constraint that specifies a primary key.

constructor

The method that executes when a class is instantiated. The constructor usually contains initialization code to set member variables to initial values.

container controls

Controls that can host other controls. Examples include *Forms*, *Panels*, and *TabPages*.

control

A class that inherits *System*.*Control*. Controls usually have a visual representation, interact with the user as a part of the user interface, and are hosted in a container control.

controls collection

The collection of a container control that manages the controls it hosts.

custom control

A user-designed control that directly inherits *System*.*Control*.

D

data binding

A way to link data (such as a data set) in your application to the properties of a control.

data consumer

This document is created with the unregistered version of CHM2PDF Pilot

A control that is bound to a data source.

Data Provider

A set of classes that work together to provide managed data access.

DataSet

A class that contains an application copy of data that has been retrieved from a data source.

delegate

An object that acts as a type-safe function pointer. Delegates can be used to invoke the method they point to.

deploying

Installing an application on the computer where it will run.

derived class

A class that is based on another class (called a base class) through inheritance. A derived class inherits the members of its base class and can override or shadow those members.

design time

Represents the state of the application while being designed. Applications at design time consist of noncompiled code that can be manipulated in the code editor or designer window.

destructor

The method that is run just before a class is removed from memory. Generally, a destructor contains any code required to "clean up" after the class. *See also* finalizer.

disconnected data access

Data access that uses a data connection only when executing database operations. This is the default kind of data access implemented by ADO.NET.

This document is created with the unregistered version of CHM2PDF Pilot

docking

In Windows Forms, attaching a control to the edge of the form. You can dock a control by setting the *Dock* property.

E

encapsulation

In component programming, separating the implementation of a component from the interface. Only the public interface of a component is made accessible to the rest of the application. Component data should never be accessible to outside callers.

event

A notification raised by a control or component that a specified condition has occurred.

event handler

A method that is executed when a particular event is raised.

exception

Any class that derives from *System.Exception*. Exceptions are thrown in response to run-time errors. *See also* throwing.

exception handling

The process of dealing with unusual occurrences within code so that it does not cause the program to crash or lose data.

explicit conversion

A conversion of one type to another that cannot be performed automatically. An explicit conversion usually presents some danger of a failed conversion or a loss of data.

extender provider

This document is created with the unregistered version of CHM2PDF Pilot

A specialized type of control that provides design time properties to other controls. *ErrorProvider* and *HelpProvider* are examples of extender providers.

F

field

A member variable of a class or structure.

finalizer

In Visual Basic .NET, the method that is called just before an instance of a class is destroyed. *See also* destructor.

floating-point numbers

A number that has a decimal component. *See also* floating-point types.

floating-point types

Types capable of representing floating-point numbers. Floating-point types include Single (float) and Double (double). *See also* floating-point numbers.

focus

When referring to a control, the condition of being able to receive user input through the keyboard or other input devices. A control that is capable of receiving input is said to have the focus.

foreign key

In a data table, a column that contains a key value that uniquely identifies a record in a related table.

form

The primary unit of the user interface in a Windows Forms application. A form serves as a nexus for data display and user input, and can host controls.

This document is created with the unregistered version of CHM2PDF Pilot

G

GAC

See Global Assembly Cache (GAC).

garbage collection

Automatic memory management provided by the common language runtime. Unused memory is automatically reclaimed by garbage collection without interaction with the application.

GDI+

See Graphic Device Interface (GDI).

getter

The code segment of a property that returns the value the property contains.

Global Assembly Cache (GAC)

A machine-wide cache that stores assemblies that have been specifically designated to be shared between several applications.

globalization

Applying culture-based formatting to existing data, for example, using culture-specific currency symbols and formatting.

Graphic Device Interface (GDI)

In Windows, a graphics display system used by applications to display or print bitmapped text (TrueType fonts), images, and other graphical elements. GDI+ is the name given to the .NET Framework's managed implementation of the GDI.

I

This document is created with the unregistered version of CHM2PDF Pilot

implicit conversion

A conversion between types that can be performed automatically by the common language runtime. An implicit conversion will always succeed and never pose a danger of data loss.

inheritance

In object-oriented programming, creating a new class that incorporates and extends all of the characteristics and functionality of a base class.

inherited control

A user-created control that inherits an existing control. An inherited control inherits all of the functionality of the base control and can incorporate custom functionality as well.

interface

A contract for behavior that can be implemented by a class. Classes that implement the same interface can be expected to provide the members defined by the interface.

L

localization

Providing multiple sets of data specific to different cultures and retrieving the data appropriate to the current culture setting.

logical error

An error that compiles and executes correctly but produces unexpected results. These can be the most difficult errors to detect in application testing.

M

This document is created with the unregistered version of CHM2PDF Pilot

managed code

Code that runs under the common language runtime. The common language runtime handles many tasks that would formerly have been handled in the application's executable. Managed code solves the Windows programming problems of component registration and versioning (sometimes called DLL Hell) because managed code contains all the versioning and type information that the common language runtime needs to run the application. The common language runtime handles registration dynamically at run time rather than statically through the system registry, as is done with applications based on the Component Object Model (COM).

metadata

Data about data. For example, the name, version, security information, and size of a file constitute metadata about the file.

Microsoft Intermediate Language (MSIL)

A low-level language that is just-in-time compiled to native code at run time. All .NET assemblies are represented in the MSIL.

MSIL

See Microsoft Intermediate Language (MSIL).

multidimensional array

An array that contains more than one dimension.

N

namespace

A logical organization of types that perform related functions.

.NET Framework

This document is created with the unregistered version of CHM2PDF Pilot

A managed environment for the development and execution of code. It consists of the common language runtime and .NET Framework base class library.

.NET Framework base class library

A collection of object-oriented types and interfaces that provides object models to assist in the development of your applications.

O

object

The base class for all types in the .NET Framework. More generically, any instance of a class or structure.

object-oriented programming

The process of creating applications by using programmatic constructs to represent real-world objects.

operator overloading

In Visual C#, defining custom behavior for an operator when interacting with custom types.

optimization

The process of examining your application and modifying it to be more efficient in terms of either speed or size of the deployed application.

overloading

Creating multiple methods with the same name but different signatures.

overriding

When inheriting from a base class, providing a new method for one of the base class methods.

P

This document is created with the unregistered version of CHM2PDF Pilot

parent-child relationship

In data access, a relationship between two tables wherein a record from the parent table can be used to retrieve one or more related records from a child table. A parent-child relationship can be defined by using a *DataRelation* object.

polymorphism

The ability of classes to provide different implementations of the same public interface. Thus, two classes that exhibit polymorphism might contain different implementations, but because the interfaces are identical, they can be treated the same in code.

primary key

In a data table, the column that uniquely identifies a record in the table.

private assembly

An assembly that is used by only one application. In .NET Framework programming, most assemblies are private assemblies.

project

The collection of Visual Studio .NET source files that make up an application.

R

Reference type

A type, such as a class, whose instance data is allocated on the heap. Variables that represent Reference types contain a reference to the object on the heap as opposed to the object's data itself. *See also* Value type.

registry

The system repository for information about a computer's settings. The registry can contain configuration information

This document is created with the unregistered version of CHM2PDF Pilot

about programs and system components.

resource

Nonexecutable data, such as a string or image, that is deployed in an application.

role-based security

Code that ensures that users of your application have appropriate permission to access your application based on the identity of the user.

run time

Represents the state of the application while executing. At run time, the application consists of executing code.

run-time error

An error that causes an invalid result or attempts an invalid operation at run time.

S

satellite assembly

When localizing an application, assemblies that contain alternate sets of resources to be used in the application for different cultures.

setter

The code segment of a property that sets the property's value.

shadowing

Hiding a member in a base class with a member in an inherited class that has a different signature.

shared assembly

This document is created with the unregistered version of CHM2PDF Pilot

An assembly that is referenced by several applications on a single machine. A shared assembly must be installed to the Global Assembly Cache (GAC).

shared members

In Visual Basic .NET, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Shared members are called static members in Visual C#.

simple binding

In data binding, binding a control to a single data column. A *TextBox* is an example of a control that can be simply bound.

solution

A group of Visual Studio .NET projects that make up a single functional unit.

SQL

See Structured Query Language (SQL).

static members

In Visual C#, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Static members are called shared members in Visual Basic .NET.

Step Into

Moving from a calling procedure to a called procedure during debugging. Also used in reference to setting breakpoints to stop execution in a specific procedure during debugging.

Step Out

In Break mode, resuming execution of a function call until the call returns.

Step Over

In Break mode, executing a procedure call as a single statement during debugging.

This document is created with the unregistered version of CHM2PDF Pilot

strong name

A name that uniquely identifies an assembly. It consists of the assembly's name, version, and culture information as well as a public key encrypted by a private key. An assembly must have a strong name to be installed to the Global Assembly Cache (GAC).

strongly typed

Refers to types for all variables being explicitly declared. Strong typing eliminates the possibility of type mismatch errors, which are discovered at compile time. Visual C# is inherently strongly typed, and Visual Basic .NET is strongly typed when Option Strict is set to On.

Struct

In Visual C#, a user-defined Value type that serves as a template for an instance of that type. *See also* Structure.

Structure

In Visual Basic .NET, a user-defined Value type that serves as a template for an instance of that type. *See also* Struct .

structured exception handling

Using *Try...Catch...Finally* blocks to handle errors that might occur during code execution. When exceptions are thrown, they are handled in the *Catch* block before code in the *Finally* block is executed.

Structured Query Language (SQL)

The standard language used for issuing commands to databases.

syntax error

An error that occurs because of typographical errors or nonsensical code. In the .NET Framework, syntax errors prevent compilation and are thus discovered at compile time.

T

target audience

This document is created with the unregistered version of CHM2PDF Pilot

The group of end users for whom an application is designed.

threads

The basic unit to which the server allocates processor time. A single process can have multiple threads.

throwing

The act of passing an exception to the next caller in the call stack. *See also* exception.

tracing

A programming technique for recording events, such as exceptions, in an application. Tracing is used during debugging and in the testing phase of application deployment.

transaction

A group of commands (treated as a single unit) that change the data stored in a database. The transaction ensures that the commands are handled in an all-or-nothing fashion if one of the commands fails, all of the commands fail, and any data that was written to the database by the commands is backed out. In this way, transactions maintain the integrity of data in a database.

tuning

The process of making adjustments to a deployed application that do not affect code.

U

Unicode

The universal character encoding scheme for characters and text. Each character is represented by a unique numeric value.

unmanaged code

Code that is not managed by the common language runtime. Unmanaged code is not checked for type-safety and

This document is created with the unregistered version of CHM2PDF Pilot

must be used with extreme care.

user control

A user-defined control that encapsulates other Windows Forms controls with custom functionality in a common container.

V

Value type

A type, such as an Integer, Decimal, or Structure, whose instance data is allocated on the stack. Variables that represent Value types contain the object's instance data itself as opposed to a reference to that data. *See also* Reference type.

W

Watch window

When debugging, the window that allows you to examine the values of predetermined watch variables.

Web Service

A service or component that is implemented by a Web server and can be instantiated and accessed programmatically as a component over the World Wide Web.

X

XML Schema

A description of the data elements contained in an XML file. The XML Schema provides the names of the elements, their types, whether or not they are key fields, and other information.

This document is created with the unregistered version of CHM2PDF Pilot

Glossary

A

abstract class

A class that cannot be instantiated but is used as a base from which other classes can be derived. In Microsoft Visual Basic .NET, abstract classes are declared using the *MustInherit* keyword. In Microsoft Visual C#, abstract classes are declared using the *abstract* keyword.

abstraction

Representing real-world objects as programmatic constructs.

abstract members

A member of a base class that cannot be invoked, but instead provides a template for members of a derived class. In Visual Basic .NET, abstract members are declared using the *MustOverride* keyword. In Visual C#, abstract members are declared using the *abstract* keyword.

ADO.NET

The data access architecture for the Microsoft .NET Framework. ADO.NET is built around a disconnected data access model that uses a set of classes called a Data Provider to retrieve data from a data source.

alias

The name you use to identify resources on the Web. Aliases represent physical resources on the Web server, such as a Web form, an HTML page, or a graphic.

anchoring

In Windows Forms controls, the resizing and repositioning of a control when its parent form is resized. You can anchor a control by setting the *Anchor* property.

This document is created with the unregistered version of CHM2PDF Pilot

assembly

The primary unit of deployment for a .NET application. An assembly is either an executable application (EXE) or a class library (DLL).

assembly manifest

The metadata for the assembly. It contains all of the information needed to describe the assembly to the common language runtime.

authentication

The process of determining the identity of a user. In effect, authentication validates that the users are who they say they are. *See also* authorization.

authorization

The process of allowing access to an application or resource based on credentials supplied by the user. *See also* authentication.

B

base class

A class that provides properties and methods as a foundation for a derived class. In objected-oriented programming, one class can be based on another through inheritance. Using this technique, the base class provides characteristics (such as properties and methods) to a derived class. The derived class can reuse, modify, or add to the members of the base class.

Bezier

A curve specified by four points. Two points specify the ends of the curve, and the other two points influence the shape of the curve between the end points.

Boolean

A data type that can represent a value of either true or false.

This document is created with the unregistered version of CHM2PDF Pilot

bootstrapper

In deployment, an application that installs Microsoft Windows Installer on machines that do not currently have it installed.

boundary condition

In testing, test conditions at the limit of normal operation and just over or under those limits.

boxing

The act of wrapping a Value type, such as an Integer, inside an object so that it can be treated like a Reference type.

Break mode

In debugging, a state of suspended execution wherein the application can be executed in a line-by-line manner, and individual program variables can be examined.

breakpoint

In debugging, a predetermined point where execution of an application will halt and enter Break mode.

C

cast

An explicit conversion from one type to another.

circular reference

A condition that occurs when two objects refer to each other but do not have an external reference. Circular references are automatically detected and disposed of by the garbage collector.

class

A user-defined Reference type that serves as a template for an object of that type.

This document is created with the unregistered version of CHM2PDF Pilot

CLR

See common language runtime (CLR).

code access security

Verification that all callers to code have appropriate permission to access it.

code tuning

The act of examining application code and optimizing for faster execution. Code tuning is usually an iterative process that involves several successive rounds of planning, improvement, and assessment of the results.

common language runtime (CLR)

The environment in which managed code executes. The common language runtime provides just-in-time compilation, enforces type safety, and manages memory through garbage collection.

Common Type System

A set of types that are used by all .NET languages, thus ensuring .NET language type compatibility.

complex binding

In data binding, binding more than one column to a single control. The *DataGrid* is an example of a control that can be complexly bound.

component

Strictly speaking, a class that implements *IComponent* or inherits *System*.*Component*. More generally, any class that provides a reusable set of functionality.

connected data access

Data access that requires a data connection that is always open.

Constant

This document is created with the unregistered version of CHM2PDF Pilot

A type that represents a value that does not change in the course of an application's execution and cannot be changed.

constraint

In ADO.NET, an object that represents a rule that enforces referential integrity, such as a constraint that specifies a primary key.

constructor

The method that executes when a class is instantiated. The constructor usually contains initialization code to set member variables to initial values.

container controls

Controls that can host other controls. Examples include *Forms*, *Panels*, and *TabPages*.

control

A class that inherits *System*.*Control*. Controls usually have a visual representation, interact with the user as a part of the user interface, and are hosted in a container control.

controls collection

The collection of a container control that manages the controls it hosts.

custom control

A user-designed control that directly inherits *System*.*Control*.

D

data binding

A way to link data (such as a data set) in your application to the properties of a control.

data consumer

This document is created with the unregistered version of CHM2PDF Pilot

A control that is bound to a data source.

Data Provider

A set of classes that work together to provide managed data access.

DataSet

A class that contains an application copy of data that has been retrieved from a data source.

delegate

An object that acts as a type-safe function pointer. Delegates can be used to invoke the method they point to.

deploying

Installing an application on the computer where it will run.

derived class

A class that is based on another class (called a base class) through inheritance. A derived class inherits the members of its base class and can override or shadow those members.

design time

Represents the state of the application while being designed. Applications at design time consist of noncompiled code that can be manipulated in the code editor or designer window.

destructor

The method that is run just before a class is removed from memory. Generally, a destructor contains any code required to "clean up" after the class. *See also* finalizer.

disconnected data access

Data access that uses a data connection only when executing database operations. This is the default kind of data access implemented by ADO.NET.

This document is created with the unregistered version of CHM2PDF Pilot

docking

In Windows Forms, attaching a control to the edge of the form. You can dock a control by setting the *Dock* property.

E

encapsulation

In component programming, separating the implementation of a component from the interface. Only the public interface of a component is made accessible to the rest of the application. Component data should never be accessible to outside callers.

event

A notification raised by a control or component that a specified condition has occurred.

event handler

A method that is executed when a particular event is raised.

exception

Any class that derives from *System.Exception*. Exceptions are thrown in response to run-time errors. *See also* throwing.

exception handling

The process of dealing with unusual occurrences within code so that it does not cause the program to crash or lose data.

explicit conversion

A conversion of one type to another that cannot be performed automatically. An explicit conversion usually presents some danger of a failed conversion or a loss of data.

extender provider

This document is created with the unregistered version of CHM2PDF Pilot

A specialized type of control that provides design time properties to other controls. *ErrorProvider* and *HelpProvider* are examples of extender providers.

F

field

A member variable of a class or structure.

finalizer

In Visual Basic .NET, the method that is called just before an instance of a class is destroyed. *See also* destructor.

floating-point numbers

A number that has a decimal component. *See also* floating-point types.

floating-point types

Types capable of representing floating-point numbers. Floating-point types include Single (float) and Double (double). *See also* floating-point numbers.

focus

When referring to a control, the condition of being able to receive user input through the keyboard or other input devices. A control that is capable of receiving input is said to have the focus.

foreign key

In a data table, a column that contains a key value that uniquely identifies a record in a related table.

form

The primary unit of the user interface in a Windows Forms application. A form serves as a nexus for data display and user input, and can host controls.

This document is created with the unregistered version of CHM2PDF Pilot

G

GAC

See Global Assembly Cache (GAC).

garbage collection

Automatic memory management provided by the common language runtime. Unused memory is automatically reclaimed by garbage collection without interaction with the application.

GDI+

See Graphic Device Interface (GDI).

getter

The code segment of a property that returns the value the property contains.

Global Assembly Cache (GAC)

A machine-wide cache that stores assemblies that have been specifically designated to be shared between several applications.

globalization

Applying culture-based formatting to existing data, for example, using culture-specific currency symbols and formatting.

Graphic Device Interface (GDI)

In Windows, a graphics display system used by applications to display or print bitmapped text (TrueType fonts), images, and other graphical elements. GDI+ is the name given to the .NET Framework's managed implementation of the GDI.

I

This document is created with the unregistered version of CHM2PDF Pilot

implicit conversion

A conversion between types that can be performed automatically by the common language runtime. An implicit conversion will always succeed and never pose a danger of data loss.

inheritance

In object-oriented programming, creating a new class that incorporates and extends all of the characteristics and functionality of a base class.

inherited control

A user-created control that inherits an existing control. An inherited control inherits all of the functionality of the base control and can incorporate custom functionality as well.

interface

A contract for behavior that can be implemented by a class. Classes that implement the same interface can be expected to provide the members defined by the interface.

L

localization

Providing multiple sets of data specific to different cultures and retrieving the data appropriate to the current culture setting.

logical error

An error that compiles and executes correctly but produces unexpected results. These can be the most difficult errors to detect in application testing.

M

This document is created with the unregistered version of CHM2PDF Pilot

managed code

Code that runs under the common language runtime. The common language runtime handles many tasks that would formerly have been handled in the application's executable. Managed code solves the Windows programming problems of component registration and versioning (sometimes called DLL Hell) because managed code contains all the versioning and type information that the common language runtime needs to run the application. The common language runtime handles registration dynamically at run time rather than statically through the system registry, as is done with applications based on the Component Object Model (COM).

metadata

Data about data. For example, the name, version, security information, and size of a file constitute metadata about the file.

Microsoft Intermediate Language (MSIL)

A low-level language that is just-in-time compiled to native code at run time. All .NET assemblies are represented in the MSIL.

MSIL

See Microsoft Intermediate Language (MSIL).

multidimensional array

An array that contains more than one dimension.

N

namespace

A logical organization of types that perform related functions.

.NET Framework

This document is created with the unregistered version of CHM2PDF Pilot

A managed environment for the development and execution of code. It consists of the common language runtime and .NET Framework base class library.

.NET Framework base class library

A collection of object-oriented types and interfaces that provides object models to assist in the development of your applications.

O

object

The base class for all types in the .NET Framework. More generically, any instance of a class or structure.

object-oriented programming

The process of creating applications by using programmatic constructs to represent real-world objects.

operator overloading

In Visual C#, defining custom behavior for an operator when interacting with custom types.

optimization

The process of examining your application and modifying it to be more efficient in terms of either speed or size of the deployed application.

overloading

Creating multiple methods with the same name but different signatures.

overriding

When inheriting from a base class, providing a new method for one of the base class methods.

P

This document is created with the unregistered version of CHM2PDF Pilot

parent-child relationship

In data access, a relationship between two tables wherein a record from the parent table can be used to retrieve one or more related records from a child table. A parent-child relationship can be defined by using a *DataRelation* object.

polymorphism

The ability of classes to provide different implementations of the same public interface. Thus, two classes that exhibit polymorphism might contain different implementations, but because the interfaces are identical, they can be treated the same in code.

primary key

In a data table, the column that uniquely identifies a record in the table.

private assembly

An assembly that is used by only one application. In .NET Framework programming, most assemblies are private assemblies.

project

The collection of Visual Studio .NET source files that make up an application.

R

Reference type

A type, such as a class, whose instance data is allocated on the heap. Variables that represent Reference types contain a reference to the object on the heap as opposed to the object's data itself. *See also* Value type.

registry

The system repository for information about a computer's settings. The registry can contain configuration information

This document is created with the unregistered version of CHM2PDF Pilot

about programs and system components.

resource

Nonexecutable data, such as a string or image, that is deployed in an application.

role-based security

Code that ensures that users of your application have appropriate permission to access your application based on the identity of the user.

run time

Represents the state of the application while executing. At run time, the application consists of executing code.

run-time error

An error that causes an invalid result or attempts an invalid operation at run time.

S

satellite assembly

When localizing an application, assemblies that contain alternate sets of resources to be used in the application for different cultures.

setter

The code segment of a property that sets the property's value.

shadowing

Hiding a member in a base class with a member in an inherited class that has a different signature.

shared assembly

This document is created with the unregistered version of CHM2PDF Pilot

An assembly that is referenced by several applications on a single machine. A shared assembly must be installed to the Global Assembly Cache (GAC).

shared members

In Visual Basic .NET, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Shared members are called static members in Visual C#.

simple binding

In data binding, binding a control to a single data column. A *TextBox* is an example of a control that can be simply bound.

solution

A group of Visual Studio .NET projects that make up a single functional unit.

SQL

See Structured Query Language (SQL).

static members

In Visual C#, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Static members are called shared members in Visual Basic .NET.

Step Into

Moving from a calling procedure to a called procedure during debugging. Also used in reference to setting breakpoints to stop execution in a specific procedure during debugging.

Step Out

In Break mode, resuming execution of a function call until the call returns.

Step Over

In Break mode, executing a procedure call as a single statement during debugging.

This document is created with the unregistered version of CHM2PDF Pilot

strong name

A name that uniquely identifies an assembly. It consists of the assembly's name, version, and culture information as well as a public key encrypted by a private key. An assembly must have a strong name to be installed to the Global Assembly Cache (GAC).

strongly typed

Refers to types for all variables being explicitly declared. Strong typing eliminates the possibility of type mismatch errors, which are discovered at compile time. Visual C# is inherently strongly typed, and Visual Basic .NET is strongly typed when Option Strict is set to On.

Struct

In Visual C#, a user-defined Value type that serves as a template for an instance of that type. *See also* Structure.

Structure

In Visual Basic .NET, a user-defined Value type that serves as a template for an instance of that type. *See also* Struct .

structured exception handling

Using *Try...Catch...Finally* blocks to handle errors that might occur during code execution. When exceptions are thrown, they are handled in the *Catch* block before code in the *Finally* block is executed.

Structured Query Language (SQL)

The standard language used for issuing commands to databases.

syntax error

An error that occurs because of typographical errors or nonsensical code. In the .NET Framework, syntax errors prevent compilation and are thus discovered at compile time.

T

target audience

This document is created with the unregistered version of CHM2PDF Pilot

The group of end users for whom an application is designed.

threads

The basic unit to which the server allocates processor time. A single process can have multiple threads.

throwing

The act of passing an exception to the next caller in the call stack. *See also* exception.

tracing

A programming technique for recording events, such as exceptions, in an application. Tracing is used during debugging and in the testing phase of application deployment.

transaction

A group of commands (treated as a single unit) that change the data stored in a database. The transaction ensures that the commands are handled in an all-or-nothing fashion if one of the commands fails, all of the commands fail, and any data that was written to the database by the commands is backed out. In this way, transactions maintain the integrity of data in a database.

tuning

The process of making adjustments to a deployed application that do not affect code.

U

Unicode

The universal character encoding scheme for characters and text. Each character is represented by a unique numeric value.

unmanaged code

Code that is not managed by the common language runtime. Unmanaged code is not checked for type-safety and

This document is created with the unregistered version of CHM2PDF Pilot

must be used with extreme care.

user control

A user-defined control that encapsulates other Windows Forms controls with custom functionality in a common container.

V

Value type

A type, such as an Integer, Decimal, or Structure, whose instance data is allocated on the stack. Variables that represent Value types contain the object's instance data itself as opposed to a reference to that data. *See also* Reference type.

W

Watch window

When debugging, the window that allows you to examine the values of predetermined watch variables.

Web Service

A service or component that is implemented by a Web server and can be instantiated and accessed programmatically as a component over the World Wide Web.

X

XML Schema

A description of the data elements contained in an XML file. The XML Schema provides the names of the elements, their types, whether or not they are key fields, and other information.

This document is created with the unregistered version of CHM2PDF Pilot

Glossary

A

abstract class

A class that cannot be instantiated but is used as a base from which other classes can be derived. In Microsoft Visual Basic .NET, abstract classes are declared using the *MustInherit* keyword. In Microsoft Visual C#, abstract classes are declared using the *abstract* keyword.

abstraction

Representing real-world objects as programmatic constructs.

abstract members

A member of a base class that cannot be invoked, but instead provides a template for members of a derived class. In Visual Basic .NET, abstract members are declared using the *MustOverride* keyword. In Visual C#, abstract members are declared using the *abstract* keyword.

ADO.NET

The data access architecture for the Microsoft .NET Framework. ADO.NET is built around a disconnected data access model that uses a set of classes called a Data Provider to retrieve data from a data source.

alias

The name you use to identify resources on the Web. Aliases represent physical resources on the Web server, such as a Web form, an HTML page, or a graphic.

anchoring

In Windows Forms controls, the resizing and repositioning of a control when its parent form is resized. You can anchor a control by setting the *Anchor* property.

This document is created with the unregistered version of CHM2PDF Pilot

assembly

The primary unit of deployment for a .NET application. An assembly is either an executable application (EXE) or a class library (DLL).

assembly manifest

The metadata for the assembly. It contains all of the information needed to describe the assembly to the common language runtime.

authentication

The process of determining the identity of a user. In effect, authentication validates that the users are who they say they are. *See also* authorization.

authorization

The process of allowing access to an application or resource based on credentials supplied by the user. *See also* authentication.

B

base class

A class that provides properties and methods as a foundation for a derived class. In objected-oriented programming, one class can be based on another through inheritance. Using this technique, the base class provides characteristics (such as properties and methods) to a derived class. The derived class can reuse, modify, or add to the members of the base class.

Bezier

A curve specified by four points. Two points specify the ends of the curve, and the other two points influence the shape of the curve between the end points.

Boolean

A data type that can represent a value of either true or false.

This document is created with the unregistered version of CHM2PDF Pilot

bootstrapper

In deployment, an application that installs Microsoft Windows Installer on machines that do not currently have it installed.

boundary condition

In testing, test conditions at the limit of normal operation and just over or under those limits.

boxing

The act of wrapping a Value type, such as an Integer, inside an object so that it can be treated like a Reference type.

Break mode

In debugging, a state of suspended execution wherein the application can be executed in a line-by-line manner, and individual program variables can be examined.

breakpoint

In debugging, a predetermined point where execution of an application will halt and enter Break mode.

C

cast

An explicit conversion from one type to another.

circular reference

A condition that occurs when two objects refer to each other but do not have an external reference. Circular references are automatically detected and disposed of by the garbage collector.

class

A user-defined Reference type that serves as a template for an object of that type.

This document is created with the unregistered version of CHM2PDF Pilot

CLR

See common language runtime (CLR).

code access security

Verification that all callers to code have appropriate permission to access it.

code tuning

The act of examining application code and optimizing for faster execution. Code tuning is usually an iterative process that involves several successive rounds of planning, improvement, and assessment of the results.

common language runtime (CLR)

The environment in which managed code executes. The common language runtime provides just-in-time compilation, enforces type safety, and manages memory through garbage collection.

Common Type System

A set of types that are used by all .NET languages, thus ensuring .NET language type compatibility.

complex binding

In data binding, binding more than one column to a single control. The *DataGrid* is an example of a control that can be complexly bound.

component

Strictly speaking, a class that implements *IComponent* or inherits *System*.*Component*. More generally, any class that provides a reusable set of functionality.

connected data access

Data access that requires a data connection that is always open.

Constant

This document is created with the unregistered version of CHM2PDF Pilot

A type that represents a value that does not change in the course of an application's execution and cannot be changed.

constraint

In ADO.NET, an object that represents a rule that enforces referential integrity, such as a constraint that specifies a primary key.

constructor

The method that executes when a class is instantiated. The constructor usually contains initialization code to set member variables to initial values.

container controls

Controls that can host other controls. Examples include *Forms*, *Panels*, and *TabPages*.

control

A class that inherits *System*.*Control*. Controls usually have a visual representation, interact with the user as a part of the user interface, and are hosted in a container control.

controls collection

The collection of a container control that manages the controls it hosts.

custom control

A user-designed control that directly inherits *System*.*Control*.

D

data binding

A way to link data (such as a data set) in your application to the properties of a control.

data consumer

This document is created with the unregistered version of CHM2PDF Pilot

A control that is bound to a data source.

Data Provider

A set of classes that work together to provide managed data access.

DataSet

A class that contains an application copy of data that has been retrieved from a data source.

delegate

An object that acts as a type-safe function pointer. Delegates can be used to invoke the method they point to.

deploying

Installing an application on the computer where it will run.

derived class

A class that is based on another class (called a base class) through inheritance. A derived class inherits the members of its base class and can override or shadow those members.

design time

Represents the state of the application while being designed. Applications at design time consist of noncompiled code that can be manipulated in the code editor or designer window.

destructor

The method that is run just before a class is removed from memory. Generally, a destructor contains any code required to "clean up" after the class. *See also* finalizer.

disconnected data access

Data access that uses a data connection only when executing database operations. This is the default kind of data access implemented by ADO.NET.

This document is created with the unregistered version of CHM2PDF Pilot

docking

In Windows Forms, attaching a control to the edge of the form. You can dock a control by setting the *Dock* property.

E

encapsulation

In component programming, separating the implementation of a component from the interface. Only the public interface of a component is made accessible to the rest of the application. Component data should never be accessible to outside callers.

event

A notification raised by a control or component that a specified condition has occurred.

event handler

A method that is executed when a particular event is raised.

exception

Any class that derives from *System.Exception*. Exceptions are thrown in response to run-time errors. *See also* throwing.

exception handling

The process of dealing with unusual occurrences within code so that it does not cause the program to crash or lose data.

explicit conversion

A conversion of one type to another that cannot be performed automatically. An explicit conversion usually presents some danger of a failed conversion or a loss of data.

extender provider

This document is created with the unregistered version of CHM2PDF Pilot

A specialized type of control that provides design time properties to other controls. *ErrorProvider* and *HelpProvider* are examples of extender providers.

F

field

A member variable of a class or structure.

finalizer

In Visual Basic .NET, the method that is called just before an instance of a class is destroyed. *See also* destructor.

floating-point numbers

A number that has a decimal component. *See also* floating-point types.

floating-point types

Types capable of representing floating-point numbers. Floating-point types include Single (float) and Double (double). *See also* floating-point numbers.

focus

When referring to a control, the condition of being able to receive user input through the keyboard or other input devices. A control that is capable of receiving input is said to have the focus.

foreign key

In a data table, a column that contains a key value that uniquely identifies a record in a related table.

form

The primary unit of the user interface in a Windows Forms application. A form serves as a nexus for data display and user input, and can host controls.

This document is created with the unregistered version of CHM2PDF Pilot

G

GAC

See Global Assembly Cache (GAC).

garbage collection

Automatic memory management provided by the common language runtime. Unused memory is automatically reclaimed by garbage collection without interaction with the application.

GDI+

See Graphic Device Interface (GDI).

getter

The code segment of a property that returns the value the property contains.

Global Assembly Cache (GAC)

A machine-wide cache that stores assemblies that have been specifically designated to be shared between several applications.

globalization

Applying culture-based formatting to existing data, for example, using culture-specific currency symbols and formatting.

Graphic Device Interface (GDI)

In Windows, a graphics display system used by applications to display or print bitmapped text (TrueType fonts), images, and other graphical elements. GDI+ is the name given to the .NET Framework's managed implementation of the GDI.

I

This document is created with the unregistered version of CHM2PDF Pilot

implicit conversion

A conversion between types that can be performed automatically by the common language runtime. An implicit conversion will always succeed and never pose a danger of data loss.

inheritance

In object-oriented programming, creating a new class that incorporates and extends all of the characteristics and functionality of a base class.

inherited control

A user-created control that inherits an existing control. An inherited control inherits all of the functionality of the base control and can incorporate custom functionality as well.

interface

A contract for behavior that can be implemented by a class. Classes that implement the same interface can be expected to provide the members defined by the interface.

L

localization

Providing multiple sets of data specific to different cultures and retrieving the data appropriate to the current culture setting.

logical error

An error that compiles and executes correctly but produces unexpected results. These can be the most difficult errors to detect in application testing.

M

This document is created with the unregistered version of CHM2PDF Pilot

managed code

Code that runs under the common language runtime. The common language runtime handles many tasks that would formerly have been handled in the application's executable. Managed code solves the Windows programming problems of component registration and versioning (sometimes called DLL Hell) because managed code contains all the versioning and type information that the common language runtime needs to run the application. The common language runtime handles registration dynamically at run time rather than statically through the system registry, as is done with applications based on the Component Object Model (COM).

metadata

Data about data. For example, the name, version, security information, and size of a file constitute metadata about the file.

Microsoft Intermediate Language (MSIL)

A low-level language that is just-in-time compiled to native code at run time. All .NET assemblies are represented in the MSIL.

MSIL

See Microsoft Intermediate Language (MSIL).

multidimensional array

An array that contains more than one dimension.

N

namespace

A logical organization of types that perform related functions.

.NET Framework

This document is created with the unregistered version of CHM2PDF Pilot

A managed environment for the development and execution of code. It consists of the common language runtime and .NET Framework base class library.

.NET Framework base class library

A collection of object-oriented types and interfaces that provides object models to assist in the development of your applications.

O

object

The base class for all types in the .NET Framework. More generically, any instance of a class or structure.

object-oriented programming

The process of creating applications by using programmatic constructs to represent real-world objects.

operator overloading

In Visual C#, defining custom behavior for an operator when interacting with custom types.

optimization

The process of examining your application and modifying it to be more efficient in terms of either speed or size of the deployed application.

overloading

Creating multiple methods with the same name but different signatures.

overriding

When inheriting from a base class, providing a new method for one of the base class methods.

P

This document is created with the unregistered version of CHM2PDF Pilot

parent-child relationship

In data access, a relationship between two tables wherein a record from the parent table can be used to retrieve one or more related records from a child table. A parent-child relationship can be defined by using a *DataRelation* object.

polymorphism

The ability of classes to provide different implementations of the same public interface. Thus, two classes that exhibit polymorphism might contain different implementations, but because the interfaces are identical, they can be treated the same in code.

primary key

In a data table, the column that uniquely identifies a record in the table.

private assembly

An assembly that is used by only one application. In .NET Framework programming, most assemblies are private assemblies.

project

The collection of Visual Studio .NET source files that make up an application.

R

Reference type

A type, such as a class, whose instance data is allocated on the heap. Variables that represent Reference types contain a reference to the object on the heap as opposed to the object's data itself. *See also* Value type.

registry

The system repository for information about a computer's settings. The registry can contain configuration information

This document is created with the unregistered version of CHM2PDF Pilot

about programs and system components.

resource

Nonexecutable data, such as a string or image, that is deployed in an application.

role-based security

Code that ensures that users of your application have appropriate permission to access your application based on the identity of the user.

run time

Represents the state of the application while executing. At run time, the application consists of executing code.

run-time error

An error that causes an invalid result or attempts an invalid operation at run time.

S

satellite assembly

When localizing an application, assemblies that contain alternate sets of resources to be used in the application for different cultures.

setter

The code segment of a property that sets the property's value.

shadowing

Hiding a member in a base class with a member in an inherited class that has a different signature.

shared assembly

This document is created with the unregistered version of CHM2PDF Pilot

An assembly that is referenced by several applications on a single machine. A shared assembly must be installed to the Global Assembly Cache (GAC).

shared members

In Visual Basic .NET, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Shared members are called static members in Visual C#.

simple binding

In data binding, binding a control to a single data column. A *TextBox* is an example of a control that can be simply bound.

solution

A group of Visual Studio .NET projects that make up a single functional unit.

SQL

See Structured Query Language (SQL).

static members

In Visual C#, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Static members are called shared members in Visual Basic .NET.

Step Into

Moving from a calling procedure to a called procedure during debugging. Also used in reference to setting breakpoints to stop execution in a specific procedure during debugging.

Step Out

In Break mode, resuming execution of a function call until the call returns.

Step Over

In Break mode, executing a procedure call as a single statement during debugging.

This document is created with the unregistered version of CHM2PDF Pilot

strong name

A name that uniquely identifies an assembly. It consists of the assembly's name, version, and culture information as well as a public key encrypted by a private key. An assembly must have a strong name to be installed to the Global Assembly Cache (GAC).

strongly typed

Refers to types for all variables being explicitly declared. Strong typing eliminates the possibility of type mismatch errors, which are discovered at compile time. Visual C# is inherently strongly typed, and Visual Basic .NET is strongly typed when Option Strict is set to On.

Struct

In Visual C#, a user-defined Value type that serves as a template for an instance of that type. *See also* Structure.

Structure

In Visual Basic .NET, a user-defined Value type that serves as a template for an instance of that type. *See also* Struct .

structured exception handling

Using *Try...Catch...Finally* blocks to handle errors that might occur during code execution. When exceptions are thrown, they are handled in the *Catch* block before code in the *Finally* block is executed.

Structured Query Language (SQL)

The standard language used for issuing commands to databases.

syntax error

An error that occurs because of typographical errors or nonsensical code. In the .NET Framework, syntax errors prevent compilation and are thus discovered at compile time.

T

target audience

This document is created with the unregistered version of CHM2PDF Pilot

The group of end users for whom an application is designed.

threads

The basic unit to which the server allocates processor time. A single process can have multiple threads.

throwing

The act of passing an exception to the next caller in the call stack. *See also* exception.

tracing

A programming technique for recording events, such as exceptions, in an application. Tracing is used during debugging and in the testing phase of application deployment.

transaction

A group of commands (treated as a single unit) that change the data stored in a database. The transaction ensures that the commands are handled in an all-or-nothing fashion if one of the commands fails, all of the commands fail, and any data that was written to the database by the commands is backed out. In this way, transactions maintain the integrity of data in a database.

tuning

The process of making adjustments to a deployed application that do not affect code.

U

Unicode

The universal character encoding scheme for characters and text. Each character is represented by a unique numeric value.

unmanaged code

Code that is not managed by the common language runtime. Unmanaged code is not checked for type-safety and

This document is created with the unregistered version of CHM2PDF Pilot

must be used with extreme care.

user control

A user-defined control that encapsulates other Windows Forms controls with custom functionality in a common container.

V

Value type

A type, such as an Integer, Decimal, or Structure, whose instance data is allocated on the stack. Variables that represent Value types contain the object's instance data itself as opposed to a reference to that data. *See also* Reference type.

W

Watch window

When debugging, the window that allows you to examine the values of predetermined watch variables.

Web Service

A service or component that is implemented by a Web server and can be instantiated and accessed programmatically as a component over the World Wide Web.

X

XML Schema

A description of the data elements contained in an XML file. The XML Schema provides the names of the elements, their types, whether or not they are key fields, and other information.

This document is created with the unregistered version of CHM2PDF Pilot

Glossary

A

abstract class

A class that cannot be instantiated but is used as a base from which other classes can be derived. In Microsoft Visual Basic .NET, abstract classes are declared using the *MustInherit* keyword. In Microsoft Visual C#, abstract classes are declared using the *abstract* keyword.

abstraction

Representing real-world objects as programmatic constructs.

abstract members

A member of a base class that cannot be invoked, but instead provides a template for members of a derived class. In Visual Basic .NET, abstract members are declared using the *MustOverride* keyword. In Visual C#, abstract members are declared using the *abstract* keyword.

ADO.NET

The data access architecture for the Microsoft .NET Framework. ADO.NET is built around a disconnected data access model that uses a set of classes called a Data Provider to retrieve data from a data source.

alias

The name you use to identify resources on the Web. Aliases represent physical resources on the Web server, such as a Web form, an HTML page, or a graphic.

anchoring

In Windows Forms controls, the resizing and repositioning of a control when its parent form is resized. You can anchor a control by setting the *Anchor* property.

This document is created with the unregistered version of CHM2PDF Pilot

assembly

The primary unit of deployment for a .NET application. An assembly is either an executable application (EXE) or a class library (DLL).

assembly manifest

The metadata for the assembly. It contains all of the information needed to describe the assembly to the common language runtime.

authentication

The process of determining the identity of a user. In effect, authentication validates that the users are who they say they are. *See also* authorization.

authorization

The process of allowing access to an application or resource based on credentials supplied by the user. *See also* authentication.

B

base class

A class that provides properties and methods as a foundation for a derived class. In objected-oriented programming, one class can be based on another through inheritance. Using this technique, the base class provides characteristics (such as properties and methods) to a derived class. The derived class can reuse, modify, or add to the members of the base class.

Bezier

A curve specified by four points. Two points specify the ends of the curve, and the other two points influence the shape of the curve between the end points.

Boolean

A data type that can represent a value of either true or false.

This document is created with the unregistered version of CHM2PDF Pilot

bootstrapper

In deployment, an application that installs Microsoft Windows Installer on machines that do not currently have it installed.

boundary condition

In testing, test conditions at the limit of normal operation and just over or under those limits.

boxing

The act of wrapping a Value type, such as an Integer, inside an object so that it can be treated like a Reference type.

Break mode

In debugging, a state of suspended execution wherein the application can be executed in a line-by-line manner, and individual program variables can be examined.

breakpoint

In debugging, a predetermined point where execution of an application will halt and enter Break mode.

C

cast

An explicit conversion from one type to another.

circular reference

A condition that occurs when two objects refer to each other but do not have an external reference. Circular references are automatically detected and disposed of by the garbage collector.

class

A user-defined Reference type that serves as a template for an object of that type.

This document is created with the unregistered version of CHM2PDF Pilot

CLR

See common language runtime (CLR).

code access security

Verification that all callers to code have appropriate permission to access it.

code tuning

The act of examining application code and optimizing for faster execution. Code tuning is usually an iterative process that involves several successive rounds of planning, improvement, and assessment of the results.

common language runtime (CLR)

The environment in which managed code executes. The common language runtime provides just-in-time compilation, enforces type safety, and manages memory through garbage collection.

Common Type System

A set of types that are used by all .NET languages, thus ensuring .NET language type compatibility.

complex binding

In data binding, binding more than one column to a single control. The *DataGrid* is an example of a control that can be complexly bound.

component

Strictly speaking, a class that implements *IComponent* or inherits *System*.*Component*. More generally, any class that provides a reusable set of functionality.

connected data access

Data access that requires a data connection that is always open.

Constant

This document is created with the unregistered version of CHM2PDF Pilot

A type that represents a value that does not change in the course of an application's execution and cannot be changed.

constraint

In ADO.NET, an object that represents a rule that enforces referential integrity, such as a constraint that specifies a primary key.

constructor

The method that executes when a class is instantiated. The constructor usually contains initialization code to set member variables to initial values.

container controls

Controls that can host other controls. Examples include *Forms*, *Panels*, and *TabPages*.

control

A class that inherits *System*.*Control*. Controls usually have a visual representation, interact with the user as a part of the user interface, and are hosted in a container control.

controls collection

The collection of a container control that manages the controls it hosts.

custom control

A user-designed control that directly inherits *System*.*Control*.

D

data binding

A way to link data (such as a data set) in your application to the properties of a control.

data consumer

This document is created with the unregistered version of CHM2PDF Pilot

A control that is bound to a data source.

Data Provider

A set of classes that work together to provide managed data access.

DataSet

A class that contains an application copy of data that has been retrieved from a data source.

delegate

An object that acts as a type-safe function pointer. Delegates can be used to invoke the method they point to.

deploying

Installing an application on the computer where it will run.

derived class

A class that is based on another class (called a base class) through inheritance. A derived class inherits the members of its base class and can override or shadow those members.

design time

Represents the state of the application while being designed. Applications at design time consist of noncompiled code that can be manipulated in the code editor or designer window.

destructor

The method that is run just before a class is removed from memory. Generally, a destructor contains any code required to "clean up" after the class. *See also* finalizer.

disconnected data access

Data access that uses a data connection only when executing database operations. This is the default kind of data access implemented by ADO.NET.

This document is created with the unregistered version of CHM2PDF Pilot

docking

In Windows Forms, attaching a control to the edge of the form. You can dock a control by setting the *Dock* property.

E

encapsulation

In component programming, separating the implementation of a component from the interface. Only the public interface of a component is made accessible to the rest of the application. Component data should never be accessible to outside callers.

event

A notification raised by a control or component that a specified condition has occurred.

event handler

A method that is executed when a particular event is raised.

exception

Any class that derives from *System.Exception*. Exceptions are thrown in response to run-time errors. *See also* throwing.

exception handling

The process of dealing with unusual occurrences within code so that it does not cause the program to crash or lose data.

explicit conversion

A conversion of one type to another that cannot be performed automatically. An explicit conversion usually presents some danger of a failed conversion or a loss of data.

extender provider

This document is created with the unregistered version of CHM2PDF Pilot

A specialized type of control that provides design time properties to other controls. *ErrorProvider* and *HelpProvider* are examples of extender providers.

F

field

A member variable of a class or structure.

finalizer

In Visual Basic .NET, the method that is called just before an instance of a class is destroyed. *See also* destructor.

floating-point numbers

A number that has a decimal component. *See also* floating-point types.

floating-point types

Types capable of representing floating-point numbers. Floating-point types include Single (float) and Double (double). *See also* floating-point numbers.

focus

When referring to a control, the condition of being able to receive user input through the keyboard or other input devices. A control that is capable of receiving input is said to have the focus.

foreign key

In a data table, a column that contains a key value that uniquely identifies a record in a related table.

form

The primary unit of the user interface in a Windows Forms application. A form serves as a nexus for data display and user input, and can host controls.

This document is created with the unregistered version of CHM2PDF Pilot

G

GAC

See Global Assembly Cache (GAC).

garbage collection

Automatic memory management provided by the common language runtime. Unused memory is automatically reclaimed by garbage collection without interaction with the application.

GDI+

See Graphic Device Interface (GDI).

getter

The code segment of a property that returns the value the property contains.

Global Assembly Cache (GAC)

A machine-wide cache that stores assemblies that have been specifically designated to be shared between several applications.

globalization

Applying culture-based formatting to existing data, for example, using culture-specific currency symbols and formatting.

Graphic Device Interface (GDI)

In Windows, a graphics display system used by applications to display or print bitmapped text (TrueType fonts), images, and other graphical elements. GDI+ is the name given to the .NET Framework's managed implementation of the GDI.

I

This document is created with the unregistered version of CHM2PDF Pilot

implicit conversion

A conversion between types that can be performed automatically by the common language runtime. An implicit conversion will always succeed and never pose a danger of data loss.

inheritance

In object-oriented programming, creating a new class that incorporates and extends all of the characteristics and functionality of a base class.

inherited control

A user-created control that inherits an existing control. An inherited control inherits all of the functionality of the base control and can incorporate custom functionality as well.

interface

A contract for behavior that can be implemented by a class. Classes that implement the same interface can be expected to provide the members defined by the interface.

L

localization

Providing multiple sets of data specific to different cultures and retrieving the data appropriate to the current culture setting.

logical error

An error that compiles and executes correctly but produces unexpected results. These can be the most difficult errors to detect in application testing.

M

This document is created with the unregistered version of CHM2PDF Pilot

managed code

Code that runs under the common language runtime. The common language runtime handles many tasks that would formerly have been handled in the application's executable. Managed code solves the Windows programming problems of component registration and versioning (sometimes called DLL Hell) because managed code contains all the versioning and type information that the common language runtime needs to run the application. The common language runtime handles registration dynamically at run time rather than statically through the system registry, as is done with applications based on the Component Object Model (COM).

metadata

Data about data. For example, the name, version, security information, and size of a file constitute metadata about the file.

Microsoft Intermediate Language (MSIL)

A low-level language that is just-in-time compiled to native code at run time. All .NET assemblies are represented in the MSIL.

MSIL

See Microsoft Intermediate Language (MSIL).

multidimensional array

An array that contains more than one dimension.

N

namespace

A logical organization of types that perform related functions.

.NET Framework

This document is created with the unregistered version of CHM2PDF Pilot

A managed environment for the development and execution of code. It consists of the common language runtime and .NET Framework base class library.

.NET Framework base class library

A collection of object-oriented types and interfaces that provides object models to assist in the development of your applications.

O

object

The base class for all types in the .NET Framework. More generically, any instance of a class or structure.

object-oriented programming

The process of creating applications by using programmatic constructs to represent real-world objects.

operator overloading

In Visual C#, defining custom behavior for an operator when interacting with custom types.

optimization

The process of examining your application and modifying it to be more efficient in terms of either speed or size of the deployed application.

overloading

Creating multiple methods with the same name but different signatures.

overriding

When inheriting from a base class, providing a new method for one of the base class methods.

P

This document is created with the unregistered version of CHM2PDF Pilot

parent-child relationship

In data access, a relationship between two tables wherein a record from the parent table can be used to retrieve one or more related records from a child table. A parent-child relationship can be defined by using a *DataRelation* object.

polymorphism

The ability of classes to provide different implementations of the same public interface. Thus, two classes that exhibit polymorphism might contain different implementations, but because the interfaces are identical, they can be treated the same in code.

primary key

In a data table, the column that uniquely identifies a record in the table.

private assembly

An assembly that is used by only one application. In .NET Framework programming, most assemblies are private assemblies.

project

The collection of Visual Studio .NET source files that make up an application.

R

Reference type

A type, such as a class, whose instance data is allocated on the heap. Variables that represent Reference types contain a reference to the object on the heap as opposed to the object's data itself. *See also* Value type.

registry

The system repository for information about a computer's settings. The registry can contain configuration information

This document is created with the unregistered version of CHM2PDF Pilot

about programs and system components.

resource

Nonexecutable data, such as a string or image, that is deployed in an application.

role-based security

Code that ensures that users of your application have appropriate permission to access your application based on the identity of the user.

run time

Represents the state of the application while executing. At run time, the application consists of executing code.

run-time error

An error that causes an invalid result or attempts an invalid operation at run time.

S

satellite assembly

When localizing an application, assemblies that contain alternate sets of resources to be used in the application for different cultures.

setter

The code segment of a property that sets the property's value.

shadowing

Hiding a member in a base class with a member in an inherited class that has a different signature.

shared assembly

This document is created with the unregistered version of CHM2PDF Pilot

An assembly that is referenced by several applications on a single machine. A shared assembly must be installed to the Global Assembly Cache (GAC).

shared members

In Visual Basic .NET, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Shared members are called static members in Visual C#.

simple binding

In data binding, binding a control to a single data column. A *TextBox* is an example of a control that can be simply bound.

solution

A group of Visual Studio .NET projects that make up a single functional unit.

SQL

See Structured Query Language (SQL).

static members

In Visual C#, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Static members are called shared members in Visual Basic .NET.

Step Into

Moving from a calling procedure to a called procedure during debugging. Also used in reference to setting breakpoints to stop execution in a specific procedure during debugging.

Step Out

In Break mode, resuming execution of a function call until the call returns.

Step Over

In Break mode, executing a procedure call as a single statement during debugging.

This document is created with the unregistered version of CHM2PDF Pilot

strong name

A name that uniquely identifies an assembly. It consists of the assembly's name, version, and culture information as well as a public key encrypted by a private key. An assembly must have a strong name to be installed to the Global Assembly Cache (GAC).

strongly typed

Refers to types for all variables being explicitly declared. Strong typing eliminates the possibility of type mismatch errors, which are discovered at compile time. Visual C# is inherently strongly typed, and Visual Basic .NET is strongly typed when Option Strict is set to On.

Struct

In Visual C#, a user-defined Value type that serves as a template for an instance of that type. *See also* Structure.

Structure

In Visual Basic .NET, a user-defined Value type that serves as a template for an instance of that type. *See also* Struct .

structured exception handling

Using *Try...Catch...Finally* blocks to handle errors that might occur during code execution. When exceptions are thrown, they are handled in the *Catch* block before code in the *Finally* block is executed.

Structured Query Language (SQL)

The standard language used for issuing commands to databases.

syntax error

An error that occurs because of typographical errors or nonsensical code. In the .NET Framework, syntax errors prevent compilation and are thus discovered at compile time.

T

target audience

This document is created with the unregistered version of CHM2PDF Pilot

The group of end users for whom an application is designed.

threads

The basic unit to which the server allocates processor time. A single process can have multiple threads.

throwing

The act of passing an exception to the next caller in the call stack. *See also* exception.

tracing

A programming technique for recording events, such as exceptions, in an application. Tracing is used during debugging and in the testing phase of application deployment.

transaction

A group of commands (treated as a single unit) that change the data stored in a database. The transaction ensures that the commands are handled in an all-or-nothing fashion if one of the commands fails, all of the commands fail, and any data that was written to the database by the commands is backed out. In this way, transactions maintain the integrity of data in a database.

tuning

The process of making adjustments to a deployed application that do not affect code.

U

Unicode

The universal character encoding scheme for characters and text. Each character is represented by a unique numeric value.

unmanaged code

Code that is not managed by the common language runtime. Unmanaged code is not checked for type-safety and

This document is created with the unregistered version of CHM2PDF Pilot

must be used with extreme care.

user control

A user-defined control that encapsulates other Windows Forms controls with custom functionality in a common container.

V

Value type

A type, such as an Integer, Decimal, or Structure, whose instance data is allocated on the stack. Variables that represent Value types contain the object's instance data itself as opposed to a reference to that data. *See also* Reference type.

W

Watch window

When debugging, the window that allows you to examine the values of predetermined watch variables.

Web Service

A service or component that is implemented by a Web server and can be instantiated and accessed programmatically as a component over the World Wide Web.

X

XML Schema

A description of the data elements contained in an XML file. The XML Schema provides the names of the elements, their types, whether or not they are key fields, and other information.

This document is created with the unregistered version of CHM2PDF Pilot

Glossary

A

abstract class

A class that cannot be instantiated but is used as a base from which other classes can be derived. In Microsoft Visual Basic .NET, abstract classes are declared using the *MustInherit* keyword. In Microsoft Visual C#, abstract classes are declared using the *abstract* keyword.

abstraction

Representing real-world objects as programmatic constructs.

abstract members

A member of a base class that cannot be invoked, but instead provides a template for members of a derived class. In Visual Basic .NET, abstract members are declared using the *MustOverride* keyword. In Visual C#, abstract members are declared using the *abstract* keyword.

ADO.NET

The data access architecture for the Microsoft .NET Framework. ADO.NET is built around a disconnected data access model that uses a set of classes called a Data Provider to retrieve data from a data source.

alias

The name you use to identify resources on the Web. Aliases represent physical resources on the Web server, such as a Web form, an HTML page, or a graphic.

anchoring

In Windows Forms controls, the resizing and repositioning of a control when its parent form is resized. You can anchor a control by setting the *Anchor* property.

This document is created with the unregistered version of CHM2PDF Pilot

assembly

The primary unit of deployment for a .NET application. An assembly is either an executable application (EXE) or a class library (DLL).

assembly manifest

The metadata for the assembly. It contains all of the information needed to describe the assembly to the common language runtime.

authentication

The process of determining the identity of a user. In effect, authentication validates that the users are who they say they are. *See also* authorization.

authorization

The process of allowing access to an application or resource based on credentials supplied by the user. *See also* authentication.

B

base class

A class that provides properties and methods as a foundation for a derived class. In objected-oriented programming, one class can be based on another through inheritance. Using this technique, the base class provides characteristics (such as properties and methods) to a derived class. The derived class can reuse, modify, or add to the members of the base class.

Bezier

A curve specified by four points. Two points specify the ends of the curve, and the other two points influence the shape of the curve between the end points.

Boolean

A data type that can represent a value of either true or false.

This document is created with the unregistered version of CHM2PDF Pilot

bootstrapper

In deployment, an application that installs Microsoft Windows Installer on machines that do not currently have it installed.

boundary condition

In testing, test conditions at the limit of normal operation and just over or under those limits.

boxing

The act of wrapping a Value type, such as an Integer, inside an object so that it can be treated like a Reference type.

Break mode

In debugging, a state of suspended execution wherein the application can be executed in a line-by-line manner, and individual program variables can be examined.

breakpoint

In debugging, a predetermined point where execution of an application will halt and enter Break mode.

C

cast

An explicit conversion from one type to another.

circular reference

A condition that occurs when two objects refer to each other but do not have an external reference. Circular references are automatically detected and disposed of by the garbage collector.

class

A user-defined Reference type that serves as a template for an object of that type.

This document is created with the unregistered version of CHM2PDF Pilot

CLR

See common language runtime (CLR).

code access security

Verification that all callers to code have appropriate permission to access it.

code tuning

The act of examining application code and optimizing for faster execution. Code tuning is usually an iterative process that involves several successive rounds of planning, improvement, and assessment of the results.

common language runtime (CLR)

The environment in which managed code executes. The common language runtime provides just-in-time compilation, enforces type safety, and manages memory through garbage collection.

Common Type System

A set of types that are used by all .NET languages, thus ensuring .NET language type compatibility.

complex binding

In data binding, binding more than one column to a single control. The *DataGrid* is an example of a control that can be complexly bound.

component

Strictly speaking, a class that implements *IComponent* or inherits *System*.*Component*. More generally, any class that provides a reusable set of functionality.

connected data access

Data access that requires a data connection that is always open.

Constant

This document is created with the unregistered version of CHM2PDF Pilot

A type that represents a value that does not change in the course of an application's execution and cannot be changed.

constraint

In ADO.NET, an object that represents a rule that enforces referential integrity, such as a constraint that specifies a primary key.

constructor

The method that executes when a class is instantiated. The constructor usually contains initialization code to set member variables to initial values.

container controls

Controls that can host other controls. Examples include *Forms*, *Panels*, and *TabPages*.

control

A class that inherits *System*.*Control*. Controls usually have a visual representation, interact with the user as a part of the user interface, and are hosted in a container control.

controls collection

The collection of a container control that manages the controls it hosts.

custom control

A user-designed control that directly inherits *System*.*Control*.

D

data binding

A way to link data (such as a data set) in your application to the properties of a control.

data consumer

This document is created with the unregistered version of CHM2PDF Pilot

A control that is bound to a data source.

Data Provider

A set of classes that work together to provide managed data access.

DataSet

A class that contains an application copy of data that has been retrieved from a data source.

delegate

An object that acts as a type-safe function pointer. Delegates can be used to invoke the method they point to.

deploying

Installing an application on the computer where it will run.

derived class

A class that is based on another class (called a base class) through inheritance. A derived class inherits the members of its base class and can override or shadow those members.

design time

Represents the state of the application while being designed. Applications at design time consist of noncompiled code that can be manipulated in the code editor or designer window.

destructor

The method that is run just before a class is removed from memory. Generally, a destructor contains any code required to "clean up" after the class. *See also* finalizer.

disconnected data access

Data access that uses a data connection only when executing database operations. This is the default kind of data access implemented by ADO.NET.

This document is created with the unregistered version of CHM2PDF Pilot

docking

In Windows Forms, attaching a control to the edge of the form. You can dock a control by setting the *Dock* property.

E

encapsulation

In component programming, separating the implementation of a component from the interface. Only the public interface of a component is made accessible to the rest of the application. Component data should never be accessible to outside callers.

event

A notification raised by a control or component that a specified condition has occurred.

event handler

A method that is executed when a particular event is raised.

exception

Any class that derives from *System.Exception*. Exceptions are thrown in response to run-time errors. *See also* throwing.

exception handling

The process of dealing with unusual occurrences within code so that it does not cause the program to crash or lose data.

explicit conversion

A conversion of one type to another that cannot be performed automatically. An explicit conversion usually presents some danger of a failed conversion or a loss of data.

extender provider

This document is created with the unregistered version of CHM2PDF Pilot

A specialized type of control that provides design time properties to other controls. *ErrorProvider* and *HelpProvider* are examples of extender providers.

F

field

A member variable of a class or structure.

finalizer

In Visual Basic .NET, the method that is called just before an instance of a class is destroyed. *See also* destructor.

floating-point numbers

A number that has a decimal component. *See also* floating-point types.

floating-point types

Types capable of representing floating-point numbers. Floating-point types include Single (float) and Double (double). *See also* floating-point numbers.

focus

When referring to a control, the condition of being able to receive user input through the keyboard or other input devices. A control that is capable of receiving input is said to have the focus.

foreign key

In a data table, a column that contains a key value that uniquely identifies a record in a related table.

form

The primary unit of the user interface in a Windows Forms application. A form serves as a nexus for data display and user input, and can host controls.

This document is created with the unregistered version of CHM2PDF Pilot

G

GAC

See Global Assembly Cache (GAC).

garbage collection

Automatic memory management provided by the common language runtime. Unused memory is automatically reclaimed by garbage collection without interaction with the application.

GDI+

See Graphic Device Interface (GDI).

getter

The code segment of a property that returns the value the property contains.

Global Assembly Cache (GAC)

A machine-wide cache that stores assemblies that have been specifically designated to be shared between several applications.

globalization

Applying culture-based formatting to existing data, for example, using culture-specific currency symbols and formatting.

Graphic Device Interface (GDI)

In Windows, a graphics display system used by applications to display or print bitmapped text (TrueType fonts), images, and other graphical elements. GDI+ is the name given to the .NET Framework's managed implementation of the GDI.

I

This document is created with the unregistered version of CHM2PDF Pilot

implicit conversion

A conversion between types that can be performed automatically by the common language runtime. An implicit conversion will always succeed and never pose a danger of data loss.

inheritance

In object-oriented programming, creating a new class that incorporates and extends all of the characteristics and functionality of a base class.

inherited control

A user-created control that inherits an existing control. An inherited control inherits all of the functionality of the base control and can incorporate custom functionality as well.

interface

A contract for behavior that can be implemented by a class. Classes that implement the same interface can be expected to provide the members defined by the interface.

L

localization

Providing multiple sets of data specific to different cultures and retrieving the data appropriate to the current culture setting.

logical error

An error that compiles and executes correctly but produces unexpected results. These can be the most difficult errors to detect in application testing.

M

This document is created with the unregistered version of CHM2PDF Pilot

managed code

Code that runs under the common language runtime. The common language runtime handles many tasks that would formerly have been handled in the application's executable. Managed code solves the Windows programming problems of component registration and versioning (sometimes called DLL Hell) because managed code contains all the versioning and type information that the common language runtime needs to run the application. The common language runtime handles registration dynamically at run time rather than statically through the system registry, as is done with applications based on the Component Object Model (COM).

metadata

Data about data. For example, the name, version, security information, and size of a file constitute metadata about the file.

Microsoft Intermediate Language (MSIL)

A low-level language that is just-in-time compiled to native code at run time. All .NET assemblies are represented in the MSIL.

MSIL

See Microsoft Intermediate Language (MSIL).

multidimensional array

An array that contains more than one dimension.

N

namespace

A logical organization of types that perform related functions.

.NET Framework

This document is created with the unregistered version of CHM2PDF Pilot

A managed environment for the development and execution of code. It consists of the common language runtime and .NET Framework base class library.

.NET Framework base class library

A collection of object-oriented types and interfaces that provides object models to assist in the development of your applications.

O

object

The base class for all types in the .NET Framework. More generically, any instance of a class or structure.

object-oriented programming

The process of creating applications by using programmatic constructs to represent real-world objects.

operator overloading

In Visual C#, defining custom behavior for an operator when interacting with custom types.

optimization

The process of examining your application and modifying it to be more efficient in terms of either speed or size of the deployed application.

overloading

Creating multiple methods with the same name but different signatures.

overriding

When inheriting from a base class, providing a new method for one of the base class methods.

P

This document is created with the unregistered version of CHM2PDF Pilot

parent-child relationship

In data access, a relationship between two tables wherein a record from the parent table can be used to retrieve one or more related records from a child table. A parent-child relationship can be defined by using a *DataRelation* object.

polymorphism

The ability of classes to provide different implementations of the same public interface. Thus, two classes that exhibit polymorphism might contain different implementations, but because the interfaces are identical, they can be treated the same in code.

primary key

In a data table, the column that uniquely identifies a record in the table.

private assembly

An assembly that is used by only one application. In .NET Framework programming, most assemblies are private assemblies.

project

The collection of Visual Studio .NET source files that make up an application.

R

Reference type

A type, such as a class, whose instance data is allocated on the heap. Variables that represent Reference types contain a reference to the object on the heap as opposed to the object's data itself. *See also* Value type.

registry

The system repository for information about a computer's settings. The registry can contain configuration information

This document is created with the unregistered version of CHM2PDF Pilot

about programs and system components.

resource

Nonexecutable data, such as a string or image, that is deployed in an application.

role-based security

Code that ensures that users of your application have appropriate permission to access your application based on the identity of the user.

run time

Represents the state of the application while executing. At run time, the application consists of executing code.

run-time error

An error that causes an invalid result or attempts an invalid operation at run time.

S

satellite assembly

When localizing an application, assemblies that contain alternate sets of resources to be used in the application for different cultures.

setter

The code segment of a property that sets the property's value.

shadowing

Hiding a member in a base class with a member in an inherited class that has a different signature.

shared assembly

This document is created with the unregistered version of CHM2PDF Pilot

An assembly that is referenced by several applications on a single machine. A shared assembly must be installed to the Global Assembly Cache (GAC).

shared members

In Visual Basic .NET, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Shared members are called static members in Visual C#.

simple binding

In data binding, binding a control to a single data column. A *TextBox* is an example of a control that can be simply bound.

solution

A group of Visual Studio .NET projects that make up a single functional unit.

SQL

See Structured Query Language (SQL).

static members

In Visual C#, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Static members are called shared members in Visual Basic .NET.

Step Into

Moving from a calling procedure to a called procedure during debugging. Also used in reference to setting breakpoints to stop execution in a specific procedure during debugging.

Step Out

In Break mode, resuming execution of a function call until the call returns.

Step Over

In Break mode, executing a procedure call as a single statement during debugging.

This document is created with the unregistered version of CHM2PDF Pilot

strong name

A name that uniquely identifies an assembly. It consists of the assembly's name, version, and culture information as well as a public key encrypted by a private key. An assembly must have a strong name to be installed to the Global Assembly Cache (GAC).

strongly typed

Refers to types for all variables being explicitly declared. Strong typing eliminates the possibility of type mismatch errors, which are discovered at compile time. Visual C# is inherently strongly typed, and Visual Basic .NET is strongly typed when Option Strict is set to On.

Struct

In Visual C#, a user-defined Value type that serves as a template for an instance of that type. *See also* Structure.

Structure

In Visual Basic .NET, a user-defined Value type that serves as a template for an instance of that type. *See also* Struct .

structured exception handling

Using *Try...Catch...Finally* blocks to handle errors that might occur during code execution. When exceptions are thrown, they are handled in the *Catch* block before code in the *Finally* block is executed.

Structured Query Language (SQL)

The standard language used for issuing commands to databases.

syntax error

An error that occurs because of typographical errors or nonsensical code. In the .NET Framework, syntax errors prevent compilation and are thus discovered at compile time.

T

target audience

This document is created with the unregistered version of CHM2PDF Pilot

The group of end users for whom an application is designed.

threads

The basic unit to which the server allocates processor time. A single process can have multiple threads.

throwing

The act of passing an exception to the next caller in the call stack. *See also* exception.

tracing

A programming technique for recording events, such as exceptions, in an application. Tracing is used during debugging and in the testing phase of application deployment.

transaction

A group of commands (treated as a single unit) that change the data stored in a database. The transaction ensures that the commands are handled in an all-or-nothing fashion if one of the commands fails, all of the commands fail, and any data that was written to the database by the commands is backed out. In this way, transactions maintain the integrity of data in a database.

tuning

The process of making adjustments to a deployed application that do not affect code.

U

Unicode

The universal character encoding scheme for characters and text. Each character is represented by a unique numeric value.

unmanaged code

Code that is not managed by the common language runtime. Unmanaged code is not checked for type-safety and

This document is created with the unregistered version of CHM2PDF Pilot

must be used with extreme care.

user control

A user-defined control that encapsulates other Windows Forms controls with custom functionality in a common container.

V

Value type

A type, such as an Integer, Decimal, or Structure, whose instance data is allocated on the stack. Variables that represent Value types contain the object's instance data itself as opposed to a reference to that data. *See also* Reference type.

W

Watch window

When debugging, the window that allows you to examine the values of predetermined watch variables.

Web Service

A service or component that is implemented by a Web server and can be instantiated and accessed programmatically as a component over the World Wide Web.

X

XML Schema

A description of the data elements contained in an XML file. The XML Schema provides the names of the elements, their types, whether or not they are key fields, and other information.

This document is created with the unregistered version of CHM2PDF Pilot

Glossary

A

abstract class

A class that cannot be instantiated but is used as a base from which other classes can be derived. In Microsoft Visual Basic .NET, abstract classes are declared using the *MustInherit* keyword. In Microsoft Visual C#, abstract classes are declared using the *abstract* keyword.

abstraction

Representing real-world objects as programmatic constructs.

abstract members

A member of a base class that cannot be invoked, but instead provides a template for members of a derived class. In Visual Basic .NET, abstract members are declared using the *MustOverride* keyword. In Visual C#, abstract members are declared using the *abstract* keyword.

ADO.NET

The data access architecture for the Microsoft .NET Framework. ADO.NET is built around a disconnected data access model that uses a set of classes called a Data Provider to retrieve data from a data source.

alias

The name you use to identify resources on the Web. Aliases represent physical resources on the Web server, such as a Web form, an HTML page, or a graphic.

anchoring

In Windows Forms controls, the resizing and repositioning of a control when its parent form is resized. You can anchor a control by setting the *Anchor* property.

This document is created with the unregistered version of CHM2PDF Pilot

assembly

The primary unit of deployment for a .NET application. An assembly is either an executable application (EXE) or a class library (DLL).

assembly manifest

The metadata for the assembly. It contains all of the information needed to describe the assembly to the common language runtime.

authentication

The process of determining the identity of a user. In effect, authentication validates that the users are who they say they are. *See also* authorization.

authorization

The process of allowing access to an application or resource based on credentials supplied by the user. *See also* authentication.

B

base class

A class that provides properties and methods as a foundation for a derived class. In objected-oriented programming, one class can be based on another through inheritance. Using this technique, the base class provides characteristics (such as properties and methods) to a derived class. The derived class can reuse, modify, or add to the members of the base class.

Bezier

A curve specified by four points. Two points specify the ends of the curve, and the other two points influence the shape of the curve between the end points.

Boolean

A data type that can represent a value of either true or false.

This document is created with the unregistered version of CHM2PDF Pilot

bootstrapper

In deployment, an application that installs Microsoft Windows Installer on machines that do not currently have it installed.

boundary condition

In testing, test conditions at the limit of normal operation and just over or under those limits.

boxing

The act of wrapping a Value type, such as an Integer, inside an object so that it can be treated like a Reference type.

Break mode

In debugging, a state of suspended execution wherein the application can be executed in a line-by-line manner, and individual program variables can be examined.

breakpoint

In debugging, a predetermined point where execution of an application will halt and enter Break mode.

C

cast

An explicit conversion from one type to another.

circular reference

A condition that occurs when two objects refer to each other but do not have an external reference. Circular references are automatically detected and disposed of by the garbage collector.

class

A user-defined Reference type that serves as a template for an object of that type.

This document is created with the unregistered version of CHM2PDF Pilot

CLR

See common language runtime (CLR).

code access security

Verification that all callers to code have appropriate permission to access it.

code tuning

The act of examining application code and optimizing for faster execution. Code tuning is usually an iterative process that involves several successive rounds of planning, improvement, and assessment of the results.

common language runtime (CLR)

The environment in which managed code executes. The common language runtime provides just-in-time compilation, enforces type safety, and manages memory through garbage collection.

Common Type System

A set of types that are used by all .NET languages, thus ensuring .NET language type compatibility.

complex binding

In data binding, binding more than one column to a single control. The *DataGrid* is an example of a control that can be complexly bound.

component

Strictly speaking, a class that implements *IComponent* or inherits *System*.*Component*. More generally, any class that provides a reusable set of functionality.

connected data access

Data access that requires a data connection that is always open.

Constant

This document is created with the unregistered version of CHM2PDF Pilot

A type that represents a value that does not change in the course of an application's execution and cannot be changed.

constraint

In ADO.NET, an object that represents a rule that enforces referential integrity, such as a constraint that specifies a primary key.

constructor

The method that executes when a class is instantiated. The constructor usually contains initialization code to set member variables to initial values.

container controls

Controls that can host other controls. Examples include *Forms*, *Panels*, and *TabPages*.

control

A class that inherits *System*.*Control*. Controls usually have a visual representation, interact with the user as a part of the user interface, and are hosted in a container control.

controls collection

The collection of a container control that manages the controls it hosts.

custom control

A user-designed control that directly inherits *System*.*Control*.

D

data binding

A way to link data (such as a data set) in your application to the properties of a control.

data consumer

This document is created with the unregistered version of CHM2PDF Pilot

A control that is bound to a data source.

Data Provider

A set of classes that work together to provide managed data access.

DataSet

A class that contains an application copy of data that has been retrieved from a data source.

delegate

An object that acts as a type-safe function pointer. Delegates can be used to invoke the method they point to.

deploying

Installing an application on the computer where it will run.

derived class

A class that is based on another class (called a base class) through inheritance. A derived class inherits the members of its base class and can override or shadow those members.

design time

Represents the state of the application while being designed. Applications at design time consist of noncompiled code that can be manipulated in the code editor or designer window.

destructor

The method that is run just before a class is removed from memory. Generally, a destructor contains any code required to "clean up" after the class. *See also* finalizer.

disconnected data access

Data access that uses a data connection only when executing database operations. This is the default kind of data access implemented by ADO.NET.

This document is created with the unregistered version of CHM2PDF Pilot

docking

In Windows Forms, attaching a control to the edge of the form. You can dock a control by setting the *Dock* property.

E

encapsulation

In component programming, separating the implementation of a component from the interface. Only the public interface of a component is made accessible to the rest of the application. Component data should never be accessible to outside callers.

event

A notification raised by a control or component that a specified condition has occurred.

event handler

A method that is executed when a particular event is raised.

exception

Any class that derives from *System.Exception*. Exceptions are thrown in response to run-time errors. *See also* throwing.

exception handling

The process of dealing with unusual occurrences within code so that it does not cause the program to crash or lose data.

explicit conversion

A conversion of one type to another that cannot be performed automatically. An explicit conversion usually presents some danger of a failed conversion or a loss of data.

extender provider

This document is created with the unregistered version of CHM2PDF Pilot

A specialized type of control that provides design time properties to other controls. *ErrorProvider* and *HelpProvider* are examples of extender providers.

F

field

A member variable of a class or structure.

finalizer

In Visual Basic .NET, the method that is called just before an instance of a class is destroyed. *See also* destructor.

floating-point numbers

A number that has a decimal component. *See also* floating-point types.

floating-point types

Types capable of representing floating-point numbers. Floating-point types include Single (float) and Double (double). *See also* floating-point numbers.

focus

When referring to a control, the condition of being able to receive user input through the keyboard or other input devices. A control that is capable of receiving input is said to have the focus.

foreign key

In a data table, a column that contains a key value that uniquely identifies a record in a related table.

form

The primary unit of the user interface in a Windows Forms application. A form serves as a nexus for data display and user input, and can host controls.

This document is created with the unregistered version of CHM2PDF Pilot

G

GAC

See Global Assembly Cache (GAC).

garbage collection

Automatic memory management provided by the common language runtime. Unused memory is automatically reclaimed by garbage collection without interaction with the application.

GDI+

See Graphic Device Interface (GDI).

getter

The code segment of a property that returns the value the property contains.

Global Assembly Cache (GAC)

A machine-wide cache that stores assemblies that have been specifically designated to be shared between several applications.

globalization

Applying culture-based formatting to existing data, for example, using culture-specific currency symbols and formatting.

Graphic Device Interface (GDI)

In Windows, a graphics display system used by applications to display or print bitmapped text (TrueType fonts), images, and other graphical elements. GDI+ is the name given to the .NET Framework's managed implementation of the GDI.

I

This document is created with the unregistered version of CHM2PDF Pilot

implicit conversion

A conversion between types that can be performed automatically by the common language runtime. An implicit conversion will always succeed and never pose a danger of data loss.

inheritance

In object-oriented programming, creating a new class that incorporates and extends all of the characteristics and functionality of a base class.

inherited control

A user-created control that inherits an existing control. An inherited control inherits all of the functionality of the base control and can incorporate custom functionality as well.

interface

A contract for behavior that can be implemented by a class. Classes that implement the same interface can be expected to provide the members defined by the interface.

L

localization

Providing multiple sets of data specific to different cultures and retrieving the data appropriate to the current culture setting.

logical error

An error that compiles and executes correctly but produces unexpected results. These can be the most difficult errors to detect in application testing.

M

This document is created with the unregistered version of CHM2PDF Pilot

managed code

Code that runs under the common language runtime. The common language runtime handles many tasks that would formerly have been handled in the application's executable. Managed code solves the Windows programming problems of component registration and versioning (sometimes called DLL Hell) because managed code contains all the versioning and type information that the common language runtime needs to run the application. The common language runtime handles registration dynamically at run time rather than statically through the system registry, as is done with applications based on the Component Object Model (COM).

metadata

Data about data. For example, the name, version, security information, and size of a file constitute metadata about the file.

Microsoft Intermediate Language (MSIL)

A low-level language that is just-in-time compiled to native code at run time. All .NET assemblies are represented in the MSIL.

MSIL

See Microsoft Intermediate Language (MSIL).

multidimensional array

An array that contains more than one dimension.

N

namespace

A logical organization of types that perform related functions.

.NET Framework

This document is created with the unregistered version of CHM2PDF Pilot

A managed environment for the development and execution of code. It consists of the common language runtime and .NET Framework base class library.

.NET Framework base class library

A collection of object-oriented types and interfaces that provides object models to assist in the development of your applications.

O

object

The base class for all types in the .NET Framework. More generically, any instance of a class or structure.

object-oriented programming

The process of creating applications by using programmatic constructs to represent real-world objects.

operator overloading

In Visual C#, defining custom behavior for an operator when interacting with custom types.

optimization

The process of examining your application and modifying it to be more efficient in terms of either speed or size of the deployed application.

overloading

Creating multiple methods with the same name but different signatures.

overriding

When inheriting from a base class, providing a new method for one of the base class methods.

P

This document is created with the unregistered version of CHM2PDF Pilot

parent-child relationship

In data access, a relationship between two tables wherein a record from the parent table can be used to retrieve one or more related records from a child table. A parent-child relationship can be defined by using a *DataRelation* object.

polymorphism

The ability of classes to provide different implementations of the same public interface. Thus, two classes that exhibit polymorphism might contain different implementations, but because the interfaces are identical, they can be treated the same in code.

primary key

In a data table, the column that uniquely identifies a record in the table.

private assembly

An assembly that is used by only one application. In .NET Framework programming, most assemblies are private assemblies.

project

The collection of Visual Studio .NET source files that make up an application.

R

Reference type

A type, such as a class, whose instance data is allocated on the heap. Variables that represent Reference types contain a reference to the object on the heap as opposed to the object's data itself. *See also* Value type.

registry

The system repository for information about a computer's settings. The registry can contain configuration information

This document is created with the unregistered version of CHM2PDF Pilot

about programs and system components.

resource

Nonexecutable data, such as a string or image, that is deployed in an application.

role-based security

Code that ensures that users of your application have appropriate permission to access your application based on the identity of the user.

run time

Represents the state of the application while executing. At run time, the application consists of executing code.

run-time error

An error that causes an invalid result or attempts an invalid operation at run time.

S

satellite assembly

When localizing an application, assemblies that contain alternate sets of resources to be used in the application for different cultures.

setter

The code segment of a property that sets the property's value.

shadowing

Hiding a member in a base class with a member in an inherited class that has a different signature.

shared assembly

This document is created with the unregistered version of CHM2PDF Pilot

An assembly that is referenced by several applications on a single machine. A shared assembly must be installed to the Global Assembly Cache (GAC).

shared members

In Visual Basic .NET, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Shared members are called static members in Visual C#.

simple binding

In data binding, binding a control to a single data column. A *TextBox* is an example of a control that can be simply bound.

solution

A group of Visual Studio .NET projects that make up a single functional unit.

SQL

See Structured Query Language (SQL).

static members

In Visual C#, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Static members are called shared members in Visual Basic .NET.

Step Into

Moving from a calling procedure to a called procedure during debugging. Also used in reference to setting breakpoints to stop execution in a specific procedure during debugging.

Step Out

In Break mode, resuming execution of a function call until the call returns.

Step Over

In Break mode, executing a procedure call as a single statement during debugging.

This document is created with the unregistered version of CHM2PDF Pilot

strong name

A name that uniquely identifies an assembly. It consists of the assembly's name, version, and culture information as well as a public key encrypted by a private key. An assembly must have a strong name to be installed to the Global Assembly Cache (GAC).

strongly typed

Refers to types for all variables being explicitly declared. Strong typing eliminates the possibility of type mismatch errors, which are discovered at compile time. Visual C# is inherently strongly typed, and Visual Basic .NET is strongly typed when Option Strict is set to On.

Struct

In Visual C#, a user-defined Value type that serves as a template for an instance of that type. *See also* Structure.

Structure

In Visual Basic .NET, a user-defined Value type that serves as a template for an instance of that type. *See also* Struct .

structured exception handling

Using *Try...Catch...Finally* blocks to handle errors that might occur during code execution. When exceptions are thrown, they are handled in the *Catch* block before code in the *Finally* block is executed.

Structured Query Language (SQL)

The standard language used for issuing commands to databases.

syntax error

An error that occurs because of typographical errors or nonsensical code. In the .NET Framework, syntax errors prevent compilation and are thus discovered at compile time.

T

target audience

This document is created with the unregistered version of CHM2PDF Pilot

The group of end users for whom an application is designed.

threads

The basic unit to which the server allocates processor time. A single process can have multiple threads.

throwing

The act of passing an exception to the next caller in the call stack. *See also* exception.

tracing

A programming technique for recording events, such as exceptions, in an application. Tracing is used during debugging and in the testing phase of application deployment.

transaction

A group of commands (treated as a single unit) that change the data stored in a database. The transaction ensures that the commands are handled in an all-or-nothing fashion if one of the commands fails, all of the commands fail, and any data that was written to the database by the commands is backed out. In this way, transactions maintain the integrity of data in a database.

tuning

The process of making adjustments to a deployed application that do not affect code.

U

Unicode

The universal character encoding scheme for characters and text. Each character is represented by a unique numeric value.

unmanaged code

Code that is not managed by the common language runtime. Unmanaged code is not checked for type-safety and

This document is created with the unregistered version of CHM2PDF Pilot

must be used with extreme care.

user control

A user-defined control that encapsulates other Windows Forms controls with custom functionality in a common container.

V

Value type

A type, such as an Integer, Decimal, or Structure, whose instance data is allocated on the stack. Variables that represent Value types contain the object's instance data itself as opposed to a reference to that data. *See also* Reference type.

W

Watch window

When debugging, the window that allows you to examine the values of predetermined watch variables.

Web Service

A service or component that is implemented by a Web server and can be instantiated and accessed programmatically as a component over the World Wide Web.

X

XML Schema

A description of the data elements contained in an XML file. The XML Schema provides the names of the elements, their types, whether or not they are key fields, and other information.

This document is created with the unregistered version of CHM2PDF Pilot

Glossary

A

abstract class

A class that cannot be instantiated but is used as a base from which other classes can be derived. In Microsoft Visual Basic .NET, abstract classes are declared using the *MustInherit* keyword. In Microsoft Visual C#, abstract classes are declared using the *abstract* keyword.

abstraction

Representing real-world objects as programmatic constructs.

abstract members

A member of a base class that cannot be invoked, but instead provides a template for members of a derived class. In Visual Basic .NET, abstract members are declared using the *MustOverride* keyword. In Visual C#, abstract members are declared using the *abstract* keyword.

ADO.NET

The data access architecture for the Microsoft .NET Framework. ADO.NET is built around a disconnected data access model that uses a set of classes called a Data Provider to retrieve data from a data source.

alias

The name you use to identify resources on the Web. Aliases represent physical resources on the Web server, such as a Web form, an HTML page, or a graphic.

anchoring

In Windows Forms controls, the resizing and repositioning of a control when its parent form is resized. You can anchor a control by setting the *Anchor* property.

This document is created with the unregistered version of CHM2PDF Pilot

assembly

The primary unit of deployment for a .NET application. An assembly is either an executable application (EXE) or a class library (DLL).

assembly manifest

The metadata for the assembly. It contains all of the information needed to describe the assembly to the common language runtime.

authentication

The process of determining the identity of a user. In effect, authentication validates that the users are who they say they are. *See also* authorization.

authorization

The process of allowing access to an application or resource based on credentials supplied by the user. *See also* authentication.

B

base class

A class that provides properties and methods as a foundation for a derived class. In objected-oriented programming, one class can be based on another through inheritance. Using this technique, the base class provides characteristics (such as properties and methods) to a derived class. The derived class can reuse, modify, or add to the members of the base class.

Bezier

A curve specified by four points. Two points specify the ends of the curve, and the other two points influence the shape of the curve between the end points.

Boolean

A data type that can represent a value of either true or false.

This document is created with the unregistered version of CHM2PDF Pilot

bootstrapper

In deployment, an application that installs Microsoft Windows Installer on machines that do not currently have it installed.

boundary condition

In testing, test conditions at the limit of normal operation and just over or under those limits.

boxing

The act of wrapping a Value type, such as an Integer, inside an object so that it can be treated like a Reference type.

Break mode

In debugging, a state of suspended execution wherein the application can be executed in a line-by-line manner, and individual program variables can be examined.

breakpoint

In debugging, a predetermined point where execution of an application will halt and enter Break mode.

C

cast

An explicit conversion from one type to another.

circular reference

A condition that occurs when two objects refer to each other but do not have an external reference. Circular references are automatically detected and disposed of by the garbage collector.

class

A user-defined Reference type that serves as a template for an object of that type.

This document is created with the unregistered version of CHM2PDF Pilot

CLR

See common language runtime (CLR).

code access security

Verification that all callers to code have appropriate permission to access it.

code tuning

The act of examining application code and optimizing for faster execution. Code tuning is usually an iterative process that involves several successive rounds of planning, improvement, and assessment of the results.

common language runtime (CLR)

The environment in which managed code executes. The common language runtime provides just-in-time compilation, enforces type safety, and manages memory through garbage collection.

Common Type System

A set of types that are used by all .NET languages, thus ensuring .NET language type compatibility.

complex binding

In data binding, binding more than one column to a single control. The *DataGrid* is an example of a control that can be complexly bound.

component

Strictly speaking, a class that implements *IComponent* or inherits *System*.*Component*. More generally, any class that provides a reusable set of functionality.

connected data access

Data access that requires a data connection that is always open.

Constant

This document is created with the unregistered version of CHM2PDF Pilot

A type that represents a value that does not change in the course of an application's execution and cannot be changed.

constraint

In ADO.NET, an object that represents a rule that enforces referential integrity, such as a constraint that specifies a primary key.

constructor

The method that executes when a class is instantiated. The constructor usually contains initialization code to set member variables to initial values.

container controls

Controls that can host other controls. Examples include *Forms*, *Panels*, and *TabPages*.

control

A class that inherits *System*.*Control*. Controls usually have a visual representation, interact with the user as a part of the user interface, and are hosted in a container control.

controls collection

The collection of a container control that manages the controls it hosts.

custom control

A user-designed control that directly inherits *System*.*Control*.

D

data binding

A way to link data (such as a data set) in your application to the properties of a control.

data consumer

This document is created with the unregistered version of CHM2PDF Pilot

A control that is bound to a data source.

Data Provider

A set of classes that work together to provide managed data access.

DataSet

A class that contains an application copy of data that has been retrieved from a data source.

delegate

An object that acts as a type-safe function pointer. Delegates can be used to invoke the method they point to.

deploying

Installing an application on the computer where it will run.

derived class

A class that is based on another class (called a base class) through inheritance. A derived class inherits the members of its base class and can override or shadow those members.

design time

Represents the state of the application while being designed. Applications at design time consist of noncompiled code that can be manipulated in the code editor or designer window.

destructor

The method that is run just before a class is removed from memory. Generally, a destructor contains any code required to "clean up" after the class. *See also* finalizer.

disconnected data access

Data access that uses a data connection only when executing database operations. This is the default kind of data access implemented by ADO.NET.

This document is created with the unregistered version of CHM2PDF Pilot

docking

In Windows Forms, attaching a control to the edge of the form. You can dock a control by setting the *Dock* property.

E

encapsulation

In component programming, separating the implementation of a component from the interface. Only the public interface of a component is made accessible to the rest of the application. Component data should never be accessible to outside callers.

event

A notification raised by a control or component that a specified condition has occurred.

event handler

A method that is executed when a particular event is raised.

exception

Any class that derives from *System.Exception*. Exceptions are thrown in response to run-time errors. *See also* throwing.

exception handling

The process of dealing with unusual occurrences within code so that it does not cause the program to crash or lose data.

explicit conversion

A conversion of one type to another that cannot be performed automatically. An explicit conversion usually presents some danger of a failed conversion or a loss of data.

extender provider

This document is created with the unregistered version of CHM2PDF Pilot

A specialized type of control that provides design time properties to other controls. *ErrorProvider* and *HelpProvider* are examples of extender providers.

F

field

A member variable of a class or structure.

finalizer

In Visual Basic .NET, the method that is called just before an instance of a class is destroyed. *See also* destructor.

floating-point numbers

A number that has a decimal component. *See also* floating-point types.

floating-point types

Types capable of representing floating-point numbers. Floating-point types include Single (float) and Double (double). *See also* floating-point numbers.

focus

When referring to a control, the condition of being able to receive user input through the keyboard or other input devices. A control that is capable of receiving input is said to have the focus.

foreign key

In a data table, a column that contains a key value that uniquely identifies a record in a related table.

form

The primary unit of the user interface in a Windows Forms application. A form serves as a nexus for data display and user input, and can host controls.

This document is created with the unregistered version of CHM2PDF Pilot

G

GAC

See Global Assembly Cache (GAC).

garbage collection

Automatic memory management provided by the common language runtime. Unused memory is automatically reclaimed by garbage collection without interaction with the application.

GDI+

See Graphic Device Interface (GDI).

getter

The code segment of a property that returns the value the property contains.

Global Assembly Cache (GAC)

A machine-wide cache that stores assemblies that have been specifically designated to be shared between several applications.

globalization

Applying culture-based formatting to existing data, for example, using culture-specific currency symbols and formatting.

Graphic Device Interface (GDI)

In Windows, a graphics display system used by applications to display or print bitmapped text (TrueType fonts), images, and other graphical elements. GDI+ is the name given to the .NET Framework's managed implementation of the GDI.

I

This document is created with the unregistered version of CHM2PDF Pilot

implicit conversion

A conversion between types that can be performed automatically by the common language runtime. An implicit conversion will always succeed and never pose a danger of data loss.

inheritance

In object-oriented programming, creating a new class that incorporates and extends all of the characteristics and functionality of a base class.

inherited control

A user-created control that inherits an existing control. An inherited control inherits all of the functionality of the base control and can incorporate custom functionality as well.

interface

A contract for behavior that can be implemented by a class. Classes that implement the same interface can be expected to provide the members defined by the interface.

L

localization

Providing multiple sets of data specific to different cultures and retrieving the data appropriate to the current culture setting.

logical error

An error that compiles and executes correctly but produces unexpected results. These can be the most difficult errors to detect in application testing.

M

This document is created with the unregistered version of CHM2PDF Pilot

managed code

Code that runs under the common language runtime. The common language runtime handles many tasks that would formerly have been handled in the application's executable. Managed code solves the Windows programming problems of component registration and versioning (sometimes called DLL Hell) because managed code contains all the versioning and type information that the common language runtime needs to run the application. The common language runtime handles registration dynamically at run time rather than statically through the system registry, as is done with applications based on the Component Object Model (COM).

metadata

Data about data. For example, the name, version, security information, and size of a file constitute metadata about the file.

Microsoft Intermediate Language (MSIL)

A low-level language that is just-in-time compiled to native code at run time. All .NET assemblies are represented in the MSIL.

MSIL

See Microsoft Intermediate Language (MSIL).

multidimensional array

An array that contains more than one dimension.

N

namespace

A logical organization of types that perform related functions.

.NET Framework

This document is created with the unregistered version of CHM2PDF Pilot

A managed environment for the development and execution of code. It consists of the common language runtime and .NET Framework base class library.

.NET Framework base class library

A collection of object-oriented types and interfaces that provides object models to assist in the development of your applications.

O

object

The base class for all types in the .NET Framework. More generically, any instance of a class or structure.

object-oriented programming

The process of creating applications by using programmatic constructs to represent real-world objects.

operator overloading

In Visual C#, defining custom behavior for an operator when interacting with custom types.

optimization

The process of examining your application and modifying it to be more efficient in terms of either speed or size of the deployed application.

overloading

Creating multiple methods with the same name but different signatures.

overriding

When inheriting from a base class, providing a new method for one of the base class methods.

P

This document is created with the unregistered version of CHM2PDF Pilot

parent-child relationship

In data access, a relationship between two tables wherein a record from the parent table can be used to retrieve one or more related records from a child table. A parent-child relationship can be defined by using a *DataRelation* object.

polymorphism

The ability of classes to provide different implementations of the same public interface. Thus, two classes that exhibit polymorphism might contain different implementations, but because the interfaces are identical, they can be treated the same in code.

primary key

In a data table, the column that uniquely identifies a record in the table.

private assembly

An assembly that is used by only one application. In .NET Framework programming, most assemblies are private assemblies.

project

The collection of Visual Studio .NET source files that make up an application.

R

Reference type

A type, such as a class, whose instance data is allocated on the heap. Variables that represent Reference types contain a reference to the object on the heap as opposed to the object's data itself. *See also* Value type.

registry

The system repository for information about a computer's settings. The registry can contain configuration information

This document is created with the unregistered version of CHM2PDF Pilot

about programs and system components.

resource

Nonexecutable data, such as a string or image, that is deployed in an application.

role-based security

Code that ensures that users of your application have appropriate permission to access your application based on the identity of the user.

run time

Represents the state of the application while executing. At run time, the application consists of executing code.

run-time error

An error that causes an invalid result or attempts an invalid operation at run time.

S

satellite assembly

When localizing an application, assemblies that contain alternate sets of resources to be used in the application for different cultures.

setter

The code segment of a property that sets the property's value.

shadowing

Hiding a member in a base class with a member in an inherited class that has a different signature.

shared assembly

This document is created with the unregistered version of CHM2PDF Pilot

An assembly that is referenced by several applications on a single machine. A shared assembly must be installed to the Global Assembly Cache (GAC).

shared members

In Visual Basic .NET, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Shared members are called static members in Visual C#.

simple binding

In data binding, binding a control to a single data column. A *TextBox* is an example of a control that can be simply bound.

solution

A group of Visual Studio .NET projects that make up a single functional unit.

SQL

See Structured Query Language (SQL).

static members

In Visual C#, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Static members are called shared members in Visual Basic .NET.

Step Into

Moving from a calling procedure to a called procedure during debugging. Also used in reference to setting breakpoints to stop execution in a specific procedure during debugging.

Step Out

In Break mode, resuming execution of a function call until the call returns.

Step Over

In Break mode, executing a procedure call as a single statement during debugging.

This document is created with the unregistered version of CHM2PDF Pilot

strong name

A name that uniquely identifies an assembly. It consists of the assembly's name, version, and culture information as well as a public key encrypted by a private key. An assembly must have a strong name to be installed to the Global Assembly Cache (GAC).

strongly typed

Refers to types for all variables being explicitly declared. Strong typing eliminates the possibility of type mismatch errors, which are discovered at compile time. Visual C# is inherently strongly typed, and Visual Basic .NET is strongly typed when Option Strict is set to On.

Struct

In Visual C#, a user-defined Value type that serves as a template for an instance of that type. *See also* Structure.

Structure

In Visual Basic .NET, a user-defined Value type that serves as a template for an instance of that type. *See also* Struct .

structured exception handling

Using *Try...Catch...Finally* blocks to handle errors that might occur during code execution. When exceptions are thrown, they are handled in the *Catch* block before code in the *Finally* block is executed.

Structured Query Language (SQL)

The standard language used for issuing commands to databases.

syntax error

An error that occurs because of typographical errors or nonsensical code. In the .NET Framework, syntax errors prevent compilation and are thus discovered at compile time.

T

target audience

This document is created with the unregistered version of CHM2PDF Pilot

The group of end users for whom an application is designed.

threads

The basic unit to which the server allocates processor time. A single process can have multiple threads.

throwing

The act of passing an exception to the next caller in the call stack. *See also* exception.

tracing

A programming technique for recording events, such as exceptions, in an application. Tracing is used during debugging and in the testing phase of application deployment.

transaction

A group of commands (treated as a single unit) that change the data stored in a database. The transaction ensures that the commands are handled in an all-or-nothing fashion if one of the commands fails, all of the commands fail, and any data that was written to the database by the commands is backed out. In this way, transactions maintain the integrity of data in a database.

tuning

The process of making adjustments to a deployed application that do not affect code.

U

Unicode

The universal character encoding scheme for characters and text. Each character is represented by a unique numeric value.

unmanaged code

Code that is not managed by the common language runtime. Unmanaged code is not checked for type-safety and

This document is created with the unregistered version of CHM2PDF Pilot

must be used with extreme care.

user control

A user-defined control that encapsulates other Windows Forms controls with custom functionality in a common container.

V

Value type

A type, such as an Integer, Decimal, or Structure, whose instance data is allocated on the stack. Variables that represent Value types contain the object's instance data itself as opposed to a reference to that data. *See also* Reference type.

W

Watch window

When debugging, the window that allows you to examine the values of predetermined watch variables.

Web Service

A service or component that is implemented by a Web server and can be instantiated and accessed programmatically as a component over the World Wide Web.

X

XML Schema

A description of the data elements contained in an XML file. The XML Schema provides the names of the elements, their types, whether or not they are key fields, and other information.

This document is created with the unregistered version of CHM2PDF Pilot

Glossary

A

abstract class

A class that cannot be instantiated but is used as a base from which other classes can be derived. In Microsoft Visual Basic .NET, abstract classes are declared using the *MustInherit* keyword. In Microsoft Visual C#, abstract classes are declared using the *abstract* keyword.

abstraction

Representing real-world objects as programmatic constructs.

abstract members

A member of a base class that cannot be invoked, but instead provides a template for members of a derived class. In Visual Basic .NET, abstract members are declared using the *MustOverride* keyword. In Visual C#, abstract members are declared using the *abstract* keyword.

ADO.NET

The data access architecture for the Microsoft .NET Framework. ADO.NET is built around a disconnected data access model that uses a set of classes called a Data Provider to retrieve data from a data source.

alias

The name you use to identify resources on the Web. Aliases represent physical resources on the Web server, such as a Web form, an HTML page, or a graphic.

anchoring

In Windows Forms controls, the resizing and repositioning of a control when its parent form is resized. You can anchor a control by setting the *Anchor* property.

This document is created with the unregistered version of CHM2PDF Pilot

assembly

The primary unit of deployment for a .NET application. An assembly is either an executable application (EXE) or a class library (DLL).

assembly manifest

The metadata for the assembly. It contains all of the information needed to describe the assembly to the common language runtime.

authentication

The process of determining the identity of a user. In effect, authentication validates that the users are who they say they are. *See also* authorization.

authorization

The process of allowing access to an application or resource based on credentials supplied by the user. *See also* authentication.

B

base class

A class that provides properties and methods as a foundation for a derived class. In objected-oriented programming, one class can be based on another through inheritance. Using this technique, the base class provides characteristics (such as properties and methods) to a derived class. The derived class can reuse, modify, or add to the members of the base class.

Bezier

A curve specified by four points. Two points specify the ends of the curve, and the other two points influence the shape of the curve between the end points.

Boolean

A data type that can represent a value of either true or false.

This document is created with the unregistered version of CHM2PDF Pilot

bootstrapper

In deployment, an application that installs Microsoft Windows Installer on machines that do not currently have it installed.

boundary condition

In testing, test conditions at the limit of normal operation and just over or under those limits.

boxing

The act of wrapping a Value type, such as an Integer, inside an object so that it can be treated like a Reference type.

Break mode

In debugging, a state of suspended execution wherein the application can be executed in a line-by-line manner, and individual program variables can be examined.

breakpoint

In debugging, a predetermined point where execution of an application will halt and enter Break mode.

C

cast

An explicit conversion from one type to another.

circular reference

A condition that occurs when two objects refer to each other but do not have an external reference. Circular references are automatically detected and disposed of by the garbage collector.

class

A user-defined Reference type that serves as a template for an object of that type.

This document is created with the unregistered version of CHM2PDF Pilot

CLR

See common language runtime (CLR).

code access security

Verification that all callers to code have appropriate permission to access it.

code tuning

The act of examining application code and optimizing for faster execution. Code tuning is usually an iterative process that involves several successive rounds of planning, improvement, and assessment of the results.

common language runtime (CLR)

The environment in which managed code executes. The common language runtime provides just-in-time compilation, enforces type safety, and manages memory through garbage collection.

Common Type System

A set of types that are used by all .NET languages, thus ensuring .NET language type compatibility.

complex binding

In data binding, binding more than one column to a single control. The *DataGrid* is an example of a control that can be complexly bound.

component

Strictly speaking, a class that implements *IComponent* or inherits *System*.*Component*. More generally, any class that provides a reusable set of functionality.

connected data access

Data access that requires a data connection that is always open.

Constant

This document is created with the unregistered version of CHM2PDF Pilot

A type that represents a value that does not change in the course of an application's execution and cannot be changed.

constraint

In ADO.NET, an object that represents a rule that enforces referential integrity, such as a constraint that specifies a primary key.

constructor

The method that executes when a class is instantiated. The constructor usually contains initialization code to set member variables to initial values.

container controls

Controls that can host other controls. Examples include *Forms*, *Panels*, and *TabPages*.

control

A class that inherits *System*.*Control*. Controls usually have a visual representation, interact with the user as a part of the user interface, and are hosted in a container control.

controls collection

The collection of a container control that manages the controls it hosts.

custom control

A user-designed control that directly inherits *System*.*Control*.

D

data binding

A way to link data (such as a data set) in your application to the properties of a control.

data consumer

This document is created with the unregistered version of CHM2PDF Pilot

A control that is bound to a data source.

Data Provider

A set of classes that work together to provide managed data access.

DataSet

A class that contains an application copy of data that has been retrieved from a data source.

delegate

An object that acts as a type-safe function pointer. Delegates can be used to invoke the method they point to.

deploying

Installing an application on the computer where it will run.

derived class

A class that is based on another class (called a base class) through inheritance. A derived class inherits the members of its base class and can override or shadow those members.

design time

Represents the state of the application while being designed. Applications at design time consist of noncompiled code that can be manipulated in the code editor or designer window.

destructor

The method that is run just before a class is removed from memory. Generally, a destructor contains any code required to "clean up" after the class. *See also* finalizer.

disconnected data access

Data access that uses a data connection only when executing database operations. This is the default kind of data access implemented by ADO.NET.

This document is created with the unregistered version of CHM2PDF Pilot

docking

In Windows Forms, attaching a control to the edge of the form. You can dock a control by setting the *Dock* property.

E

encapsulation

In component programming, separating the implementation of a component from the interface. Only the public interface of a component is made accessible to the rest of the application. Component data should never be accessible to outside callers.

event

A notification raised by a control or component that a specified condition has occurred.

event handler

A method that is executed when a particular event is raised.

exception

Any class that derives from *System.Exception*. Exceptions are thrown in response to run-time errors. *See also* throwing.

exception handling

The process of dealing with unusual occurrences within code so that it does not cause the program to crash or lose data.

explicit conversion

A conversion of one type to another that cannot be performed automatically. An explicit conversion usually presents some danger of a failed conversion or a loss of data.

extender provider

This document is created with the unregistered version of CHM2PDF Pilot

A specialized type of control that provides design time properties to other controls. *ErrorProvider* and *HelpProvider* are examples of extender providers.

F

field

A member variable of a class or structure.

finalizer

In Visual Basic .NET, the method that is called just before an instance of a class is destroyed. *See also* destructor.

floating-point numbers

A number that has a decimal component. *See also* floating-point types.

floating-point types

Types capable of representing floating-point numbers. Floating-point types include Single (float) and Double (double). *See also* floating-point numbers.

focus

When referring to a control, the condition of being able to receive user input through the keyboard or other input devices. A control that is capable of receiving input is said to have the focus.

foreign key

In a data table, a column that contains a key value that uniquely identifies a record in a related table.

form

The primary unit of the user interface in a Windows Forms application. A form serves as a nexus for data display and user input, and can host controls.

This document is created with the unregistered version of CHM2PDF Pilot

G

GAC

See Global Assembly Cache (GAC).

garbage collection

Automatic memory management provided by the common language runtime. Unused memory is automatically reclaimed by garbage collection without interaction with the application.

GDI+

See Graphic Device Interface (GDI).

getter

The code segment of a property that returns the value the property contains.

Global Assembly Cache (GAC)

A machine-wide cache that stores assemblies that have been specifically designated to be shared between several applications.

globalization

Applying culture-based formatting to existing data, for example, using culture-specific currency symbols and formatting.

Graphic Device Interface (GDI)

In Windows, a graphics display system used by applications to display or print bitmapped text (TrueType fonts), images, and other graphical elements. GDI+ is the name given to the .NET Framework's managed implementation of the GDI.

I

This document is created with the unregistered version of CHM2PDF Pilot

implicit conversion

A conversion between types that can be performed automatically by the common language runtime. An implicit conversion will always succeed and never pose a danger of data loss.

inheritance

In object-oriented programming, creating a new class that incorporates and extends all of the characteristics and functionality of a base class.

inherited control

A user-created control that inherits an existing control. An inherited control inherits all of the functionality of the base control and can incorporate custom functionality as well.

interface

A contract for behavior that can be implemented by a class. Classes that implement the same interface can be expected to provide the members defined by the interface.

L

localization

Providing multiple sets of data specific to different cultures and retrieving the data appropriate to the current culture setting.

logical error

An error that compiles and executes correctly but produces unexpected results. These can be the most difficult errors to detect in application testing.

M

This document is created with the unregistered version of CHM2PDF Pilot

managed code

Code that runs under the common language runtime. The common language runtime handles many tasks that would formerly have been handled in the application's executable. Managed code solves the Windows programming problems of component registration and versioning (sometimes called DLL Hell) because managed code contains all the versioning and type information that the common language runtime needs to run the application. The common language runtime handles registration dynamically at run time rather than statically through the system registry, as is done with applications based on the Component Object Model (COM).

metadata

Data about data. For example, the name, version, security information, and size of a file constitute metadata about the file.

Microsoft Intermediate Language (MSIL)

A low-level language that is just-in-time compiled to native code at run time. All .NET assemblies are represented in the MSIL.

MSIL

See Microsoft Intermediate Language (MSIL).

multidimensional array

An array that contains more than one dimension.

N

namespace

A logical organization of types that perform related functions.

.NET Framework

This document is created with the unregistered version of CHM2PDF Pilot

A managed environment for the development and execution of code. It consists of the common language runtime and .NET Framework base class library.

.NET Framework base class library

A collection of object-oriented types and interfaces that provides object models to assist in the development of your applications.

O

object

The base class for all types in the .NET Framework. More generically, any instance of a class or structure.

object-oriented programming

The process of creating applications by using programmatic constructs to represent real-world objects.

operator overloading

In Visual C#, defining custom behavior for an operator when interacting with custom types.

optimization

The process of examining your application and modifying it to be more efficient in terms of either speed or size of the deployed application.

overloading

Creating multiple methods with the same name but different signatures.

overriding

When inheriting from a base class, providing a new method for one of the base class methods.

P

This document is created with the unregistered version of CHM2PDF Pilot

parent-child relationship

In data access, a relationship between two tables wherein a record from the parent table can be used to retrieve one or more related records from a child table. A parent-child relationship can be defined by using a *DataRelation* object.

polymorphism

The ability of classes to provide different implementations of the same public interface. Thus, two classes that exhibit polymorphism might contain different implementations, but because the interfaces are identical, they can be treated the same in code.

primary key

In a data table, the column that uniquely identifies a record in the table.

private assembly

An assembly that is used by only one application. In .NET Framework programming, most assemblies are private assemblies.

project

The collection of Visual Studio .NET source files that make up an application.

R

Reference type

A type, such as a class, whose instance data is allocated on the heap. Variables that represent Reference types contain a reference to the object on the heap as opposed to the object's data itself. *See also* Value type.

registry

The system repository for information about a computer's settings. The registry can contain configuration information

This document is created with the unregistered version of CHM2PDF Pilot

about programs and system components.

resource

Nonexecutable data, such as a string or image, that is deployed in an application.

role-based security

Code that ensures that users of your application have appropriate permission to access your application based on the identity of the user.

run time

Represents the state of the application while executing. At run time, the application consists of executing code.

run-time error

An error that causes an invalid result or attempts an invalid operation at run time.

S

satellite assembly

When localizing an application, assemblies that contain alternate sets of resources to be used in the application for different cultures.

setter

The code segment of a property that sets the property's value.

shadowing

Hiding a member in a base class with a member in an inherited class that has a different signature.

shared assembly

This document is created with the unregistered version of CHM2PDF Pilot

An assembly that is referenced by several applications on a single machine. A shared assembly must be installed to the Global Assembly Cache (GAC).

shared members

In Visual Basic .NET, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Shared members are called static members in Visual C#.

simple binding

In data binding, binding a control to a single data column. A *TextBox* is an example of a control that can be simply bound.

solution

A group of Visual Studio .NET projects that make up a single functional unit.

SQL

See Structured Query Language (SQL).

static members

In Visual C#, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Static members are called shared members in Visual Basic .NET.

Step Into

Moving from a calling procedure to a called procedure during debugging. Also used in reference to setting breakpoints to stop execution in a specific procedure during debugging.

Step Out

In Break mode, resuming execution of a function call until the call returns.

Step Over

In Break mode, executing a procedure call as a single statement during debugging.

This document is created with the unregistered version of CHM2PDF Pilot

strong name

A name that uniquely identifies an assembly. It consists of the assembly's name, version, and culture information as well as a public key encrypted by a private key. An assembly must have a strong name to be installed to the Global Assembly Cache (GAC).

strongly typed

Refers to types for all variables being explicitly declared. Strong typing eliminates the possibility of type mismatch errors, which are discovered at compile time. Visual C# is inherently strongly typed, and Visual Basic .NET is strongly typed when Option Strict is set to On.

Struct

In Visual C#, a user-defined Value type that serves as a template for an instance of that type. *See also* Structure.

Structure

In Visual Basic .NET, a user-defined Value type that serves as a template for an instance of that type. *See also* Struct .

structured exception handling

Using *Try...Catch...Finally* blocks to handle errors that might occur during code execution. When exceptions are thrown, they are handled in the *Catch* block before code in the *Finally* block is executed.

Structured Query Language (SQL)

The standard language used for issuing commands to databases.

syntax error

An error that occurs because of typographical errors or nonsensical code. In the .NET Framework, syntax errors prevent compilation and are thus discovered at compile time.

T

target audience

This document is created with the unregistered version of CHM2PDF Pilot

The group of end users for whom an application is designed.

threads

The basic unit to which the server allocates processor time. A single process can have multiple threads.

throwing

The act of passing an exception to the next caller in the call stack. *See also* exception.

tracing

A programming technique for recording events, such as exceptions, in an application. Tracing is used during debugging and in the testing phase of application deployment.

transaction

A group of commands (treated as a single unit) that change the data stored in a database. The transaction ensures that the commands are handled in an all-or-nothing fashion if one of the commands fails, all of the commands fail, and any data that was written to the database by the commands is backed out. In this way, transactions maintain the integrity of data in a database.

tuning

The process of making adjustments to a deployed application that do not affect code.

U

Unicode

The universal character encoding scheme for characters and text. Each character is represented by a unique numeric value.

unmanaged code

Code that is not managed by the common language runtime. Unmanaged code is not checked for type-safety and

This document is created with the unregistered version of CHM2PDF Pilot

must be used with extreme care.

user control

A user-defined control that encapsulates other Windows Forms controls with custom functionality in a common container.

V

Value type

A type, such as an Integer, Decimal, or Structure, whose instance data is allocated on the stack. Variables that represent Value types contain the object's instance data itself as opposed to a reference to that data. *See also* Reference type.

W

Watch window

When debugging, the window that allows you to examine the values of predetermined watch variables.

Web Service

A service or component that is implemented by a Web server and can be instantiated and accessed programmatically as a component over the World Wide Web.

X

XML Schema

A description of the data elements contained in an XML file. The XML Schema provides the names of the elements, their types, whether or not they are key fields, and other information.

This document is created with the unregistered version of CHM2PDF Pilot

Glossary

A

abstract class

A class that cannot be instantiated but is used as a base from which other classes can be derived. In Microsoft Visual Basic .NET, abstract classes are declared using the *MustInherit* keyword. In Microsoft Visual C#, abstract classes are declared using the *abstract* keyword.

abstraction

Representing real-world objects as programmatic constructs.

abstract members

A member of a base class that cannot be invoked, but instead provides a template for members of a derived class. In Visual Basic .NET, abstract members are declared using the *MustOverride* keyword. In Visual C#, abstract members are declared using the *abstract* keyword.

ADO.NET

The data access architecture for the Microsoft .NET Framework. ADO.NET is built around a disconnected data access model that uses a set of classes called a Data Provider to retrieve data from a data source.

alias

The name you use to identify resources on the Web. Aliases represent physical resources on the Web server, such as a Web form, an HTML page, or a graphic.

anchoring

In Windows Forms controls, the resizing and repositioning of a control when its parent form is resized. You can anchor a control by setting the *Anchor* property.

This document is created with the unregistered version of CHM2PDF Pilot

assembly

The primary unit of deployment for a .NET application. An assembly is either an executable application (EXE) or a class library (DLL).

assembly manifest

The metadata for the assembly. It contains all of the information needed to describe the assembly to the common language runtime.

authentication

The process of determining the identity of a user. In effect, authentication validates that the users are who they say they are. *See also* authorization.

authorization

The process of allowing access to an application or resource based on credentials supplied by the user. *See also* authentication.

B

base class

A class that provides properties and methods as a foundation for a derived class. In objected-oriented programming, one class can be based on another through inheritance. Using this technique, the base class provides characteristics (such as properties and methods) to a derived class. The derived class can reuse, modify, or add to the members of the base class.

Bezier

A curve specified by four points. Two points specify the ends of the curve, and the other two points influence the shape of the curve between the end points.

Boolean

A data type that can represent a value of either true or false.

This document is created with the unregistered version of CHM2PDF Pilot

bootstrapper

In deployment, an application that installs Microsoft Windows Installer on machines that do not currently have it installed.

boundary condition

In testing, test conditions at the limit of normal operation and just over or under those limits.

boxing

The act of wrapping a Value type, such as an Integer, inside an object so that it can be treated like a Reference type.

Break mode

In debugging, a state of suspended execution wherein the application can be executed in a line-by-line manner, and individual program variables can be examined.

breakpoint

In debugging, a predetermined point where execution of an application will halt and enter Break mode.

C

cast

An explicit conversion from one type to another.

circular reference

A condition that occurs when two objects refer to each other but do not have an external reference. Circular references are automatically detected and disposed of by the garbage collector.

class

A user-defined Reference type that serves as a template for an object of that type.

This document is created with the unregistered version of CHM2PDF Pilot

CLR

See common language runtime (CLR).

code access security

Verification that all callers to code have appropriate permission to access it.

code tuning

The act of examining application code and optimizing for faster execution. Code tuning is usually an iterative process that involves several successive rounds of planning, improvement, and assessment of the results.

common language runtime (CLR)

The environment in which managed code executes. The common language runtime provides just-in-time compilation, enforces type safety, and manages memory through garbage collection.

Common Type System

A set of types that are used by all .NET languages, thus ensuring .NET language type compatibility.

complex binding

In data binding, binding more than one column to a single control. The *DataGrid* is an example of a control that can be complexly bound.

component

Strictly speaking, a class that implements *IComponent* or inherits *System*.*Component*. More generally, any class that provides a reusable set of functionality.

connected data access

Data access that requires a data connection that is always open.

Constant

This document is created with the unregistered version of CHM2PDF Pilot

A type that represents a value that does not change in the course of an application's execution and cannot be changed.

constraint

In ADO.NET, an object that represents a rule that enforces referential integrity, such as a constraint that specifies a primary key.

constructor

The method that executes when a class is instantiated. The constructor usually contains initialization code to set member variables to initial values.

container controls

Controls that can host other controls. Examples include *Forms*, *Panels*, and *TabPages*.

control

A class that inherits *System*.*Control*. Controls usually have a visual representation, interact with the user as a part of the user interface, and are hosted in a container control.

controls collection

The collection of a container control that manages the controls it hosts.

custom control

A user-designed control that directly inherits *System*.*Control*.

D

data binding

A way to link data (such as a data set) in your application to the properties of a control.

data consumer

This document is created with the unregistered version of CHM2PDF Pilot

A control that is bound to a data source.

Data Provider

A set of classes that work together to provide managed data access.

DataSet

A class that contains an application copy of data that has been retrieved from a data source.

delegate

An object that acts as a type-safe function pointer. Delegates can be used to invoke the method they point to.

deploying

Installing an application on the computer where it will run.

derived class

A class that is based on another class (called a base class) through inheritance. A derived class inherits the members of its base class and can override or shadow those members.

design time

Represents the state of the application while being designed. Applications at design time consist of noncompiled code that can be manipulated in the code editor or designer window.

destructor

The method that is run just before a class is removed from memory. Generally, a destructor contains any code required to "clean up" after the class. *See also* finalizer.

disconnected data access

Data access that uses a data connection only when executing database operations. This is the default kind of data access implemented by ADO.NET.

This document is created with the unregistered version of CHM2PDF Pilot

docking

In Windows Forms, attaching a control to the edge of the form. You can dock a control by setting the *Dock* property.

E

encapsulation

In component programming, separating the implementation of a component from the interface. Only the public interface of a component is made accessible to the rest of the application. Component data should never be accessible to outside callers.

event

A notification raised by a control or component that a specified condition has occurred.

event handler

A method that is executed when a particular event is raised.

exception

Any class that derives from *System.Exception*. Exceptions are thrown in response to run-time errors. *See also* throwing.

exception handling

The process of dealing with unusual occurrences within code so that it does not cause the program to crash or lose data.

explicit conversion

A conversion of one type to another that cannot be performed automatically. An explicit conversion usually presents some danger of a failed conversion or a loss of data.

extender provider

This document is created with the unregistered version of CHM2PDF Pilot

A specialized type of control that provides design time properties to other controls. *ErrorProvider* and *HelpProvider* are examples of extender providers.

F

field

A member variable of a class or structure.

finalizer

In Visual Basic .NET, the method that is called just before an instance of a class is destroyed. *See also* destructor.

floating-point numbers

A number that has a decimal component. *See also* floating-point types.

floating-point types

Types capable of representing floating-point numbers. Floating-point types include Single (float) and Double (double). *See also* floating-point numbers.

focus

When referring to a control, the condition of being able to receive user input through the keyboard or other input devices. A control that is capable of receiving input is said to have the focus.

foreign key

In a data table, a column that contains a key value that uniquely identifies a record in a related table.

form

The primary unit of the user interface in a Windows Forms application. A form serves as a nexus for data display and user input, and can host controls.

This document is created with the unregistered version of CHM2PDF Pilot

G

GAC

See Global Assembly Cache (GAC).

garbage collection

Automatic memory management provided by the common language runtime. Unused memory is automatically reclaimed by garbage collection without interaction with the application.

GDI+

See Graphic Device Interface (GDI).

getter

The code segment of a property that returns the value the property contains.

Global Assembly Cache (GAC)

A machine-wide cache that stores assemblies that have been specifically designated to be shared between several applications.

globalization

Applying culture-based formatting to existing data, for example, using culture-specific currency symbols and formatting.

Graphic Device Interface (GDI)

In Windows, a graphics display system used by applications to display or print bitmapped text (TrueType fonts), images, and other graphical elements. GDI+ is the name given to the .NET Framework's managed implementation of the GDI.

I

This document is created with the unregistered version of CHM2PDF Pilot

implicit conversion

A conversion between types that can be performed automatically by the common language runtime. An implicit conversion will always succeed and never pose a danger of data loss.

inheritance

In object-oriented programming, creating a new class that incorporates and extends all of the characteristics and functionality of a base class.

inherited control

A user-created control that inherits an existing control. An inherited control inherits all of the functionality of the base control and can incorporate custom functionality as well.

interface

A contract for behavior that can be implemented by a class. Classes that implement the same interface can be expected to provide the members defined by the interface.

L

localization

Providing multiple sets of data specific to different cultures and retrieving the data appropriate to the current culture setting.

logical error

An error that compiles and executes correctly but produces unexpected results. These can be the most difficult errors to detect in application testing.

M

This document is created with the unregistered version of CHM2PDF Pilot

managed code

Code that runs under the common language runtime. The common language runtime handles many tasks that would formerly have been handled in the application's executable. Managed code solves the Windows programming problems of component registration and versioning (sometimes called DLL Hell) because managed code contains all the versioning and type information that the common language runtime needs to run the application. The common language runtime handles registration dynamically at run time rather than statically through the system registry, as is done with applications based on the Component Object Model (COM).

metadata

Data about data. For example, the name, version, security information, and size of a file constitute metadata about the file.

Microsoft Intermediate Language (MSIL)

A low-level language that is just-in-time compiled to native code at run time. All .NET assemblies are represented in the MSIL.

MSIL

See Microsoft Intermediate Language (MSIL).

multidimensional array

An array that contains more than one dimension.

N

namespace

A logical organization of types that perform related functions.

.NET Framework

This document is created with the unregistered version of CHM2PDF Pilot

A managed environment for the development and execution of code. It consists of the common language runtime and .NET Framework base class library.

.NET Framework base class library

A collection of object-oriented types and interfaces that provides object models to assist in the development of your applications.

O

object

The base class for all types in the .NET Framework. More generically, any instance of a class or structure.

object-oriented programming

The process of creating applications by using programmatic constructs to represent real-world objects.

operator overloading

In Visual C#, defining custom behavior for an operator when interacting with custom types.

optimization

The process of examining your application and modifying it to be more efficient in terms of either speed or size of the deployed application.

overloading

Creating multiple methods with the same name but different signatures.

overriding

When inheriting from a base class, providing a new method for one of the base class methods.

P

This document is created with the unregistered version of CHM2PDF Pilot

parent-child relationship

In data access, a relationship between two tables wherein a record from the parent table can be used to retrieve one or more related records from a child table. A parent-child relationship can be defined by using a *DataRelation* object.

polymorphism

The ability of classes to provide different implementations of the same public interface. Thus, two classes that exhibit polymorphism might contain different implementations, but because the interfaces are identical, they can be treated the same in code.

primary key

In a data table, the column that uniquely identifies a record in the table.

private assembly

An assembly that is used by only one application. In .NET Framework programming, most assemblies are private assemblies.

project

The collection of Visual Studio .NET source files that make up an application.

R

Reference type

A type, such as a class, whose instance data is allocated on the heap. Variables that represent Reference types contain a reference to the object on the heap as opposed to the object's data itself. *See also* Value type.

registry

The system repository for information about a computer's settings. The registry can contain configuration information

This document is created with the unregistered version of CHM2PDF Pilot

about programs and system components.

resource

Nonexecutable data, such as a string or image, that is deployed in an application.

role-based security

Code that ensures that users of your application have appropriate permission to access your application based on the identity of the user.

run time

Represents the state of the application while executing. At run time, the application consists of executing code.

run-time error

An error that causes an invalid result or attempts an invalid operation at run time.

S

satellite assembly

When localizing an application, assemblies that contain alternate sets of resources to be used in the application for different cultures.

setter

The code segment of a property that sets the property's value.

shadowing

Hiding a member in a base class with a member in an inherited class that has a different signature.

shared assembly

This document is created with the unregistered version of CHM2PDF Pilot

An assembly that is referenced by several applications on a single machine. A shared assembly must be installed to the Global Assembly Cache (GAC).

shared members

In Visual Basic .NET, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Shared members are called static members in Visual C#.

simple binding

In data binding, binding a control to a single data column. A *TextBox* is an example of a control that can be simply bound.

solution

A group of Visual Studio .NET projects that make up a single functional unit.

SQL

See Structured Query Language (SQL).

static members

In Visual C#, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Static members are called shared members in Visual Basic .NET.

Step Into

Moving from a calling procedure to a called procedure during debugging. Also used in reference to setting breakpoints to stop execution in a specific procedure during debugging.

Step Out

In Break mode, resuming execution of a function call until the call returns.

Step Over

In Break mode, executing a procedure call as a single statement during debugging.

This document is created with the unregistered version of CHM2PDF Pilot

strong name

A name that uniquely identifies an assembly. It consists of the assembly's name, version, and culture information as well as a public key encrypted by a private key. An assembly must have a strong name to be installed to the Global Assembly Cache (GAC).

strongly typed

Refers to types for all variables being explicitly declared. Strong typing eliminates the possibility of type mismatch errors, which are discovered at compile time. Visual C# is inherently strongly typed, and Visual Basic .NET is strongly typed when Option Strict is set to On.

Struct

In Visual C#, a user-defined Value type that serves as a template for an instance of that type. *See also* Structure.

Structure

In Visual Basic .NET, a user-defined Value type that serves as a template for an instance of that type. *See also* Struct .

structured exception handling

Using *Try...Catch...Finally* blocks to handle errors that might occur during code execution. When exceptions are thrown, they are handled in the *Catch* block before code in the *Finally* block is executed.

Structured Query Language (SQL)

The standard language used for issuing commands to databases.

syntax error

An error that occurs because of typographical errors or nonsensical code. In the .NET Framework, syntax errors prevent compilation and are thus discovered at compile time.

T

target audience

This document is created with the unregistered version of CHM2PDF Pilot

The group of end users for whom an application is designed.

threads

The basic unit to which the server allocates processor time. A single process can have multiple threads.

throwing

The act of passing an exception to the next caller in the call stack. *See also* exception.

tracing

A programming technique for recording events, such as exceptions, in an application. Tracing is used during debugging and in the testing phase of application deployment.

transaction

A group of commands (treated as a single unit) that change the data stored in a database. The transaction ensures that the commands are handled in an all-or-nothing fashion if one of the commands fails, all of the commands fail, and any data that was written to the database by the commands is backed out. In this way, transactions maintain the integrity of data in a database.

tuning

The process of making adjustments to a deployed application that do not affect code.

U

Unicode

The universal character encoding scheme for characters and text. Each character is represented by a unique numeric value.

unmanaged code

Code that is not managed by the common language runtime. Unmanaged code is not checked for type-safety and

This document is created with the unregistered version of CHM2PDF Pilot

must be used with extreme care.

user control

A user-defined control that encapsulates other Windows Forms controls with custom functionality in a common container.

V

Value type

A type, such as an Integer, Decimal, or Structure, whose instance data is allocated on the stack. Variables that represent Value types contain the object's instance data itself as opposed to a reference to that data. *See also* Reference type.

W

Watch window

When debugging, the window that allows you to examine the values of predetermined watch variables.

Web Service

A service or component that is implemented by a Web server and can be instantiated and accessed programmatically as a component over the World Wide Web.

X

XML Schema

A description of the data elements contained in an XML file. The XML Schema provides the names of the elements, their types, whether or not they are key fields, and other information.

This document is created with the unregistered version of CHM2PDF Pilot

Glossary

A

abstract class

A class that cannot be instantiated but is used as a base from which other classes can be derived. In Microsoft Visual Basic .NET, abstract classes are declared using the *MustInherit* keyword. In Microsoft Visual C#, abstract classes are declared using the *abstract* keyword.

abstraction

Representing real-world objects as programmatic constructs.

abstract members

A member of a base class that cannot be invoked, but instead provides a template for members of a derived class. In Visual Basic .NET, abstract members are declared using the *MustOverride* keyword. In Visual C#, abstract members are declared using the *abstract* keyword.

ADO.NET

The data access architecture for the Microsoft .NET Framework. ADO.NET is built around a disconnected data access model that uses a set of classes called a Data Provider to retrieve data from a data source.

alias

The name you use to identify resources on the Web. Aliases represent physical resources on the Web server, such as a Web form, an HTML page, or a graphic.

anchoring

In Windows Forms controls, the resizing and repositioning of a control when its parent form is resized. You can anchor a control by setting the *Anchor* property.

This document is created with the unregistered version of CHM2PDF Pilot

assembly

The primary unit of deployment for a .NET application. An assembly is either an executable application (EXE) or a class library (DLL).

assembly manifest

The metadata for the assembly. It contains all of the information needed to describe the assembly to the common language runtime.

authentication

The process of determining the identity of a user. In effect, authentication validates that the users are who they say they are. *See also* authorization.

authorization

The process of allowing access to an application or resource based on credentials supplied by the user. *See also* authentication.

B

base class

A class that provides properties and methods as a foundation for a derived class. In objected-oriented programming, one class can be based on another through inheritance. Using this technique, the base class provides characteristics (such as properties and methods) to a derived class. The derived class can reuse, modify, or add to the members of the base class.

Bezier

A curve specified by four points. Two points specify the ends of the curve, and the other two points influence the shape of the curve between the end points.

Boolean

A data type that can represent a value of either true or false.

This document is created with the unregistered version of CHM2PDF Pilot

bootstrapper

In deployment, an application that installs Microsoft Windows Installer on machines that do not currently have it installed.

boundary condition

In testing, test conditions at the limit of normal operation and just over or under those limits.

boxing

The act of wrapping a Value type, such as an Integer, inside an object so that it can be treated like a Reference type.

Break mode

In debugging, a state of suspended execution wherein the application can be executed in a line-by-line manner, and individual program variables can be examined.

breakpoint

In debugging, a predetermined point where execution of an application will halt and enter Break mode.

C

cast

An explicit conversion from one type to another.

circular reference

A condition that occurs when two objects refer to each other but do not have an external reference. Circular references are automatically detected and disposed of by the garbage collector.

class

A user-defined Reference type that serves as a template for an object of that type.

This document is created with the unregistered version of CHM2PDF Pilot

CLR

See common language runtime (CLR).

code access security

Verification that all callers to code have appropriate permission to access it.

code tuning

The act of examining application code and optimizing for faster execution. Code tuning is usually an iterative process that involves several successive rounds of planning, improvement, and assessment of the results.

common language runtime (CLR)

The environment in which managed code executes. The common language runtime provides just-in-time compilation, enforces type safety, and manages memory through garbage collection.

Common Type System

A set of types that are used by all .NET languages, thus ensuring .NET language type compatibility.

complex binding

In data binding, binding more than one column to a single control. The *DataGrid* is an example of a control that can be complexly bound.

component

Strictly speaking, a class that implements *IComponent* or inherits *System*.*Component*. More generally, any class that provides a reusable set of functionality.

connected data access

Data access that requires a data connection that is always open.

Constant

This document is created with the unregistered version of CHM2PDF Pilot

A type that represents a value that does not change in the course of an application's execution and cannot be changed.

constraint

In ADO.NET, an object that represents a rule that enforces referential integrity, such as a constraint that specifies a primary key.

constructor

The method that executes when a class is instantiated. The constructor usually contains initialization code to set member variables to initial values.

container controls

Controls that can host other controls. Examples include *Forms*, *Panels*, and *TabPages*.

control

A class that inherits *System*.*Control*. Controls usually have a visual representation, interact with the user as a part of the user interface, and are hosted in a container control.

controls collection

The collection of a container control that manages the controls it hosts.

custom control

A user-designed control that directly inherits *System*.*Control*.

D

data binding

A way to link data (such as a data set) in your application to the properties of a control.

data consumer

This document is created with the unregistered version of CHM2PDF Pilot

A control that is bound to a data source.

Data Provider

A set of classes that work together to provide managed data access.

DataSet

A class that contains an application copy of data that has been retrieved from a data source.

delegate

An object that acts as a type-safe function pointer. Delegates can be used to invoke the method they point to.

deploying

Installing an application on the computer where it will run.

derived class

A class that is based on another class (called a base class) through inheritance. A derived class inherits the members of its base class and can override or shadow those members.

design time

Represents the state of the application while being designed. Applications at design time consist of noncompiled code that can be manipulated in the code editor or designer window.

destructor

The method that is run just before a class is removed from memory. Generally, a destructor contains any code required to "clean up" after the class. *See also* finalizer.

disconnected data access

Data access that uses a data connection only when executing database operations. This is the default kind of data access implemented by ADO.NET.

This document is created with the unregistered version of CHM2PDF Pilot

docking

In Windows Forms, attaching a control to the edge of the form. You can dock a control by setting the *Dock* property.

E

encapsulation

In component programming, separating the implementation of a component from the interface. Only the public interface of a component is made accessible to the rest of the application. Component data should never be accessible to outside callers.

event

A notification raised by a control or component that a specified condition has occurred.

event handler

A method that is executed when a particular event is raised.

exception

Any class that derives from *System.Exception*. Exceptions are thrown in response to run-time errors. *See also* throwing.

exception handling

The process of dealing with unusual occurrences within code so that it does not cause the program to crash or lose data.

explicit conversion

A conversion of one type to another that cannot be performed automatically. An explicit conversion usually presents some danger of a failed conversion or a loss of data.

extender provider

This document is created with the unregistered version of CHM2PDF Pilot

A specialized type of control that provides design time properties to other controls. *ErrorProvider* and *HelpProvider* are examples of extender providers.

F

field

A member variable of a class or structure.

finalizer

In Visual Basic .NET, the method that is called just before an instance of a class is destroyed. *See also* destructor.

floating-point numbers

A number that has a decimal component. *See also* floating-point types.

floating-point types

Types capable of representing floating-point numbers. Floating-point types include Single (float) and Double (double). *See also* floating-point numbers.

focus

When referring to a control, the condition of being able to receive user input through the keyboard or other input devices. A control that is capable of receiving input is said to have the focus.

foreign key

In a data table, a column that contains a key value that uniquely identifies a record in a related table.

form

The primary unit of the user interface in a Windows Forms application. A form serves as a nexus for data display and user input, and can host controls.

This document is created with the unregistered version of CHM2PDF Pilot

G

GAC

See Global Assembly Cache (GAC).

garbage collection

Automatic memory management provided by the common language runtime. Unused memory is automatically reclaimed by garbage collection without interaction with the application.

GDI+

See Graphic Device Interface (GDI).

getter

The code segment of a property that returns the value the property contains.

Global Assembly Cache (GAC)

A machine-wide cache that stores assemblies that have been specifically designated to be shared between several applications.

globalization

Applying culture-based formatting to existing data, for example, using culture-specific currency symbols and formatting.

Graphic Device Interface (GDI)

In Windows, a graphics display system used by applications to display or print bitmapped text (TrueType fonts), images, and other graphical elements. GDI+ is the name given to the .NET Framework's managed implementation of the GDI.

I

This document is created with the unregistered version of CHM2PDF Pilot

implicit conversion

A conversion between types that can be performed automatically by the common language runtime. An implicit conversion will always succeed and never pose a danger of data loss.

inheritance

In object-oriented programming, creating a new class that incorporates and extends all of the characteristics and functionality of a base class.

inherited control

A user-created control that inherits an existing control. An inherited control inherits all of the functionality of the base control and can incorporate custom functionality as well.

interface

A contract for behavior that can be implemented by a class. Classes that implement the same interface can be expected to provide the members defined by the interface.

L

localization

Providing multiple sets of data specific to different cultures and retrieving the data appropriate to the current culture setting.

logical error

An error that compiles and executes correctly but produces unexpected results. These can be the most difficult errors to detect in application testing.

M

This document is created with the unregistered version of CHM2PDF Pilot

managed code

Code that runs under the common language runtime. The common language runtime handles many tasks that would formerly have been handled in the application's executable. Managed code solves the Windows programming problems of component registration and versioning (sometimes called DLL Hell) because managed code contains all the versioning and type information that the common language runtime needs to run the application. The common language runtime handles registration dynamically at run time rather than statically through the system registry, as is done with applications based on the Component Object Model (COM).

metadata

Data about data. For example, the name, version, security information, and size of a file constitute metadata about the file.

Microsoft Intermediate Language (MSIL)

A low-level language that is just-in-time compiled to native code at run time. All .NET assemblies are represented in the MSIL.

MSIL

See Microsoft Intermediate Language (MSIL).

multidimensional array

An array that contains more than one dimension.

N

namespace

A logical organization of types that perform related functions.

.NET Framework

This document is created with the unregistered version of CHM2PDF Pilot

A managed environment for the development and execution of code. It consists of the common language runtime and .NET Framework base class library.

.NET Framework base class library

A collection of object-oriented types and interfaces that provides object models to assist in the development of your applications.

O

object

The base class for all types in the .NET Framework. More generically, any instance of a class or structure.

object-oriented programming

The process of creating applications by using programmatic constructs to represent real-world objects.

operator overloading

In Visual C#, defining custom behavior for an operator when interacting with custom types.

optimization

The process of examining your application and modifying it to be more efficient in terms of either speed or size of the deployed application.

overloading

Creating multiple methods with the same name but different signatures.

overriding

When inheriting from a base class, providing a new method for one of the base class methods.

P

This document is created with the unregistered version of CHM2PDF Pilot

parent-child relationship

In data access, a relationship between two tables wherein a record from the parent table can be used to retrieve one or more related records from a child table. A parent-child relationship can be defined by using a *DataRelation* object.

polymorphism

The ability of classes to provide different implementations of the same public interface. Thus, two classes that exhibit polymorphism might contain different implementations, but because the interfaces are identical, they can be treated the same in code.

primary key

In a data table, the column that uniquely identifies a record in the table.

private assembly

An assembly that is used by only one application. In .NET Framework programming, most assemblies are private assemblies.

project

The collection of Visual Studio .NET source files that make up an application.

R

Reference type

A type, such as a class, whose instance data is allocated on the heap. Variables that represent Reference types contain a reference to the object on the heap as opposed to the object's data itself. *See also* Value type.

registry

The system repository for information about a computer's settings. The registry can contain configuration information

This document is created with the unregistered version of CHM2PDF Pilot

about programs and system components.

resource

Nonexecutable data, such as a string or image, that is deployed in an application.

role-based security

Code that ensures that users of your application have appropriate permission to access your application based on the identity of the user.

run time

Represents the state of the application while executing. At run time, the application consists of executing code.

run-time error

An error that causes an invalid result or attempts an invalid operation at run time.

S

satellite assembly

When localizing an application, assemblies that contain alternate sets of resources to be used in the application for different cultures.

setter

The code segment of a property that sets the property's value.

shadowing

Hiding a member in a base class with a member in an inherited class that has a different signature.

shared assembly

This document is created with the unregistered version of CHM2PDF Pilot

An assembly that is referenced by several applications on a single machine. A shared assembly must be installed to the Global Assembly Cache (GAC).

shared members

In Visual Basic .NET, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Shared members are called static members in Visual C#.

simple binding

In data binding, binding a control to a single data column. A *TextBox* is an example of a control that can be simply bound.

solution

A group of Visual Studio .NET projects that make up a single functional unit.

SQL

See Structured Query Language (SQL).

static members

In Visual C#, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Static members are called shared members in Visual Basic .NET.

Step Into

Moving from a calling procedure to a called procedure during debugging. Also used in reference to setting breakpoints to stop execution in a specific procedure during debugging.

Step Out

In Break mode, resuming execution of a function call until the call returns.

Step Over

In Break mode, executing a procedure call as a single statement during debugging.

This document is created with the unregistered version of CHM2PDF Pilot

strong name

A name that uniquely identifies an assembly. It consists of the assembly's name, version, and culture information as well as a public key encrypted by a private key. An assembly must have a strong name to be installed to the Global Assembly Cache (GAC).

strongly typed

Refers to types for all variables being explicitly declared. Strong typing eliminates the possibility of type mismatch errors, which are discovered at compile time. Visual C# is inherently strongly typed, and Visual Basic .NET is strongly typed when Option Strict is set to On.

Struct

In Visual C#, a user-defined Value type that serves as a template for an instance of that type. *See also* Structure.

Structure

In Visual Basic .NET, a user-defined Value type that serves as a template for an instance of that type. *See also* Struct .

structured exception handling

Using *Try...Catch...Finally* blocks to handle errors that might occur during code execution. When exceptions are thrown, they are handled in the *Catch* block before code in the *Finally* block is executed.

Structured Query Language (SQL)

The standard language used for issuing commands to databases.

syntax error

An error that occurs because of typographical errors or nonsensical code. In the .NET Framework, syntax errors prevent compilation and are thus discovered at compile time.

T

target audience

This document is created with the unregistered version of CHM2PDF Pilot

The group of end users for whom an application is designed.

threads

The basic unit to which the server allocates processor time. A single process can have multiple threads.

throwing

The act of passing an exception to the next caller in the call stack. *See also* exception.

tracing

A programming technique for recording events, such as exceptions, in an application. Tracing is used during debugging and in the testing phase of application deployment.

transaction

A group of commands (treated as a single unit) that change the data stored in a database. The transaction ensures that the commands are handled in an all-or-nothing fashion if one of the commands fails, all of the commands fail, and any data that was written to the database by the commands is backed out. In this way, transactions maintain the integrity of data in a database.

tuning

The process of making adjustments to a deployed application that do not affect code.

U

Unicode

The universal character encoding scheme for characters and text. Each character is represented by a unique numeric value.

unmanaged code

Code that is not managed by the common language runtime. Unmanaged code is not checked for type-safety and

This document is created with the unregistered version of CHM2PDF Pilot

must be used with extreme care.

user control

A user-defined control that encapsulates other Windows Forms controls with custom functionality in a common container.

V

Value type

A type, such as an Integer, Decimal, or Structure, whose instance data is allocated on the stack. Variables that represent Value types contain the object's instance data itself as opposed to a reference to that data. *See also* Reference type.

W

Watch window

When debugging, the window that allows you to examine the values of predetermined watch variables.

Web Service

A service or component that is implemented by a Web server and can be instantiated and accessed programmatically as a component over the World Wide Web.

X

XML Schema

A description of the data elements contained in an XML file. The XML Schema provides the names of the elements, their types, whether or not they are key fields, and other information.

This document is created with the unregistered version of CHM2PDF Pilot

Glossary

A

abstract class

A class that cannot be instantiated but is used as a base from which other classes can be derived. In Microsoft Visual Basic .NET, abstract classes are declared using the *MustInherit* keyword. In Microsoft Visual C#, abstract classes are declared using the *abstract* keyword.

abstraction

Representing real-world objects as programmatic constructs.

abstract members

A member of a base class that cannot be invoked, but instead provides a template for members of a derived class. In Visual Basic .NET, abstract members are declared using the *MustOverride* keyword. In Visual C#, abstract members are declared using the *abstract* keyword.

ADO.NET

The data access architecture for the Microsoft .NET Framework. ADO.NET is built around a disconnected data access model that uses a set of classes called a Data Provider to retrieve data from a data source.

alias

The name you use to identify resources on the Web. Aliases represent physical resources on the Web server, such as a Web form, an HTML page, or a graphic.

anchoring

In Windows Forms controls, the resizing and repositioning of a control when its parent form is resized. You can anchor a control by setting the *Anchor* property.

This document is created with the unregistered version of CHM2PDF Pilot

assembly

The primary unit of deployment for a .NET application. An assembly is either an executable application (EXE) or a class library (DLL).

assembly manifest

The metadata for the assembly. It contains all of the information needed to describe the assembly to the common language runtime.

authentication

The process of determining the identity of a user. In effect, authentication validates that the users are who they say they are. *See also* authorization.

authorization

The process of allowing access to an application or resource based on credentials supplied by the user. *See also* authentication.

B

base class

A class that provides properties and methods as a foundation for a derived class. In objected-oriented programming, one class can be based on another through inheritance. Using this technique, the base class provides characteristics (such as properties and methods) to a derived class. The derived class can reuse, modify, or add to the members of the base class.

Bezier

A curve specified by four points. Two points specify the ends of the curve, and the other two points influence the shape of the curve between the end points.

Boolean

A data type that can represent a value of either true or false.

This document is created with the unregistered version of CHM2PDF Pilot

bootstrapper

In deployment, an application that installs Microsoft Windows Installer on machines that do not currently have it installed.

boundary condition

In testing, test conditions at the limit of normal operation and just over or under those limits.

boxing

The act of wrapping a Value type, such as an Integer, inside an object so that it can be treated like a Reference type.

Break mode

In debugging, a state of suspended execution wherein the application can be executed in a line-by-line manner, and individual program variables can be examined.

breakpoint

In debugging, a predetermined point where execution of an application will halt and enter Break mode.

C

cast

An explicit conversion from one type to another.

circular reference

A condition that occurs when two objects refer to each other but do not have an external reference. Circular references are automatically detected and disposed of by the garbage collector.

class

A user-defined Reference type that serves as a template for an object of that type.

This document is created with the unregistered version of CHM2PDF Pilot

CLR

See common language runtime (CLR).

code access security

Verification that all callers to code have appropriate permission to access it.

code tuning

The act of examining application code and optimizing for faster execution. Code tuning is usually an iterative process that involves several successive rounds of planning, improvement, and assessment of the results.

common language runtime (CLR)

The environment in which managed code executes. The common language runtime provides just-in-time compilation, enforces type safety, and manages memory through garbage collection.

Common Type System

A set of types that are used by all .NET languages, thus ensuring .NET language type compatibility.

complex binding

In data binding, binding more than one column to a single control. The *DataGrid* is an example of a control that can be complexly bound.

component

Strictly speaking, a class that implements *IComponent* or inherits *System*.*Component*. More generally, any class that provides a reusable set of functionality.

connected data access

Data access that requires a data connection that is always open.

Constant

This document is created with the unregistered version of CHM2PDF Pilot

A type that represents a value that does not change in the course of an application's execution and cannot be changed.

constraint

In ADO.NET, an object that represents a rule that enforces referential integrity, such as a constraint that specifies a primary key.

constructor

The method that executes when a class is instantiated. The constructor usually contains initialization code to set member variables to initial values.

container controls

Controls that can host other controls. Examples include *Forms*, *Panels*, and *TabPages*.

control

A class that inherits *System*.*Control*. Controls usually have a visual representation, interact with the user as a part of the user interface, and are hosted in a container control.

controls collection

The collection of a container control that manages the controls it hosts.

custom control

A user-designed control that directly inherits *System*.*Control*.

D

data binding

A way to link data (such as a data set) in your application to the properties of a control.

data consumer

This document is created with the unregistered version of CHM2PDF Pilot

A control that is bound to a data source.

Data Provider

A set of classes that work together to provide managed data access.

DataSet

A class that contains an application copy of data that has been retrieved from a data source.

delegate

An object that acts as a type-safe function pointer. Delegates can be used to invoke the method they point to.

deploying

Installing an application on the computer where it will run.

derived class

A class that is based on another class (called a base class) through inheritance. A derived class inherits the members of its base class and can override or shadow those members.

design time

Represents the state of the application while being designed. Applications at design time consist of noncompiled code that can be manipulated in the code editor or designer window.

destructor

The method that is run just before a class is removed from memory. Generally, a destructor contains any code required to "clean up" after the class. *See also* finalizer.

disconnected data access

Data access that uses a data connection only when executing database operations. This is the default kind of data access implemented by ADO.NET.

This document is created with the unregistered version of CHM2PDF Pilot

docking

In Windows Forms, attaching a control to the edge of the form. You can dock a control by setting the *Dock* property.

E

encapsulation

In component programming, separating the implementation of a component from the interface. Only the public interface of a component is made accessible to the rest of the application. Component data should never be accessible to outside callers.

event

A notification raised by a control or component that a specified condition has occurred.

event handler

A method that is executed when a particular event is raised.

exception

Any class that derives from *System.Exception*. Exceptions are thrown in response to run-time errors. *See also* throwing.

exception handling

The process of dealing with unusual occurrences within code so that it does not cause the program to crash or lose data.

explicit conversion

A conversion of one type to another that cannot be performed automatically. An explicit conversion usually presents some danger of a failed conversion or a loss of data.

extender provider

This document is created with the unregistered version of CHM2PDF Pilot

A specialized type of control that provides design time properties to other controls. *ErrorProvider* and *HelpProvider* are examples of extender providers.

F

field

A member variable of a class or structure.

finalizer

In Visual Basic .NET, the method that is called just before an instance of a class is destroyed. *See also* destructor.

floating-point numbers

A number that has a decimal component. *See also* floating-point types.

floating-point types

Types capable of representing floating-point numbers. Floating-point types include Single (float) and Double (double). *See also* floating-point numbers.

focus

When referring to a control, the condition of being able to receive user input through the keyboard or other input devices. A control that is capable of receiving input is said to have the focus.

foreign key

In a data table, a column that contains a key value that uniquely identifies a record in a related table.

form

The primary unit of the user interface in a Windows Forms application. A form serves as a nexus for data display and user input, and can host controls.

This document is created with the unregistered version of CHM2PDF Pilot

G

GAC

See Global Assembly Cache (GAC).

garbage collection

Automatic memory management provided by the common language runtime. Unused memory is automatically reclaimed by garbage collection without interaction with the application.

GDI+

See Graphic Device Interface (GDI).

getter

The code segment of a property that returns the value the property contains.

Global Assembly Cache (GAC)

A machine-wide cache that stores assemblies that have been specifically designated to be shared between several applications.

globalization

Applying culture-based formatting to existing data, for example, using culture-specific currency symbols and formatting.

Graphic Device Interface (GDI)

In Windows, a graphics display system used by applications to display or print bitmapped text (TrueType fonts), images, and other graphical elements. GDI+ is the name given to the .NET Framework's managed implementation of the GDI.

I

This document is created with the unregistered version of CHM2PDF Pilot

implicit conversion

A conversion between types that can be performed automatically by the common language runtime. An implicit conversion will always succeed and never pose a danger of data loss.

inheritance

In object-oriented programming, creating a new class that incorporates and extends all of the characteristics and functionality of a base class.

inherited control

A user-created control that inherits an existing control. An inherited control inherits all of the functionality of the base control and can incorporate custom functionality as well.

interface

A contract for behavior that can be implemented by a class. Classes that implement the same interface can be expected to provide the members defined by the interface.

L

localization

Providing multiple sets of data specific to different cultures and retrieving the data appropriate to the current culture setting.

logical error

An error that compiles and executes correctly but produces unexpected results. These can be the most difficult errors to detect in application testing.

M

This document is created with the unregistered version of CHM2PDF Pilot

managed code

Code that runs under the common language runtime. The common language runtime handles many tasks that would formerly have been handled in the application's executable. Managed code solves the Windows programming problems of component registration and versioning (sometimes called DLL Hell) because managed code contains all the versioning and type information that the common language runtime needs to run the application. The common language runtime handles registration dynamically at run time rather than statically through the system registry, as is done with applications based on the Component Object Model (COM).

metadata

Data about data. For example, the name, version, security information, and size of a file constitute metadata about the file.

Microsoft Intermediate Language (MSIL)

A low-level language that is just-in-time compiled to native code at run time. All .NET assemblies are represented in the MSIL.

MSIL

See Microsoft Intermediate Language (MSIL).

multidimensional array

An array that contains more than one dimension.

N

namespace

A logical organization of types that perform related functions.

.NET Framework

This document is created with the unregistered version of CHM2PDF Pilot

A managed environment for the development and execution of code. It consists of the common language runtime and .NET Framework base class library.

.NET Framework base class library

A collection of object-oriented types and interfaces that provides object models to assist in the development of your applications.

O

object

The base class for all types in the .NET Framework. More generically, any instance of a class or structure.

object-oriented programming

The process of creating applications by using programmatic constructs to represent real-world objects.

operator overloading

In Visual C#, defining custom behavior for an operator when interacting with custom types.

optimization

The process of examining your application and modifying it to be more efficient in terms of either speed or size of the deployed application.

overloading

Creating multiple methods with the same name but different signatures.

overriding

When inheriting from a base class, providing a new method for one of the base class methods.

P

This document is created with the unregistered version of CHM2PDF Pilot

parent-child relationship

In data access, a relationship between two tables wherein a record from the parent table can be used to retrieve one or more related records from a child table. A parent-child relationship can be defined by using a *DataRelation* object.

polymorphism

The ability of classes to provide different implementations of the same public interface. Thus, two classes that exhibit polymorphism might contain different implementations, but because the interfaces are identical, they can be treated the same in code.

primary key

In a data table, the column that uniquely identifies a record in the table.

private assembly

An assembly that is used by only one application. In .NET Framework programming, most assemblies are private assemblies.

project

The collection of Visual Studio .NET source files that make up an application.

R

Reference type

A type, such as a class, whose instance data is allocated on the heap. Variables that represent Reference types contain a reference to the object on the heap as opposed to the object's data itself. *See also* Value type.

registry

The system repository for information about a computer's settings. The registry can contain configuration information

This document is created with the unregistered version of CHM2PDF Pilot

about programs and system components.

resource

Nonexecutable data, such as a string or image, that is deployed in an application.

role-based security

Code that ensures that users of your application have appropriate permission to access your application based on the identity of the user.

run time

Represents the state of the application while executing. At run time, the application consists of executing code.

run-time error

An error that causes an invalid result or attempts an invalid operation at run time.

S

satellite assembly

When localizing an application, assemblies that contain alternate sets of resources to be used in the application for different cultures.

setter

The code segment of a property that sets the property's value.

shadowing

Hiding a member in a base class with a member in an inherited class that has a different signature.

shared assembly

This document is created with the unregistered version of CHM2PDF Pilot

An assembly that is referenced by several applications on a single machine. A shared assembly must be installed to the Global Assembly Cache (GAC).

shared members

In Visual Basic .NET, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Shared members are called static members in Visual C#.

simple binding

In data binding, binding a control to a single data column. A *TextBox* is an example of a control that can be simply bound.

solution

A group of Visual Studio .NET projects that make up a single functional unit.

SQL

See Structured Query Language (SQL).

static members

In Visual C#, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Static members are called shared members in Visual Basic .NET.

Step Into

Moving from a calling procedure to a called procedure during debugging. Also used in reference to setting breakpoints to stop execution in a specific procedure during debugging.

Step Out

In Break mode, resuming execution of a function call until the call returns.

Step Over

In Break mode, executing a procedure call as a single statement during debugging.

This document is created with the unregistered version of CHM2PDF Pilot

strong name

A name that uniquely identifies an assembly. It consists of the assembly's name, version, and culture information as well as a public key encrypted by a private key. An assembly must have a strong name to be installed to the Global Assembly Cache (GAC).

strongly typed

Refers to types for all variables being explicitly declared. Strong typing eliminates the possibility of type mismatch errors, which are discovered at compile time. Visual C# is inherently strongly typed, and Visual Basic .NET is strongly typed when Option Strict is set to On.

Struct

In Visual C#, a user-defined Value type that serves as a template for an instance of that type. *See also* Structure.

Structure

In Visual Basic .NET, a user-defined Value type that serves as a template for an instance of that type. *See also* Struct .

structured exception handling

Using *Try...Catch...Finally* blocks to handle errors that might occur during code execution. When exceptions are thrown, they are handled in the *Catch* block before code in the *Finally* block is executed.

Structured Query Language (SQL)

The standard language used for issuing commands to databases.

syntax error

An error that occurs because of typographical errors or nonsensical code. In the .NET Framework, syntax errors prevent compilation and are thus discovered at compile time.

T

target audience

This document is created with the unregistered version of CHM2PDF Pilot

The group of end users for whom an application is designed.

threads

The basic unit to which the server allocates processor time. A single process can have multiple threads.

throwing

The act of passing an exception to the next caller in the call stack. *See also* exception.

tracing

A programming technique for recording events, such as exceptions, in an application. Tracing is used during debugging and in the testing phase of application deployment.

transaction

A group of commands (treated as a single unit) that change the data stored in a database. The transaction ensures that the commands are handled in an all-or-nothing fashion if one of the commands fails, all of the commands fail, and any data that was written to the database by the commands is backed out. In this way, transactions maintain the integrity of data in a database.

tuning

The process of making adjustments to a deployed application that do not affect code.

U

Unicode

The universal character encoding scheme for characters and text. Each character is represented by a unique numeric value.

unmanaged code

Code that is not managed by the common language runtime. Unmanaged code is not checked for type-safety and

This document is created with the unregistered version of CHM2PDF Pilot

must be used with extreme care.

user control

A user-defined control that encapsulates other Windows Forms controls with custom functionality in a common container.

V

Value type

A type, such as an Integer, Decimal, or Structure, whose instance data is allocated on the stack. Variables that represent Value types contain the object's instance data itself as opposed to a reference to that data. *See also* Reference type.

W

Watch window

When debugging, the window that allows you to examine the values of predetermined watch variables.

Web Service

A service or component that is implemented by a Web server and can be instantiated and accessed programmatically as a component over the World Wide Web.

X

XML Schema

A description of the data elements contained in an XML file. The XML Schema provides the names of the elements, their types, whether or not they are key fields, and other information.

This document is created with the unregistered version of CHM2PDF Pilot

Glossary

A

abstract class

A class that cannot be instantiated but is used as a base from which other classes can be derived. In Microsoft Visual Basic .NET, abstract classes are declared using the *MustInherit* keyword. In Microsoft Visual C#, abstract classes are declared using the *abstract* keyword.

abstraction

Representing real-world objects as programmatic constructs.

abstract members

A member of a base class that cannot be invoked, but instead provides a template for members of a derived class. In Visual Basic .NET, abstract members are declared using the *MustOverride* keyword. In Visual C#, abstract members are declared using the *abstract* keyword.

ADO.NET

The data access architecture for the Microsoft .NET Framework. ADO.NET is built around a disconnected data access model that uses a set of classes called a Data Provider to retrieve data from a data source.

alias

The name you use to identify resources on the Web. Aliases represent physical resources on the Web server, such as a Web form, an HTML page, or a graphic.

anchoring

In Windows Forms controls, the resizing and repositioning of a control when its parent form is resized. You can anchor a control by setting the *Anchor* property.

This document is created with the unregistered version of CHM2PDF Pilot

assembly

The primary unit of deployment for a .NET application. An assembly is either an executable application (EXE) or a class library (DLL).

assembly manifest

The metadata for the assembly. It contains all of the information needed to describe the assembly to the common language runtime.

authentication

The process of determining the identity of a user. In effect, authentication validates that the users are who they say they are. *See also* authorization.

authorization

The process of allowing access to an application or resource based on credentials supplied by the user. *See also* authentication.

B

base class

A class that provides properties and methods as a foundation for a derived class. In objected-oriented programming, one class can be based on another through inheritance. Using this technique, the base class provides characteristics (such as properties and methods) to a derived class. The derived class can reuse, modify, or add to the members of the base class.

Bezier

A curve specified by four points. Two points specify the ends of the curve, and the other two points influence the shape of the curve between the end points.

Boolean

A data type that can represent a value of either true or false.

This document is created with the unregistered version of CHM2PDF Pilot

bootstrapper

In deployment, an application that installs Microsoft Windows Installer on machines that do not currently have it installed.

boundary condition

In testing, test conditions at the limit of normal operation and just over or under those limits.

boxing

The act of wrapping a Value type, such as an Integer, inside an object so that it can be treated like a Reference type.

Break mode

In debugging, a state of suspended execution wherein the application can be executed in a line-by-line manner, and individual program variables can be examined.

breakpoint

In debugging, a predetermined point where execution of an application will halt and enter Break mode.

C

cast

An explicit conversion from one type to another.

circular reference

A condition that occurs when two objects refer to each other but do not have an external reference. Circular references are automatically detected and disposed of by the garbage collector.

class

A user-defined Reference type that serves as a template for an object of that type.

This document is created with the unregistered version of CHM2PDF Pilot

CLR

See common language runtime (CLR).

code access security

Verification that all callers to code have appropriate permission to access it.

code tuning

The act of examining application code and optimizing for faster execution. Code tuning is usually an iterative process that involves several successive rounds of planning, improvement, and assessment of the results.

common language runtime (CLR)

The environment in which managed code executes. The common language runtime provides just-in-time compilation, enforces type safety, and manages memory through garbage collection.

Common Type System

A set of types that are used by all .NET languages, thus ensuring .NET language type compatibility.

complex binding

In data binding, binding more than one column to a single control. The *DataGrid* is an example of a control that can be complexly bound.

component

Strictly speaking, a class that implements *IComponent* or inherits *System*.*Component*. More generally, any class that provides a reusable set of functionality.

connected data access

Data access that requires a data connection that is always open.

Constant

This document is created with the unregistered version of CHM2PDF Pilot

A type that represents a value that does not change in the course of an application's execution and cannot be changed.

constraint

In ADO.NET, an object that represents a rule that enforces referential integrity, such as a constraint that specifies a primary key.

constructor

The method that executes when a class is instantiated. The constructor usually contains initialization code to set member variables to initial values.

container controls

Controls that can host other controls. Examples include *Forms*, *Panels*, and *TabPages*.

control

A class that inherits *System*.*Control*. Controls usually have a visual representation, interact with the user as a part of the user interface, and are hosted in a container control.

controls collection

The collection of a container control that manages the controls it hosts.

custom control

A user-designed control that directly inherits *System*.*Control*.

D

data binding

A way to link data (such as a data set) in your application to the properties of a control.

data consumer

This document is created with the unregistered version of CHM2PDF Pilot

A control that is bound to a data source.

Data Provider

A set of classes that work together to provide managed data access.

DataSet

A class that contains an application copy of data that has been retrieved from a data source.

delegate

An object that acts as a type-safe function pointer. Delegates can be used to invoke the method they point to.

deploying

Installing an application on the computer where it will run.

derived class

A class that is based on another class (called a base class) through inheritance. A derived class inherits the members of its base class and can override or shadow those members.

design time

Represents the state of the application while being designed. Applications at design time consist of noncompiled code that can be manipulated in the code editor or designer window.

destructor

The method that is run just before a class is removed from memory. Generally, a destructor contains any code required to "clean up" after the class. *See also* finalizer.

disconnected data access

Data access that uses a data connection only when executing database operations. This is the default kind of data access implemented by ADO.NET.

This document is created with the unregistered version of CHM2PDF Pilot

docking

In Windows Forms, attaching a control to the edge of the form. You can dock a control by setting the *Dock* property.

E

encapsulation

In component programming, separating the implementation of a component from the interface. Only the public interface of a component is made accessible to the rest of the application. Component data should never be accessible to outside callers.

event

A notification raised by a control or component that a specified condition has occurred.

event handler

A method that is executed when a particular event is raised.

exception

Any class that derives from *System.Exception*. Exceptions are thrown in response to run-time errors. *See also* throwing.

exception handling

The process of dealing with unusual occurrences within code so that it does not cause the program to crash or lose data.

explicit conversion

A conversion of one type to another that cannot be performed automatically. An explicit conversion usually presents some danger of a failed conversion or a loss of data.

extender provider

This document is created with the unregistered version of CHM2PDF Pilot

A specialized type of control that provides design time properties to other controls. *ErrorProvider* and *HelpProvider* are examples of extender providers.

F

field

A member variable of a class or structure.

finalizer

In Visual Basic .NET, the method that is called just before an instance of a class is destroyed. *See also* destructor.

floating-point numbers

A number that has a decimal component. *See also* floating-point types.

floating-point types

Types capable of representing floating-point numbers. Floating-point types include Single (float) and Double (double). *See also* floating-point numbers.

focus

When referring to a control, the condition of being able to receive user input through the keyboard or other input devices. A control that is capable of receiving input is said to have the focus.

foreign key

In a data table, a column that contains a key value that uniquely identifies a record in a related table.

form

The primary unit of the user interface in a Windows Forms application. A form serves as a nexus for data display and user input, and can host controls.

This document is created with the unregistered version of CHM2PDF Pilot

G

GAC

See Global Assembly Cache (GAC).

garbage collection

Automatic memory management provided by the common language runtime. Unused memory is automatically reclaimed by garbage collection without interaction with the application.

GDI+

See Graphic Device Interface (GDI).

getter

The code segment of a property that returns the value the property contains.

Global Assembly Cache (GAC)

A machine-wide cache that stores assemblies that have been specifically designated to be shared between several applications.

globalization

Applying culture-based formatting to existing data, for example, using culture-specific currency symbols and formatting.

Graphic Device Interface (GDI)

In Windows, a graphics display system used by applications to display or print bitmapped text (TrueType fonts), images, and other graphical elements. GDI+ is the name given to the .NET Framework's managed implementation of the GDI.

I

This document is created with the unregistered version of CHM2PDF Pilot

implicit conversion

A conversion between types that can be performed automatically by the common language runtime. An implicit conversion will always succeed and never pose a danger of data loss.

inheritance

In object-oriented programming, creating a new class that incorporates and extends all of the characteristics and functionality of a base class.

inherited control

A user-created control that inherits an existing control. An inherited control inherits all of the functionality of the base control and can incorporate custom functionality as well.

interface

A contract for behavior that can be implemented by a class. Classes that implement the same interface can be expected to provide the members defined by the interface.

L

localization

Providing multiple sets of data specific to different cultures and retrieving the data appropriate to the current culture setting.

logical error

An error that compiles and executes correctly but produces unexpected results. These can be the most difficult errors to detect in application testing.

M

This document is created with the unregistered version of CHM2PDF Pilot

managed code

Code that runs under the common language runtime. The common language runtime handles many tasks that would formerly have been handled in the application's executable. Managed code solves the Windows programming problems of component registration and versioning (sometimes called DLL Hell) because managed code contains all the versioning and type information that the common language runtime needs to run the application. The common language runtime handles registration dynamically at run time rather than statically through the system registry, as is done with applications based on the Component Object Model (COM).

metadata

Data about data. For example, the name, version, security information, and size of a file constitute metadata about the file.

Microsoft Intermediate Language (MSIL)

A low-level language that is just-in-time compiled to native code at run time. All .NET assemblies are represented in the MSIL.

MSIL

See Microsoft Intermediate Language (MSIL).

multidimensional array

An array that contains more than one dimension.

N

namespace

A logical organization of types that perform related functions.

.NET Framework

This document is created with the unregistered version of CHM2PDF Pilot

A managed environment for the development and execution of code. It consists of the common language runtime and .NET Framework base class library.

.NET Framework base class library

A collection of object-oriented types and interfaces that provides object models to assist in the development of your applications.

O

object

The base class for all types in the .NET Framework. More generically, any instance of a class or structure.

object-oriented programming

The process of creating applications by using programmatic constructs to represent real-world objects.

operator overloading

In Visual C#, defining custom behavior for an operator when interacting with custom types.

optimization

The process of examining your application and modifying it to be more efficient in terms of either speed or size of the deployed application.

overloading

Creating multiple methods with the same name but different signatures.

overriding

When inheriting from a base class, providing a new method for one of the base class methods.

P

This document is created with the unregistered version of CHM2PDF Pilot

parent-child relationship

In data access, a relationship between two tables wherein a record from the parent table can be used to retrieve one or more related records from a child table. A parent-child relationship can be defined by using a *DataRelation* object.

polymorphism

The ability of classes to provide different implementations of the same public interface. Thus, two classes that exhibit polymorphism might contain different implementations, but because the interfaces are identical, they can be treated the same in code.

primary key

In a data table, the column that uniquely identifies a record in the table.

private assembly

An assembly that is used by only one application. In .NET Framework programming, most assemblies are private assemblies.

project

The collection of Visual Studio .NET source files that make up an application.

R

Reference type

A type, such as a class, whose instance data is allocated on the heap. Variables that represent Reference types contain a reference to the object on the heap as opposed to the object's data itself. *See also* Value type.

registry

The system repository for information about a computer's settings. The registry can contain configuration information

This document is created with the unregistered version of CHM2PDF Pilot

about programs and system components.

resource

Nonexecutable data, such as a string or image, that is deployed in an application.

role-based security

Code that ensures that users of your application have appropriate permission to access your application based on the identity of the user.

run time

Represents the state of the application while executing. At run time, the application consists of executing code.

run-time error

An error that causes an invalid result or attempts an invalid operation at run time.

S

satellite assembly

When localizing an application, assemblies that contain alternate sets of resources to be used in the application for different cultures.

setter

The code segment of a property that sets the property's value.

shadowing

Hiding a member in a base class with a member in an inherited class that has a different signature.

shared assembly

This document is created with the unregistered version of CHM2PDF Pilot

An assembly that is referenced by several applications on a single machine. A shared assembly must be installed to the Global Assembly Cache (GAC).

shared members

In Visual Basic .NET, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Shared members are called static members in Visual C#.

simple binding

In data binding, binding a control to a single data column. A *TextBox* is an example of a control that can be simply bound.

solution

A group of Visual Studio .NET projects that make up a single functional unit.

SQL

See Structured Query Language (SQL).

static members

In Visual C#, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Static members are called shared members in Visual Basic .NET.

Step Into

Moving from a calling procedure to a called procedure during debugging. Also used in reference to setting breakpoints to stop execution in a specific procedure during debugging.

Step Out

In Break mode, resuming execution of a function call until the call returns.

Step Over

In Break mode, executing a procedure call as a single statement during debugging.

This document is created with the unregistered version of CHM2PDF Pilot

strong name

A name that uniquely identifies an assembly. It consists of the assembly's name, version, and culture information as well as a public key encrypted by a private key. An assembly must have a strong name to be installed to the Global Assembly Cache (GAC).

strongly typed

Refers to types for all variables being explicitly declared. Strong typing eliminates the possibility of type mismatch errors, which are discovered at compile time. Visual C# is inherently strongly typed, and Visual Basic .NET is strongly typed when Option Strict is set to On.

Struct

In Visual C#, a user-defined Value type that serves as a template for an instance of that type. *See also* Structure.

Structure

In Visual Basic .NET, a user-defined Value type that serves as a template for an instance of that type. *See also* Struct .

structured exception handling

Using *Try...Catch...Finally* blocks to handle errors that might occur during code execution. When exceptions are thrown, they are handled in the *Catch* block before code in the *Finally* block is executed.

Structured Query Language (SQL)

The standard language used for issuing commands to databases.

syntax error

An error that occurs because of typographical errors or nonsensical code. In the .NET Framework, syntax errors prevent compilation and are thus discovered at compile time.

T

target audience

This document is created with the unregistered version of CHM2PDF Pilot

The group of end users for whom an application is designed.

threads

The basic unit to which the server allocates processor time. A single process can have multiple threads.

throwing

The act of passing an exception to the next caller in the call stack. *See also* exception.

tracing

A programming technique for recording events, such as exceptions, in an application. Tracing is used during debugging and in the testing phase of application deployment.

transaction

A group of commands (treated as a single unit) that change the data stored in a database. The transaction ensures that the commands are handled in an all-or-nothing fashion if one of the commands fails, all of the commands fail, and any data that was written to the database by the commands is backed out. In this way, transactions maintain the integrity of data in a database.

tuning

The process of making adjustments to a deployed application that do not affect code.

U

Unicode

The universal character encoding scheme for characters and text. Each character is represented by a unique numeric value.

unmanaged code

Code that is not managed by the common language runtime. Unmanaged code is not checked for type-safety and

This document is created with the unregistered version of CHM2PDF Pilot

must be used with extreme care.

user control

A user-defined control that encapsulates other Windows Forms controls with custom functionality in a common container.

V

Value type

A type, such as an Integer, Decimal, or Structure, whose instance data is allocated on the stack. Variables that represent Value types contain the object's instance data itself as opposed to a reference to that data. *See also* Reference type.

W

Watch window

When debugging, the window that allows you to examine the values of predetermined watch variables.

Web Service

A service or component that is implemented by a Web server and can be instantiated and accessed programmatically as a component over the World Wide Web.

X

XML Schema

A description of the data elements contained in an XML file. The XML Schema provides the names of the elements, their types, whether or not they are key fields, and other information.

This document is created with the unregistered version of CHM2PDF Pilot

Glossary

A

abstract class

A class that cannot be instantiated but is used as a base from which other classes can be derived. In Microsoft Visual Basic .NET, abstract classes are declared using the *MustInherit* keyword. In Microsoft Visual C#, abstract classes are declared using the *abstract* keyword.

abstraction

Representing real-world objects as programmatic constructs.

abstract members

A member of a base class that cannot be invoked, but instead provides a template for members of a derived class. In Visual Basic .NET, abstract members are declared using the *MustOverride* keyword. In Visual C#, abstract members are declared using the *abstract* keyword.

ADO.NET

The data access architecture for the Microsoft .NET Framework. ADO.NET is built around a disconnected data access model that uses a set of classes called a Data Provider to retrieve data from a data source.

alias

The name you use to identify resources on the Web. Aliases represent physical resources on the Web server, such as a Web form, an HTML page, or a graphic.

anchoring

In Windows Forms controls, the resizing and repositioning of a control when its parent form is resized. You can anchor a control by setting the *Anchor* property.

This document is created with the unregistered version of CHM2PDF Pilot

assembly

The primary unit of deployment for a .NET application. An assembly is either an executable application (EXE) or a class library (DLL).

assembly manifest

The metadata for the assembly. It contains all of the information needed to describe the assembly to the common language runtime.

authentication

The process of determining the identity of a user. In effect, authentication validates that the users are who they say they are. *See also* authorization.

authorization

The process of allowing access to an application or resource based on credentials supplied by the user. *See also* authentication.

B

base class

A class that provides properties and methods as a foundation for a derived class. In objected-oriented programming, one class can be based on another through inheritance. Using this technique, the base class provides characteristics (such as properties and methods) to a derived class. The derived class can reuse, modify, or add to the members of the base class.

Bezier

A curve specified by four points. Two points specify the ends of the curve, and the other two points influence the shape of the curve between the end points.

Boolean

A data type that can represent a value of either true or false.

This document is created with the unregistered version of CHM2PDF Pilot

bootstrapper

In deployment, an application that installs Microsoft Windows Installer on machines that do not currently have it installed.

boundary condition

In testing, test conditions at the limit of normal operation and just over or under those limits.

boxing

The act of wrapping a Value type, such as an Integer, inside an object so that it can be treated like a Reference type.

Break mode

In debugging, a state of suspended execution wherein the application can be executed in a line-by-line manner, and individual program variables can be examined.

breakpoint

In debugging, a predetermined point where execution of an application will halt and enter Break mode.

C

cast

An explicit conversion from one type to another.

circular reference

A condition that occurs when two objects refer to each other but do not have an external reference. Circular references are automatically detected and disposed of by the garbage collector.

class

A user-defined Reference type that serves as a template for an object of that type.

This document is created with the unregistered version of CHM2PDF Pilot

CLR

See common language runtime (CLR).

code access security

Verification that all callers to code have appropriate permission to access it.

code tuning

The act of examining application code and optimizing for faster execution. Code tuning is usually an iterative process that involves several successive rounds of planning, improvement, and assessment of the results.

common language runtime (CLR)

The environment in which managed code executes. The common language runtime provides just-in-time compilation, enforces type safety, and manages memory through garbage collection.

Common Type System

A set of types that are used by all .NET languages, thus ensuring .NET language type compatibility.

complex binding

In data binding, binding more than one column to a single control. The *DataGrid* is an example of a control that can be complexly bound.

component

Strictly speaking, a class that implements *IComponent* or inherits *System*.*Component*. More generally, any class that provides a reusable set of functionality.

connected data access

Data access that requires a data connection that is always open.

Constant

This document is created with the unregistered version of CHM2PDF Pilot

A type that represents a value that does not change in the course of an application's execution and cannot be changed.

constraint

In ADO.NET, an object that represents a rule that enforces referential integrity, such as a constraint that specifies a primary key.

constructor

The method that executes when a class is instantiated. The constructor usually contains initialization code to set member variables to initial values.

container controls

Controls that can host other controls. Examples include *Forms*, *Panels*, and *TabPages*.

control

A class that inherits *System*.*Control*. Controls usually have a visual representation, interact with the user as a part of the user interface, and are hosted in a container control.

controls collection

The collection of a container control that manages the controls it hosts.

custom control

A user-designed control that directly inherits *System*.*Control*.

D

data binding

A way to link data (such as a data set) in your application to the properties of a control.

data consumer

This document is created with the unregistered version of CHM2PDF Pilot

A control that is bound to a data source.

Data Provider

A set of classes that work together to provide managed data access.

DataSet

A class that contains an application copy of data that has been retrieved from a data source.

delegate

An object that acts as a type-safe function pointer. Delegates can be used to invoke the method they point to.

deploying

Installing an application on the computer where it will run.

derived class

A class that is based on another class (called a base class) through inheritance. A derived class inherits the members of its base class and can override or shadow those members.

design time

Represents the state of the application while being designed. Applications at design time consist of noncompiled code that can be manipulated in the code editor or designer window.

destructor

The method that is run just before a class is removed from memory. Generally, a destructor contains any code required to "clean up" after the class. *See also* finalizer.

disconnected data access

Data access that uses a data connection only when executing database operations. This is the default kind of data access implemented by ADO.NET.

This document is created with the unregistered version of CHM2PDF Pilot

docking

In Windows Forms, attaching a control to the edge of the form. You can dock a control by setting the *Dock* property.

E

encapsulation

In component programming, separating the implementation of a component from the interface. Only the public interface of a component is made accessible to the rest of the application. Component data should never be accessible to outside callers.

event

A notification raised by a control or component that a specified condition has occurred.

event handler

A method that is executed when a particular event is raised.

exception

Any class that derives from *System.Exception*. Exceptions are thrown in response to run-time errors. *See also* throwing.

exception handling

The process of dealing with unusual occurrences within code so that it does not cause the program to crash or lose data.

explicit conversion

A conversion of one type to another that cannot be performed automatically. An explicit conversion usually presents some danger of a failed conversion or a loss of data.

extender provider

This document is created with the unregistered version of CHM2PDF Pilot

A specialized type of control that provides design time properties to other controls. *ErrorProvider* and *HelpProvider* are examples of extender providers.

F

field

A member variable of a class or structure.

finalizer

In Visual Basic .NET, the method that is called just before an instance of a class is destroyed. *See also* destructor.

floating-point numbers

A number that has a decimal component. *See also* floating-point types.

floating-point types

Types capable of representing floating-point numbers. Floating-point types include Single (float) and Double (double). *See also* floating-point numbers.

focus

When referring to a control, the condition of being able to receive user input through the keyboard or other input devices. A control that is capable of receiving input is said to have the focus.

foreign key

In a data table, a column that contains a key value that uniquely identifies a record in a related table.

form

The primary unit of the user interface in a Windows Forms application. A form serves as a nexus for data display and user input, and can host controls.

This document is created with the unregistered version of CHM2PDF Pilot

G

GAC

See Global Assembly Cache (GAC).

garbage collection

Automatic memory management provided by the common language runtime. Unused memory is automatically reclaimed by garbage collection without interaction with the application.

GDI+

See Graphic Device Interface (GDI).

getter

The code segment of a property that returns the value the property contains.

Global Assembly Cache (GAC)

A machine-wide cache that stores assemblies that have been specifically designated to be shared between several applications.

globalization

Applying culture-based formatting to existing data, for example, using culture-specific currency symbols and formatting.

Graphic Device Interface (GDI)

In Windows, a graphics display system used by applications to display or print bitmapped text (TrueType fonts), images, and other graphical elements. GDI+ is the name given to the .NET Framework's managed implementation of the GDI.

I

This document is created with the unregistered version of CHM2PDF Pilot

implicit conversion

A conversion between types that can be performed automatically by the common language runtime. An implicit conversion will always succeed and never pose a danger of data loss.

inheritance

In object-oriented programming, creating a new class that incorporates and extends all of the characteristics and functionality of a base class.

inherited control

A user-created control that inherits an existing control. An inherited control inherits all of the functionality of the base control and can incorporate custom functionality as well.

interface

A contract for behavior that can be implemented by a class. Classes that implement the same interface can be expected to provide the members defined by the interface.

L

localization

Providing multiple sets of data specific to different cultures and retrieving the data appropriate to the current culture setting.

logical error

An error that compiles and executes correctly but produces unexpected results. These can be the most difficult errors to detect in application testing.

M

This document is created with the unregistered version of CHM2PDF Pilot

managed code

Code that runs under the common language runtime. The common language runtime handles many tasks that would formerly have been handled in the application's executable. Managed code solves the Windows programming problems of component registration and versioning (sometimes called DLL Hell) because managed code contains all the versioning and type information that the common language runtime needs to run the application. The common language runtime handles registration dynamically at run time rather than statically through the system registry, as is done with applications based on the Component Object Model (COM).

metadata

Data about data. For example, the name, version, security information, and size of a file constitute metadata about the file.

Microsoft Intermediate Language (MSIL)

A low-level language that is just-in-time compiled to native code at run time. All .NET assemblies are represented in the MSIL.

MSIL

See Microsoft Intermediate Language (MSIL).

multidimensional array

An array that contains more than one dimension.

N

namespace

A logical organization of types that perform related functions.

.NET Framework

This document is created with the unregistered version of CHM2PDF Pilot

A managed environment for the development and execution of code. It consists of the common language runtime and .NET Framework base class library.

.NET Framework base class library

A collection of object-oriented types and interfaces that provides object models to assist in the development of your applications.

O

object

The base class for all types in the .NET Framework. More generically, any instance of a class or structure.

object-oriented programming

The process of creating applications by using programmatic constructs to represent real-world objects.

operator overloading

In Visual C#, defining custom behavior for an operator when interacting with custom types.

optimization

The process of examining your application and modifying it to be more efficient in terms of either speed or size of the deployed application.

overloading

Creating multiple methods with the same name but different signatures.

overriding

When inheriting from a base class, providing a new method for one of the base class methods.

P

This document is created with the unregistered version of CHM2PDF Pilot

parent-child relationship

In data access, a relationship between two tables wherein a record from the parent table can be used to retrieve one or more related records from a child table. A parent-child relationship can be defined by using a *DataRelation* object.

polymorphism

The ability of classes to provide different implementations of the same public interface. Thus, two classes that exhibit polymorphism might contain different implementations, but because the interfaces are identical, they can be treated the same in code.

primary key

In a data table, the column that uniquely identifies a record in the table.

private assembly

An assembly that is used by only one application. In .NET Framework programming, most assemblies are private assemblies.

project

The collection of Visual Studio .NET source files that make up an application.

R

Reference type

A type, such as a class, whose instance data is allocated on the heap. Variables that represent Reference types contain a reference to the object on the heap as opposed to the object's data itself. *See also* Value type.

registry

The system repository for information about a computer's settings. The registry can contain configuration information

This document is created with the unregistered version of CHM2PDF Pilot

about programs and system components.

resource

Nonexecutable data, such as a string or image, that is deployed in an application.

role-based security

Code that ensures that users of your application have appropriate permission to access your application based on the identity of the user.

run time

Represents the state of the application while executing. At run time, the application consists of executing code.

run-time error

An error that causes an invalid result or attempts an invalid operation at run time.

S

satellite assembly

When localizing an application, assemblies that contain alternate sets of resources to be used in the application for different cultures.

setter

The code segment of a property that sets the property's value.

shadowing

Hiding a member in a base class with a member in an inherited class that has a different signature.

shared assembly

This document is created with the unregistered version of CHM2PDF Pilot

An assembly that is referenced by several applications on a single machine. A shared assembly must be installed to the Global Assembly Cache (GAC).

shared members

In Visual Basic .NET, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Shared members are called static members in Visual C#.

simple binding

In data binding, binding a control to a single data column. A *TextBox* is an example of a control that can be simply bound.

solution

A group of Visual Studio .NET projects that make up a single functional unit.

SQL

See Structured Query Language (SQL).

static members

In Visual C#, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Static members are called shared members in Visual Basic .NET.

Step Into

Moving from a calling procedure to a called procedure during debugging. Also used in reference to setting breakpoints to stop execution in a specific procedure during debugging.

Step Out

In Break mode, resuming execution of a function call until the call returns.

Step Over

In Break mode, executing a procedure call as a single statement during debugging.

This document is created with the unregistered version of CHM2PDF Pilot

strong name

A name that uniquely identifies an assembly. It consists of the assembly's name, version, and culture information as well as a public key encrypted by a private key. An assembly must have a strong name to be installed to the Global Assembly Cache (GAC).

strongly typed

Refers to types for all variables being explicitly declared. Strong typing eliminates the possibility of type mismatch errors, which are discovered at compile time. Visual C# is inherently strongly typed, and Visual Basic .NET is strongly typed when Option Strict is set to On.

Struct

In Visual C#, a user-defined Value type that serves as a template for an instance of that type. *See also* Structure.

Structure

In Visual Basic .NET, a user-defined Value type that serves as a template for an instance of that type. *See also* Struct .

structured exception handling

Using *Try...Catch...Finally* blocks to handle errors that might occur during code execution. When exceptions are thrown, they are handled in the *Catch* block before code in the *Finally* block is executed.

Structured Query Language (SQL)

The standard language used for issuing commands to databases.

syntax error

An error that occurs because of typographical errors or nonsensical code. In the .NET Framework, syntax errors prevent compilation and are thus discovered at compile time.

T

target audience

This document is created with the unregistered version of CHM2PDF Pilot

The group of end users for whom an application is designed.

threads

The basic unit to which the server allocates processor time. A single process can have multiple threads.

throwing

The act of passing an exception to the next caller in the call stack. *See also* exception.

tracing

A programming technique for recording events, such as exceptions, in an application. Tracing is used during debugging and in the testing phase of application deployment.

transaction

A group of commands (treated as a single unit) that change the data stored in a database. The transaction ensures that the commands are handled in an all-or-nothing fashion if one of the commands fails, all of the commands fail, and any data that was written to the database by the commands is backed out. In this way, transactions maintain the integrity of data in a database.

tuning

The process of making adjustments to a deployed application that do not affect code.

U

Unicode

The universal character encoding scheme for characters and text. Each character is represented by a unique numeric value.

unmanaged code

Code that is not managed by the common language runtime. Unmanaged code is not checked for type-safety and

This document is created with the unregistered version of CHM2PDF Pilot

must be used with extreme care.

user control

A user-defined control that encapsulates other Windows Forms controls with custom functionality in a common container.

V

Value type

A type, such as an Integer, Decimal, or Structure, whose instance data is allocated on the stack. Variables that represent Value types contain the object's instance data itself as opposed to a reference to that data. *See also* Reference type.

W

Watch window

When debugging, the window that allows you to examine the values of predetermined watch variables.

Web Service

A service or component that is implemented by a Web server and can be instantiated and accessed programmatically as a component over the World Wide Web.

X

XML Schema

A description of the data elements contained in an XML file. The XML Schema provides the names of the elements, their types, whether or not they are key fields, and other information.

This document is created with the unregistered version of CHM2PDF Pilot

Glossary

A

abstract class

A class that cannot be instantiated but is used as a base from which other classes can be derived. In Microsoft Visual Basic .NET, abstract classes are declared using the *MustInherit* keyword. In Microsoft Visual C#, abstract classes are declared using the *abstract* keyword.

abstraction

Representing real-world objects as programmatic constructs.

abstract members

A member of a base class that cannot be invoked, but instead provides a template for members of a derived class. In Visual Basic .NET, abstract members are declared using the *MustOverride* keyword. In Visual C#, abstract members are declared using the *abstract* keyword.

ADO.NET

The data access architecture for the Microsoft .NET Framework. ADO.NET is built around a disconnected data access model that uses a set of classes called a Data Provider to retrieve data from a data source.

alias

The name you use to identify resources on the Web. Aliases represent physical resources on the Web server, such as a Web form, an HTML page, or a graphic.

anchoring

In Windows Forms controls, the resizing and repositioning of a control when its parent form is resized. You can anchor a control by setting the *Anchor* property.

This document is created with the unregistered version of CHM2PDF Pilot

assembly

The primary unit of deployment for a .NET application. An assembly is either an executable application (EXE) or a class library (DLL).

assembly manifest

The metadata for the assembly. It contains all of the information needed to describe the assembly to the common language runtime.

authentication

The process of determining the identity of a user. In effect, authentication validates that the users are who they say they are. *See also* authorization.

authorization

The process of allowing access to an application or resource based on credentials supplied by the user. *See also* authentication.

B

base class

A class that provides properties and methods as a foundation for a derived class. In objected-oriented programming, one class can be based on another through inheritance. Using this technique, the base class provides characteristics (such as properties and methods) to a derived class. The derived class can reuse, modify, or add to the members of the base class.

Bezier

A curve specified by four points. Two points specify the ends of the curve, and the other two points influence the shape of the curve between the end points.

Boolean

A data type that can represent a value of either true or false.

This document is created with the unregistered version of CHM2PDF Pilot

bootstrapper

In deployment, an application that installs Microsoft Windows Installer on machines that do not currently have it installed.

boundary condition

In testing, test conditions at the limit of normal operation and just over or under those limits.

boxing

The act of wrapping a Value type, such as an Integer, inside an object so that it can be treated like a Reference type.

Break mode

In debugging, a state of suspended execution wherein the application can be executed in a line-by-line manner, and individual program variables can be examined.

breakpoint

In debugging, a predetermined point where execution of an application will halt and enter Break mode.

C

cast

An explicit conversion from one type to another.

circular reference

A condition that occurs when two objects refer to each other but do not have an external reference. Circular references are automatically detected and disposed of by the garbage collector.

class

A user-defined Reference type that serves as a template for an object of that type.

This document is created with the unregistered version of CHM2PDF Pilot

CLR

See common language runtime (CLR).

code access security

Verification that all callers to code have appropriate permission to access it.

code tuning

The act of examining application code and optimizing for faster execution. Code tuning is usually an iterative process that involves several successive rounds of planning, improvement, and assessment of the results.

common language runtime (CLR)

The environment in which managed code executes. The common language runtime provides just-in-time compilation, enforces type safety, and manages memory through garbage collection.

Common Type System

A set of types that are used by all .NET languages, thus ensuring .NET language type compatibility.

complex binding

In data binding, binding more than one column to a single control. The *DataGrid* is an example of a control that can be complexly bound.

component

Strictly speaking, a class that implements *IComponent* or inherits *System*.*Component*. More generally, any class that provides a reusable set of functionality.

connected data access

Data access that requires a data connection that is always open.

Constant

This document is created with the unregistered version of CHM2PDF Pilot

A type that represents a value that does not change in the course of an application's execution and cannot be changed.

constraint

In ADO.NET, an object that represents a rule that enforces referential integrity, such as a constraint that specifies a primary key.

constructor

The method that executes when a class is instantiated. The constructor usually contains initialization code to set member variables to initial values.

container controls

Controls that can host other controls. Examples include *Forms*, *Panels*, and *TabPages*.

control

A class that inherits *System*.*Control*. Controls usually have a visual representation, interact with the user as a part of the user interface, and are hosted in a container control.

controls collection

The collection of a container control that manages the controls it hosts.

custom control

A user-designed control that directly inherits *System*.*Control*.

D

data binding

A way to link data (such as a data set) in your application to the properties of a control.

data consumer

This document is created with the unregistered version of CHM2PDF Pilot

A control that is bound to a data source.

Data Provider

A set of classes that work together to provide managed data access.

DataSet

A class that contains an application copy of data that has been retrieved from a data source.

delegate

An object that acts as a type-safe function pointer. Delegates can be used to invoke the method they point to.

deploying

Installing an application on the computer where it will run.

derived class

A class that is based on another class (called a base class) through inheritance. A derived class inherits the members of its base class and can override or shadow those members.

design time

Represents the state of the application while being designed. Applications at design time consist of noncompiled code that can be manipulated in the code editor or designer window.

destructor

The method that is run just before a class is removed from memory. Generally, a destructor contains any code required to "clean up" after the class. *See also* finalizer.

disconnected data access

Data access that uses a data connection only when executing database operations. This is the default kind of data access implemented by ADO.NET.

This document is created with the unregistered version of CHM2PDF Pilot

docking

In Windows Forms, attaching a control to the edge of the form. You can dock a control by setting the *Dock* property.

E

encapsulation

In component programming, separating the implementation of a component from the interface. Only the public interface of a component is made accessible to the rest of the application. Component data should never be accessible to outside callers.

event

A notification raised by a control or component that a specified condition has occurred.

event handler

A method that is executed when a particular event is raised.

exception

Any class that derives from *System.Exception*. Exceptions are thrown in response to run-time errors. *See also* throwing.

exception handling

The process of dealing with unusual occurrences within code so that it does not cause the program to crash or lose data.

explicit conversion

A conversion of one type to another that cannot be performed automatically. An explicit conversion usually presents some danger of a failed conversion or a loss of data.

extender provider

This document is created with the unregistered version of CHM2PDF Pilot

A specialized type of control that provides design time properties to other controls. *ErrorProvider* and *HelpProvider* are examples of extender providers.

F

field

A member variable of a class or structure.

finalizer

In Visual Basic .NET, the method that is called just before an instance of a class is destroyed. *See also* destructor.

floating-point numbers

A number that has a decimal component. *See also* floating-point types.

floating-point types

Types capable of representing floating-point numbers. Floating-point types include Single (float) and Double (double). *See also* floating-point numbers.

focus

When referring to a control, the condition of being able to receive user input through the keyboard or other input devices. A control that is capable of receiving input is said to have the focus.

foreign key

In a data table, a column that contains a key value that uniquely identifies a record in a related table.

form

The primary unit of the user interface in a Windows Forms application. A form serves as a nexus for data display and user input, and can host controls.

This document is created with the unregistered version of CHM2PDF Pilot

G

GAC

See Global Assembly Cache (GAC).

garbage collection

Automatic memory management provided by the common language runtime. Unused memory is automatically reclaimed by garbage collection without interaction with the application.

GDI+

See Graphic Device Interface (GDI).

getter

The code segment of a property that returns the value the property contains.

Global Assembly Cache (GAC)

A machine-wide cache that stores assemblies that have been specifically designated to be shared between several applications.

globalization

Applying culture-based formatting to existing data, for example, using culture-specific currency symbols and formatting.

Graphic Device Interface (GDI)

In Windows, a graphics display system used by applications to display or print bitmapped text (TrueType fonts), images, and other graphical elements. GDI+ is the name given to the .NET Framework's managed implementation of the GDI.

I

This document is created with the unregistered version of CHM2PDF Pilot

implicit conversion

A conversion between types that can be performed automatically by the common language runtime. An implicit conversion will always succeed and never pose a danger of data loss.

inheritance

In object-oriented programming, creating a new class that incorporates and extends all of the characteristics and functionality of a base class.

inherited control

A user-created control that inherits an existing control. An inherited control inherits all of the functionality of the base control and can incorporate custom functionality as well.

interface

A contract for behavior that can be implemented by a class. Classes that implement the same interface can be expected to provide the members defined by the interface.

L

localization

Providing multiple sets of data specific to different cultures and retrieving the data appropriate to the current culture setting.

logical error

An error that compiles and executes correctly but produces unexpected results. These can be the most difficult errors to detect in application testing.

M

This document is created with the unregistered version of CHM2PDF Pilot

managed code

Code that runs under the common language runtime. The common language runtime handles many tasks that would formerly have been handled in the application's executable. Managed code solves the Windows programming problems of component registration and versioning (sometimes called DLL Hell) because managed code contains all the versioning and type information that the common language runtime needs to run the application. The common language runtime handles registration dynamically at run time rather than statically through the system registry, as is done with applications based on the Component Object Model (COM).

metadata

Data about data. For example, the name, version, security information, and size of a file constitute metadata about the file.

Microsoft Intermediate Language (MSIL)

A low-level language that is just-in-time compiled to native code at run time. All .NET assemblies are represented in the MSIL.

MSIL

See Microsoft Intermediate Language (MSIL).

multidimensional array

An array that contains more than one dimension.

N

namespace

A logical organization of types that perform related functions.

.NET Framework

This document is created with the unregistered version of CHM2PDF Pilot

A managed environment for the development and execution of code. It consists of the common language runtime and .NET Framework base class library.

.NET Framework base class library

A collection of object-oriented types and interfaces that provides object models to assist in the development of your applications.

O

object

The base class for all types in the .NET Framework. More generically, any instance of a class or structure.

object-oriented programming

The process of creating applications by using programmatic constructs to represent real-world objects.

operator overloading

In Visual C#, defining custom behavior for an operator when interacting with custom types.

optimization

The process of examining your application and modifying it to be more efficient in terms of either speed or size of the deployed application.

overloading

Creating multiple methods with the same name but different signatures.

overriding

When inheriting from a base class, providing a new method for one of the base class methods.

P

This document is created with the unregistered version of CHM2PDF Pilot

parent-child relationship

In data access, a relationship between two tables wherein a record from the parent table can be used to retrieve one or more related records from a child table. A parent-child relationship can be defined by using a *DataRelation* object.

polymorphism

The ability of classes to provide different implementations of the same public interface. Thus, two classes that exhibit polymorphism might contain different implementations, but because the interfaces are identical, they can be treated the same in code.

primary key

In a data table, the column that uniquely identifies a record in the table.

private assembly

An assembly that is used by only one application. In .NET Framework programming, most assemblies are private assemblies.

project

The collection of Visual Studio .NET source files that make up an application.

R

Reference type

A type, such as a class, whose instance data is allocated on the heap. Variables that represent Reference types contain a reference to the object on the heap as opposed to the object's data itself. *See also* Value type.

registry

The system repository for information about a computer's settings. The registry can contain configuration information

This document is created with the unregistered version of CHM2PDF Pilot

about programs and system components.

resource

Nonexecutable data, such as a string or image, that is deployed in an application.

role-based security

Code that ensures that users of your application have appropriate permission to access your application based on the identity of the user.

run time

Represents the state of the application while executing. At run time, the application consists of executing code.

run-time error

An error that causes an invalid result or attempts an invalid operation at run time.

S

satellite assembly

When localizing an application, assemblies that contain alternate sets of resources to be used in the application for different cultures.

setter

The code segment of a property that sets the property's value.

shadowing

Hiding a member in a base class with a member in an inherited class that has a different signature.

shared assembly

This document is created with the unregistered version of CHM2PDF Pilot

An assembly that is referenced by several applications on a single machine. A shared assembly must be installed to the Global Assembly Cache (GAC).

shared members

In Visual Basic .NET, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Shared members are called static members in Visual C#.

simple binding

In data binding, binding a control to a single data column. A *TextBox* is an example of a control that can be simply bound.

solution

A group of Visual Studio .NET projects that make up a single functional unit.

SQL

See Structured Query Language (SQL).

static members

In Visual C#, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Static members are called shared members in Visual Basic .NET.

Step Into

Moving from a calling procedure to a called procedure during debugging. Also used in reference to setting breakpoints to stop execution in a specific procedure during debugging.

Step Out

In Break mode, resuming execution of a function call until the call returns.

Step Over

In Break mode, executing a procedure call as a single statement during debugging.

This document is created with the unregistered version of CHM2PDF Pilot

strong name

A name that uniquely identifies an assembly. It consists of the assembly's name, version, and culture information as well as a public key encrypted by a private key. An assembly must have a strong name to be installed to the Global Assembly Cache (GAC).

strongly typed

Refers to types for all variables being explicitly declared. Strong typing eliminates the possibility of type mismatch errors, which are discovered at compile time. Visual C# is inherently strongly typed, and Visual Basic .NET is strongly typed when Option Strict is set to On.

Struct

In Visual C#, a user-defined Value type that serves as a template for an instance of that type. *See also* Structure.

Structure

In Visual Basic .NET, a user-defined Value type that serves as a template for an instance of that type. *See also* Struct .

structured exception handling

Using *Try...Catch...Finally* blocks to handle errors that might occur during code execution. When exceptions are thrown, they are handled in the *Catch* block before code in the *Finally* block is executed.

Structured Query Language (SQL)

The standard language used for issuing commands to databases.

syntax error

An error that occurs because of typographical errors or nonsensical code. In the .NET Framework, syntax errors prevent compilation and are thus discovered at compile time.

T

target audience

This document is created with the unregistered version of CHM2PDF Pilot

The group of end users for whom an application is designed.

threads

The basic unit to which the server allocates processor time. A single process can have multiple threads.

throwing

The act of passing an exception to the next caller in the call stack. *See also* exception.

tracing

A programming technique for recording events, such as exceptions, in an application. Tracing is used during debugging and in the testing phase of application deployment.

transaction

A group of commands (treated as a single unit) that change the data stored in a database. The transaction ensures that the commands are handled in an all-or-nothing fashion if one of the commands fails, all of the commands fail, and any data that was written to the database by the commands is backed out. In this way, transactions maintain the integrity of data in a database.

tuning

The process of making adjustments to a deployed application that do not affect code.

U

Unicode

The universal character encoding scheme for characters and text. Each character is represented by a unique numeric value.

unmanaged code

Code that is not managed by the common language runtime. Unmanaged code is not checked for type-safety and

This document is created with the unregistered version of CHM2PDF Pilot

must be used with extreme care.

user control

A user-defined control that encapsulates other Windows Forms controls with custom functionality in a common container.

V

Value type

A type, such as an Integer, Decimal, or Structure, whose instance data is allocated on the stack. Variables that represent Value types contain the object's instance data itself as opposed to a reference to that data. *See also* Reference type.

W

Watch window

When debugging, the window that allows you to examine the values of predetermined watch variables.

Web Service

A service or component that is implemented by a Web server and can be instantiated and accessed programmatically as a component over the World Wide Web.

X

XML Schema

A description of the data elements contained in an XML file. The XML Schema provides the names of the elements, their types, whether or not they are key fields, and other information.

This document is created with the unregistered version of CHM2PDF Pilot

Glossary

A

abstract class

A class that cannot be instantiated but is used as a base from which other classes can be derived. In Microsoft Visual Basic .NET, abstract classes are declared using the *MustInherit* keyword. In Microsoft Visual C#, abstract classes are declared using the *abstract* keyword.

abstraction

Representing real-world objects as programmatic constructs.

abstract members

A member of a base class that cannot be invoked, but instead provides a template for members of a derived class. In Visual Basic .NET, abstract members are declared using the *MustOverride* keyword. In Visual C#, abstract members are declared using the *abstract* keyword.

ADO.NET

The data access architecture for the Microsoft .NET Framework. ADO.NET is built around a disconnected data access model that uses a set of classes called a Data Provider to retrieve data from a data source.

alias

The name you use to identify resources on the Web. Aliases represent physical resources on the Web server, such as a Web form, an HTML page, or a graphic.

anchoring

In Windows Forms controls, the resizing and repositioning of a control when its parent form is resized. You can anchor a control by setting the *Anchor* property.

This document is created with the unregistered version of CHM2PDF Pilot

assembly

The primary unit of deployment for a .NET application. An assembly is either an executable application (EXE) or a class library (DLL).

assembly manifest

The metadata for the assembly. It contains all of the information needed to describe the assembly to the common language runtime.

authentication

The process of determining the identity of a user. In effect, authentication validates that the users are who they say they are. *See also* authorization.

authorization

The process of allowing access to an application or resource based on credentials supplied by the user. *See also* authentication.

B

base class

A class that provides properties and methods as a foundation for a derived class. In objected-oriented programming, one class can be based on another through inheritance. Using this technique, the base class provides characteristics (such as properties and methods) to a derived class. The derived class can reuse, modify, or add to the members of the base class.

Bezier

A curve specified by four points. Two points specify the ends of the curve, and the other two points influence the shape of the curve between the end points.

Boolean

A data type that can represent a value of either true or false.

This document is created with the unregistered version of CHM2PDF Pilot

bootstrapper

In deployment, an application that installs Microsoft Windows Installer on machines that do not currently have it installed.

boundary condition

In testing, test conditions at the limit of normal operation and just over or under those limits.

boxing

The act of wrapping a Value type, such as an Integer, inside an object so that it can be treated like a Reference type.

Break mode

In debugging, a state of suspended execution wherein the application can be executed in a line-by-line manner, and individual program variables can be examined.

breakpoint

In debugging, a predetermined point where execution of an application will halt and enter Break mode.

C

cast

An explicit conversion from one type to another.

circular reference

A condition that occurs when two objects refer to each other but do not have an external reference. Circular references are automatically detected and disposed of by the garbage collector.

class

A user-defined Reference type that serves as a template for an object of that type.

This document is created with the unregistered version of CHM2PDF Pilot

CLR

See common language runtime (CLR).

code access security

Verification that all callers to code have appropriate permission to access it.

code tuning

The act of examining application code and optimizing for faster execution. Code tuning is usually an iterative process that involves several successive rounds of planning, improvement, and assessment of the results.

common language runtime (CLR)

The environment in which managed code executes. The common language runtime provides just-in-time compilation, enforces type safety, and manages memory through garbage collection.

Common Type System

A set of types that are used by all .NET languages, thus ensuring .NET language type compatibility.

complex binding

In data binding, binding more than one column to a single control. The *DataGrid* is an example of a control that can be complexly bound.

component

Strictly speaking, a class that implements *IComponent* or inherits *System*.*Component*. More generally, any class that provides a reusable set of functionality.

connected data access

Data access that requires a data connection that is always open.

Constant

This document is created with the unregistered version of CHM2PDF Pilot

A type that represents a value that does not change in the course of an application's execution and cannot be changed.

constraint

In ADO.NET, an object that represents a rule that enforces referential integrity, such as a constraint that specifies a primary key.

constructor

The method that executes when a class is instantiated. The constructor usually contains initialization code to set member variables to initial values.

container controls

Controls that can host other controls. Examples include *Forms*, *Panels*, and *TabPages*.

control

A class that inherits *System*.*Control*. Controls usually have a visual representation, interact with the user as a part of the user interface, and are hosted in a container control.

controls collection

The collection of a container control that manages the controls it hosts.

custom control

A user-designed control that directly inherits *System*.*Control*.

D

data binding

A way to link data (such as a data set) in your application to the properties of a control.

data consumer

This document is created with the unregistered version of CHM2PDF Pilot

A control that is bound to a data source.

Data Provider

A set of classes that work together to provide managed data access.

DataSet

A class that contains an application copy of data that has been retrieved from a data source.

delegate

An object that acts as a type-safe function pointer. Delegates can be used to invoke the method they point to.

deploying

Installing an application on the computer where it will run.

derived class

A class that is based on another class (called a base class) through inheritance. A derived class inherits the members of its base class and can override or shadow those members.

design time

Represents the state of the application while being designed. Applications at design time consist of noncompiled code that can be manipulated in the code editor or designer window.

destructor

The method that is run just before a class is removed from memory. Generally, a destructor contains any code required to "clean up" after the class. *See also* finalizer.

disconnected data access

Data access that uses a data connection only when executing database operations. This is the default kind of data access implemented by ADO.NET.

This document is created with the unregistered version of CHM2PDF Pilot

docking

In Windows Forms, attaching a control to the edge of the form. You can dock a control by setting the *Dock* property.

E

encapsulation

In component programming, separating the implementation of a component from the interface. Only the public interface of a component is made accessible to the rest of the application. Component data should never be accessible to outside callers.

event

A notification raised by a control or component that a specified condition has occurred.

event handler

A method that is executed when a particular event is raised.

exception

Any class that derives from *System.Exception*. Exceptions are thrown in response to run-time errors. *See also* throwing.

exception handling

The process of dealing with unusual occurrences within code so that it does not cause the program to crash or lose data.

explicit conversion

A conversion of one type to another that cannot be performed automatically. An explicit conversion usually presents some danger of a failed conversion or a loss of data.

extender provider

This document is created with the unregistered version of CHM2PDF Pilot

A specialized type of control that provides design time properties to other controls. *ErrorProvider* and *HelpProvider* are examples of extender providers.

F

field

A member variable of a class or structure.

finalizer

In Visual Basic .NET, the method that is called just before an instance of a class is destroyed. *See also* destructor.

floating-point numbers

A number that has a decimal component. *See also* floating-point types.

floating-point types

Types capable of representing floating-point numbers. Floating-point types include Single (float) and Double (double). *See also* floating-point numbers.

focus

When referring to a control, the condition of being able to receive user input through the keyboard or other input devices. A control that is capable of receiving input is said to have the focus.

foreign key

In a data table, a column that contains a key value that uniquely identifies a record in a related table.

form

The primary unit of the user interface in a Windows Forms application. A form serves as a nexus for data display and user input, and can host controls.

This document is created with the unregistered version of CHM2PDF Pilot

G

GAC

See Global Assembly Cache (GAC).

garbage collection

Automatic memory management provided by the common language runtime. Unused memory is automatically reclaimed by garbage collection without interaction with the application.

GDI+

See Graphic Device Interface (GDI).

getter

The code segment of a property that returns the value the property contains.

Global Assembly Cache (GAC)

A machine-wide cache that stores assemblies that have been specifically designated to be shared between several applications.

globalization

Applying culture-based formatting to existing data, for example, using culture-specific currency symbols and formatting.

Graphic Device Interface (GDI)

In Windows, a graphics display system used by applications to display or print bitmapped text (TrueType fonts), images, and other graphical elements. GDI+ is the name given to the .NET Framework's managed implementation of the GDI.

I

This document is created with the unregistered version of CHM2PDF Pilot

implicit conversion

A conversion between types that can be performed automatically by the common language runtime. An implicit conversion will always succeed and never pose a danger of data loss.

inheritance

In object-oriented programming, creating a new class that incorporates and extends all of the characteristics and functionality of a base class.

inherited control

A user-created control that inherits an existing control. An inherited control inherits all of the functionality of the base control and can incorporate custom functionality as well.

interface

A contract for behavior that can be implemented by a class. Classes that implement the same interface can be expected to provide the members defined by the interface.

L

localization

Providing multiple sets of data specific to different cultures and retrieving the data appropriate to the current culture setting.

logical error

An error that compiles and executes correctly but produces unexpected results. These can be the most difficult errors to detect in application testing.

M

This document is created with the unregistered version of CHM2PDF Pilot

managed code

Code that runs under the common language runtime. The common language runtime handles many tasks that would formerly have been handled in the application's executable. Managed code solves the Windows programming problems of component registration and versioning (sometimes called DLL Hell) because managed code contains all the versioning and type information that the common language runtime needs to run the application. The common language runtime handles registration dynamically at run time rather than statically through the system registry, as is done with applications based on the Component Object Model (COM).

metadata

Data about data. For example, the name, version, security information, and size of a file constitute metadata about the file.

Microsoft Intermediate Language (MSIL)

A low-level language that is just-in-time compiled to native code at run time. All .NET assemblies are represented in the MSIL.

MSIL

See Microsoft Intermediate Language (MSIL).

multidimensional array

An array that contains more than one dimension.

N

namespace

A logical organization of types that perform related functions.

.NET Framework

This document is created with the unregistered version of CHM2PDF Pilot

A managed environment for the development and execution of code. It consists of the common language runtime and .NET Framework base class library.

.NET Framework base class library

A collection of object-oriented types and interfaces that provides object models to assist in the development of your applications.

O

object

The base class for all types in the .NET Framework. More generically, any instance of a class or structure.

object-oriented programming

The process of creating applications by using programmatic constructs to represent real-world objects.

operator overloading

In Visual C#, defining custom behavior for an operator when interacting with custom types.

optimization

The process of examining your application and modifying it to be more efficient in terms of either speed or size of the deployed application.

overloading

Creating multiple methods with the same name but different signatures.

overriding

When inheriting from a base class, providing a new method for one of the base class methods.

P

This document is created with the unregistered version of CHM2PDF Pilot

parent-child relationship

In data access, a relationship between two tables wherein a record from the parent table can be used to retrieve one or more related records from a child table. A parent-child relationship can be defined by using a *DataRelation* object.

polymorphism

The ability of classes to provide different implementations of the same public interface. Thus, two classes that exhibit polymorphism might contain different implementations, but because the interfaces are identical, they can be treated the same in code.

primary key

In a data table, the column that uniquely identifies a record in the table.

private assembly

An assembly that is used by only one application. In .NET Framework programming, most assemblies are private assemblies.

project

The collection of Visual Studio .NET source files that make up an application.

R

Reference type

A type, such as a class, whose instance data is allocated on the heap. Variables that represent Reference types contain a reference to the object on the heap as opposed to the object's data itself. *See also* Value type.

registry

The system repository for information about a computer's settings. The registry can contain configuration information

This document is created with the unregistered version of CHM2PDF Pilot

about programs and system components.

resource

Nonexecutable data, such as a string or image, that is deployed in an application.

role-based security

Code that ensures that users of your application have appropriate permission to access your application based on the identity of the user.

run time

Represents the state of the application while executing. At run time, the application consists of executing code.

run-time error

An error that causes an invalid result or attempts an invalid operation at run time.

S

satellite assembly

When localizing an application, assemblies that contain alternate sets of resources to be used in the application for different cultures.

setter

The code segment of a property that sets the property's value.

shadowing

Hiding a member in a base class with a member in an inherited class that has a different signature.

shared assembly

This document is created with the unregistered version of CHM2PDF Pilot

An assembly that is referenced by several applications on a single machine. A shared assembly must be installed to the Global Assembly Cache (GAC).

shared members

In Visual Basic .NET, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Shared members are called static members in Visual C#.

simple binding

In data binding, binding a control to a single data column. A *TextBox* is an example of a control that can be simply bound.

solution

A group of Visual Studio .NET projects that make up a single functional unit.

SQL

See Structured Query Language (SQL).

static members

In Visual C#, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Static members are called shared members in Visual Basic .NET.

Step Into

Moving from a calling procedure to a called procedure during debugging. Also used in reference to setting breakpoints to stop execution in a specific procedure during debugging.

Step Out

In Break mode, resuming execution of a function call until the call returns.

Step Over

In Break mode, executing a procedure call as a single statement during debugging.

This document is created with the unregistered version of CHM2PDF Pilot

strong name

A name that uniquely identifies an assembly. It consists of the assembly's name, version, and culture information as well as a public key encrypted by a private key. An assembly must have a strong name to be installed to the Global Assembly Cache (GAC).

strongly typed

Refers to types for all variables being explicitly declared. Strong typing eliminates the possibility of type mismatch errors, which are discovered at compile time. Visual C# is inherently strongly typed, and Visual Basic .NET is strongly typed when Option Strict is set to On.

Struct

In Visual C#, a user-defined Value type that serves as a template for an instance of that type. *See also* Structure.

Structure

In Visual Basic .NET, a user-defined Value type that serves as a template for an instance of that type. *See also* Struct .

structured exception handling

Using *Try...Catch...Finally* blocks to handle errors that might occur during code execution. When exceptions are thrown, they are handled in the *Catch* block before code in the *Finally* block is executed.

Structured Query Language (SQL)

The standard language used for issuing commands to databases.

syntax error

An error that occurs because of typographical errors or nonsensical code. In the .NET Framework, syntax errors prevent compilation and are thus discovered at compile time.

T

target audience

This document is created with the unregistered version of CHM2PDF Pilot

The group of end users for whom an application is designed.

threads

The basic unit to which the server allocates processor time. A single process can have multiple threads.

throwing

The act of passing an exception to the next caller in the call stack. *See also* exception.

tracing

A programming technique for recording events, such as exceptions, in an application. Tracing is used during debugging and in the testing phase of application deployment.

transaction

A group of commands (treated as a single unit) that change the data stored in a database. The transaction ensures that the commands are handled in an all-or-nothing fashion if one of the commands fails, all of the commands fail, and any data that was written to the database by the commands is backed out. In this way, transactions maintain the integrity of data in a database.

tuning

The process of making adjustments to a deployed application that do not affect code.

U

Unicode

The universal character encoding scheme for characters and text. Each character is represented by a unique numeric value.

unmanaged code

Code that is not managed by the common language runtime. Unmanaged code is not checked for type-safety and

This document is created with the unregistered version of CHM2PDF Pilot

must be used with extreme care.

user control

A user-defined control that encapsulates other Windows Forms controls with custom functionality in a common container.

V

Value type

A type, such as an Integer, Decimal, or Structure, whose instance data is allocated on the stack. Variables that represent Value types contain the object's instance data itself as opposed to a reference to that data. *See also* Reference type.

W

Watch window

When debugging, the window that allows you to examine the values of predetermined watch variables.

Web Service

A service or component that is implemented by a Web server and can be instantiated and accessed programmatically as a component over the World Wide Web.

X

XML Schema

A description of the data elements contained in an XML file. The XML Schema provides the names of the elements, their types, whether or not they are key fields, and other information.

This document is created with the unregistered version of CHM2PDF Pilot

Glossary

A

abstract class

A class that cannot be instantiated but is used as a base from which other classes can be derived. In Microsoft Visual Basic .NET, abstract classes are declared using the *MustInherit* keyword. In Microsoft Visual C#, abstract classes are declared using the *abstract* keyword.

abstraction

Representing real-world objects as programmatic constructs.

abstract members

A member of a base class that cannot be invoked, but instead provides a template for members of a derived class. In Visual Basic .NET, abstract members are declared using the *MustOverride* keyword. In Visual C#, abstract members are declared using the *abstract* keyword.

ADO.NET

The data access architecture for the Microsoft .NET Framework. ADO.NET is built around a disconnected data access model that uses a set of classes called a Data Provider to retrieve data from a data source.

alias

The name you use to identify resources on the Web. Aliases represent physical resources on the Web server, such as a Web form, an HTML page, or a graphic.

anchoring

In Windows Forms controls, the resizing and repositioning of a control when its parent form is resized. You can anchor a control by setting the *Anchor* property.

This document is created with the unregistered version of CHM2PDF Pilot

assembly

The primary unit of deployment for a .NET application. An assembly is either an executable application (EXE) or a class library (DLL).

assembly manifest

The metadata for the assembly. It contains all of the information needed to describe the assembly to the common language runtime.

authentication

The process of determining the identity of a user. In effect, authentication validates that the users are who they say they are. *See also* authorization.

authorization

The process of allowing access to an application or resource based on credentials supplied by the user. *See also* authentication.

B

base class

A class that provides properties and methods as a foundation for a derived class. In objected-oriented programming, one class can be based on another through inheritance. Using this technique, the base class provides characteristics (such as properties and methods) to a derived class. The derived class can reuse, modify, or add to the members of the base class.

Bezier

A curve specified by four points. Two points specify the ends of the curve, and the other two points influence the shape of the curve between the end points.

Boolean

A data type that can represent a value of either true or false.

This document is created with the unregistered version of CHM2PDF Pilot

bootstrapper

In deployment, an application that installs Microsoft Windows Installer on machines that do not currently have it installed.

boundary condition

In testing, test conditions at the limit of normal operation and just over or under those limits.

boxing

The act of wrapping a Value type, such as an Integer, inside an object so that it can be treated like a Reference type.

Break mode

In debugging, a state of suspended execution wherein the application can be executed in a line-by-line manner, and individual program variables can be examined.

breakpoint

In debugging, a predetermined point where execution of an application will halt and enter Break mode.

C

cast

An explicit conversion from one type to another.

circular reference

A condition that occurs when two objects refer to each other but do not have an external reference. Circular references are automatically detected and disposed of by the garbage collector.

class

A user-defined Reference type that serves as a template for an object of that type.

This document is created with the unregistered version of CHM2PDF Pilot

CLR

See common language runtime (CLR).

code access security

Verification that all callers to code have appropriate permission to access it.

code tuning

The act of examining application code and optimizing for faster execution. Code tuning is usually an iterative process that involves several successive rounds of planning, improvement, and assessment of the results.

common language runtime (CLR)

The environment in which managed code executes. The common language runtime provides just-in-time compilation, enforces type safety, and manages memory through garbage collection.

Common Type System

A set of types that are used by all .NET languages, thus ensuring .NET language type compatibility.

complex binding

In data binding, binding more than one column to a single control. The *DataGrid* is an example of a control that can be complexly bound.

component

Strictly speaking, a class that implements *IComponent* or inherits *System*.*Component*. More generally, any class that provides a reusable set of functionality.

connected data access

Data access that requires a data connection that is always open.

Constant

This document is created with the unregistered version of CHM2PDF Pilot

A type that represents a value that does not change in the course of an application's execution and cannot be changed.

constraint

In ADO.NET, an object that represents a rule that enforces referential integrity, such as a constraint that specifies a primary key.

constructor

The method that executes when a class is instantiated. The constructor usually contains initialization code to set member variables to initial values.

container controls

Controls that can host other controls. Examples include *Forms*, *Panels*, and *TabPages*.

control

A class that inherits *System*.*Control*. Controls usually have a visual representation, interact with the user as a part of the user interface, and are hosted in a container control.

controls collection

The collection of a container control that manages the controls it hosts.

custom control

A user-designed control that directly inherits *System*.*Control*.

D

data binding

A way to link data (such as a data set) in your application to the properties of a control.

data consumer

This document is created with the unregistered version of CHM2PDF Pilot

A control that is bound to a data source.

Data Provider

A set of classes that work together to provide managed data access.

DataSet

A class that contains an application copy of data that has been retrieved from a data source.

delegate

An object that acts as a type-safe function pointer. Delegates can be used to invoke the method they point to.

deploying

Installing an application on the computer where it will run.

derived class

A class that is based on another class (called a base class) through inheritance. A derived class inherits the members of its base class and can override or shadow those members.

design time

Represents the state of the application while being designed. Applications at design time consist of noncompiled code that can be manipulated in the code editor or designer window.

destructor

The method that is run just before a class is removed from memory. Generally, a destructor contains any code required to "clean up" after the class. *See also* finalizer.

disconnected data access

Data access that uses a data connection only when executing database operations. This is the default kind of data access implemented by ADO.NET.

This document is created with the unregistered version of CHM2PDF Pilot

docking

In Windows Forms, attaching a control to the edge of the form. You can dock a control by setting the *Dock* property.

E

encapsulation

In component programming, separating the implementation of a component from the interface. Only the public interface of a component is made accessible to the rest of the application. Component data should never be accessible to outside callers.

event

A notification raised by a control or component that a specified condition has occurred.

event handler

A method that is executed when a particular event is raised.

exception

Any class that derives from *System.Exception*. Exceptions are thrown in response to run-time errors. *See also* throwing.

exception handling

The process of dealing with unusual occurrences within code so that it does not cause the program to crash or lose data.

explicit conversion

A conversion of one type to another that cannot be performed automatically. An explicit conversion usually presents some danger of a failed conversion or a loss of data.

extender provider

This document is created with the unregistered version of CHM2PDF Pilot

A specialized type of control that provides design time properties to other controls. *ErrorProvider* and *HelpProvider* are examples of extender providers.

F

field

A member variable of a class or structure.

finalizer

In Visual Basic .NET, the method that is called just before an instance of a class is destroyed. *See also* destructor.

floating-point numbers

A number that has a decimal component. *See also* floating-point types.

floating-point types

Types capable of representing floating-point numbers. Floating-point types include Single (float) and Double (double). *See also* floating-point numbers.

focus

When referring to a control, the condition of being able to receive user input through the keyboard or other input devices. A control that is capable of receiving input is said to have the focus.

foreign key

In a data table, a column that contains a key value that uniquely identifies a record in a related table.

form

The primary unit of the user interface in a Windows Forms application. A form serves as a nexus for data display and user input, and can host controls.

This document is created with the unregistered version of CHM2PDF Pilot

G

GAC

See Global Assembly Cache (GAC).

garbage collection

Automatic memory management provided by the common language runtime. Unused memory is automatically reclaimed by garbage collection without interaction with the application.

GDI+

See Graphic Device Interface (GDI).

getter

The code segment of a property that returns the value the property contains.

Global Assembly Cache (GAC)

A machine-wide cache that stores assemblies that have been specifically designated to be shared between several applications.

globalization

Applying culture-based formatting to existing data, for example, using culture-specific currency symbols and formatting.

Graphic Device Interface (GDI)

In Windows, a graphics display system used by applications to display or print bitmapped text (TrueType fonts), images, and other graphical elements. GDI+ is the name given to the .NET Framework's managed implementation of the GDI.

I

This document is created with the unregistered version of CHM2PDF Pilot

implicit conversion

A conversion between types that can be performed automatically by the common language runtime. An implicit conversion will always succeed and never pose a danger of data loss.

inheritance

In object-oriented programming, creating a new class that incorporates and extends all of the characteristics and functionality of a base class.

inherited control

A user-created control that inherits an existing control. An inherited control inherits all of the functionality of the base control and can incorporate custom functionality as well.

interface

A contract for behavior that can be implemented by a class. Classes that implement the same interface can be expected to provide the members defined by the interface.

L

localization

Providing multiple sets of data specific to different cultures and retrieving the data appropriate to the current culture setting.

logical error

An error that compiles and executes correctly but produces unexpected results. These can be the most difficult errors to detect in application testing.

M

This document is created with the unregistered version of CHM2PDF Pilot

managed code

Code that runs under the common language runtime. The common language runtime handles many tasks that would formerly have been handled in the application's executable. Managed code solves the Windows programming problems of component registration and versioning (sometimes called DLL Hell) because managed code contains all the versioning and type information that the common language runtime needs to run the application. The common language runtime handles registration dynamically at run time rather than statically through the system registry, as is done with applications based on the Component Object Model (COM).

metadata

Data about data. For example, the name, version, security information, and size of a file constitute metadata about the file.

Microsoft Intermediate Language (MSIL)

A low-level language that is just-in-time compiled to native code at run time. All .NET assemblies are represented in the MSIL.

MSIL

See Microsoft Intermediate Language (MSIL).

multidimensional array

An array that contains more than one dimension.

N

namespace

A logical organization of types that perform related functions.

.NET Framework

This document is created with the unregistered version of CHM2PDF Pilot

A managed environment for the development and execution of code. It consists of the common language runtime and .NET Framework base class library.

.NET Framework base class library

A collection of object-oriented types and interfaces that provides object models to assist in the development of your applications.

O

object

The base class for all types in the .NET Framework. More generically, any instance of a class or structure.

object-oriented programming

The process of creating applications by using programmatic constructs to represent real-world objects.

operator overloading

In Visual C#, defining custom behavior for an operator when interacting with custom types.

optimization

The process of examining your application and modifying it to be more efficient in terms of either speed or size of the deployed application.

overloading

Creating multiple methods with the same name but different signatures.

overriding

When inheriting from a base class, providing a new method for one of the base class methods.

P

This document is created with the unregistered version of CHM2PDF Pilot

parent-child relationship

In data access, a relationship between two tables wherein a record from the parent table can be used to retrieve one or more related records from a child table. A parent-child relationship can be defined by using a *DataRelation* object.

polymorphism

The ability of classes to provide different implementations of the same public interface. Thus, two classes that exhibit polymorphism might contain different implementations, but because the interfaces are identical, they can be treated the same in code.

primary key

In a data table, the column that uniquely identifies a record in the table.

private assembly

An assembly that is used by only one application. In .NET Framework programming, most assemblies are private assemblies.

project

The collection of Visual Studio .NET source files that make up an application.

R

Reference type

A type, such as a class, whose instance data is allocated on the heap. Variables that represent Reference types contain a reference to the object on the heap as opposed to the object's data itself. *See also* Value type.

registry

The system repository for information about a computer's settings. The registry can contain configuration information

This document is created with the unregistered version of CHM2PDF Pilot

about programs and system components.

resource

Nonexecutable data, such as a string or image, that is deployed in an application.

role-based security

Code that ensures that users of your application have appropriate permission to access your application based on the identity of the user.

run time

Represents the state of the application while executing. At run time, the application consists of executing code.

run-time error

An error that causes an invalid result or attempts an invalid operation at run time.

S

satellite assembly

When localizing an application, assemblies that contain alternate sets of resources to be used in the application for different cultures.

setter

The code segment of a property that sets the property's value.

shadowing

Hiding a member in a base class with a member in an inherited class that has a different signature.

shared assembly

This document is created with the unregistered version of CHM2PDF Pilot

An assembly that is referenced by several applications on a single machine. A shared assembly must be installed to the Global Assembly Cache (GAC).

shared members

In Visual Basic .NET, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Shared members are called static members in Visual C#.

simple binding

In data binding, binding a control to a single data column. A *TextBox* is an example of a control that can be simply bound.

solution

A group of Visual Studio .NET projects that make up a single functional unit.

SQL

See Structured Query Language (SQL).

static members

In Visual C#, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Static members are called shared members in Visual Basic .NET.

Step Into

Moving from a calling procedure to a called procedure during debugging. Also used in reference to setting breakpoints to stop execution in a specific procedure during debugging.

Step Out

In Break mode, resuming execution of a function call until the call returns.

Step Over

In Break mode, executing a procedure call as a single statement during debugging.

This document is created with the unregistered version of CHM2PDF Pilot

strong name

A name that uniquely identifies an assembly. It consists of the assembly's name, version, and culture information as well as a public key encrypted by a private key. An assembly must have a strong name to be installed to the Global Assembly Cache (GAC).

strongly typed

Refers to types for all variables being explicitly declared. Strong typing eliminates the possibility of type mismatch errors, which are discovered at compile time. Visual C# is inherently strongly typed, and Visual Basic .NET is strongly typed when Option Strict is set to On.

Struct

In Visual C#, a user-defined Value type that serves as a template for an instance of that type. *See also* Structure.

Structure

In Visual Basic .NET, a user-defined Value type that serves as a template for an instance of that type. *See also* Struct .

structured exception handling

Using *Try...Catch...Finally* blocks to handle errors that might occur during code execution. When exceptions are thrown, they are handled in the *Catch* block before code in the *Finally* block is executed.

Structured Query Language (SQL)

The standard language used for issuing commands to databases.

syntax error

An error that occurs because of typographical errors or nonsensical code. In the .NET Framework, syntax errors prevent compilation and are thus discovered at compile time.

T

target audience

This document is created with the unregistered version of CHM2PDF Pilot

The group of end users for whom an application is designed.

threads

The basic unit to which the server allocates processor time. A single process can have multiple threads.

throwing

The act of passing an exception to the next caller in the call stack. *See also* exception.

tracing

A programming technique for recording events, such as exceptions, in an application. Tracing is used during debugging and in the testing phase of application deployment.

transaction

A group of commands (treated as a single unit) that change the data stored in a database. The transaction ensures that the commands are handled in an all-or-nothing fashion if one of the commands fails, all of the commands fail, and any data that was written to the database by the commands is backed out. In this way, transactions maintain the integrity of data in a database.

tuning

The process of making adjustments to a deployed application that do not affect code.

U

Unicode

The universal character encoding scheme for characters and text. Each character is represented by a unique numeric value.

unmanaged code

Code that is not managed by the common language runtime. Unmanaged code is not checked for type-safety and

This document is created with the unregistered version of CHM2PDF Pilot

must be used with extreme care.

user control

A user-defined control that encapsulates other Windows Forms controls with custom functionality in a common container.

V

Value type

A type, such as an Integer, Decimal, or Structure, whose instance data is allocated on the stack. Variables that represent Value types contain the object's instance data itself as opposed to a reference to that data. *See also* Reference type.

W

Watch window

When debugging, the window that allows you to examine the values of predetermined watch variables.

Web Service

A service or component that is implemented by a Web server and can be instantiated and accessed programmatically as a component over the World Wide Web.

X

XML Schema

A description of the data elements contained in an XML file. The XML Schema provides the names of the elements, their types, whether or not they are key fields, and other information.

This document is created with the unregistered version of CHM2PDF Pilot

Glossary

A

abstract class

A class that cannot be instantiated but is used as a base from which other classes can be derived. In Microsoft Visual Basic .NET, abstract classes are declared using the *MustInherit* keyword. In Microsoft Visual C#, abstract classes are declared using the *abstract* keyword.

abstraction

Representing real-world objects as programmatic constructs.

abstract members

A member of a base class that cannot be invoked, but instead provides a template for members of a derived class. In Visual Basic .NET, abstract members are declared using the *MustOverride* keyword. In Visual C#, abstract members are declared using the *abstract* keyword.

ADO.NET

The data access architecture for the Microsoft .NET Framework. ADO.NET is built around a disconnected data access model that uses a set of classes called a Data Provider to retrieve data from a data source.

alias

The name you use to identify resources on the Web. Aliases represent physical resources on the Web server, such as a Web form, an HTML page, or a graphic.

anchoring

In Windows Forms controls, the resizing and repositioning of a control when its parent form is resized. You can anchor a control by setting the *Anchor* property.

This document is created with the unregistered version of CHM2PDF Pilot

assembly

The primary unit of deployment for a .NET application. An assembly is either an executable application (EXE) or a class library (DLL).

assembly manifest

The metadata for the assembly. It contains all of the information needed to describe the assembly to the common language runtime.

authentication

The process of determining the identity of a user. In effect, authentication validates that the users are who they say they are. *See also* authorization.

authorization

The process of allowing access to an application or resource based on credentials supplied by the user. *See also* authentication.

B

base class

A class that provides properties and methods as a foundation for a derived class. In objected-oriented programming, one class can be based on another through inheritance. Using this technique, the base class provides characteristics (such as properties and methods) to a derived class. The derived class can reuse, modify, or add to the members of the base class.

Bezier

A curve specified by four points. Two points specify the ends of the curve, and the other two points influence the shape of the curve between the end points.

Boolean

A data type that can represent a value of either true or false.

This document is created with the unregistered version of CHM2PDF Pilot

bootstrapper

In deployment, an application that installs Microsoft Windows Installer on machines that do not currently have it installed.

boundary condition

In testing, test conditions at the limit of normal operation and just over or under those limits.

boxing

The act of wrapping a Value type, such as an Integer, inside an object so that it can be treated like a Reference type.

Break mode

In debugging, a state of suspended execution wherein the application can be executed in a line-by-line manner, and individual program variables can be examined.

breakpoint

In debugging, a predetermined point where execution of an application will halt and enter Break mode.

C

cast

An explicit conversion from one type to another.

circular reference

A condition that occurs when two objects refer to each other but do not have an external reference. Circular references are automatically detected and disposed of by the garbage collector.

class

A user-defined Reference type that serves as a template for an object of that type.

This document is created with the unregistered version of CHM2PDF Pilot

CLR

See common language runtime (CLR).

code access security

Verification that all callers to code have appropriate permission to access it.

code tuning

The act of examining application code and optimizing for faster execution. Code tuning is usually an iterative process that involves several successive rounds of planning, improvement, and assessment of the results.

common language runtime (CLR)

The environment in which managed code executes. The common language runtime provides just-in-time compilation, enforces type safety, and manages memory through garbage collection.

Common Type System

A set of types that are used by all .NET languages, thus ensuring .NET language type compatibility.

complex binding

In data binding, binding more than one column to a single control. The *DataGrid* is an example of a control that can be complexly bound.

component

Strictly speaking, a class that implements *IComponent* or inherits *System*.*Component*. More generally, any class that provides a reusable set of functionality.

connected data access

Data access that requires a data connection that is always open.

Constant

This document is created with the unregistered version of CHM2PDF Pilot

A type that represents a value that does not change in the course of an application's execution and cannot be changed.

constraint

In ADO.NET, an object that represents a rule that enforces referential integrity, such as a constraint that specifies a primary key.

constructor

The method that executes when a class is instantiated. The constructor usually contains initialization code to set member variables to initial values.

container controls

Controls that can host other controls. Examples include *Forms*, *Panels*, and *TabPages*.

control

A class that inherits *System*.*Control*. Controls usually have a visual representation, interact with the user as a part of the user interface, and are hosted in a container control.

controls collection

The collection of a container control that manages the controls it hosts.

custom control

A user-designed control that directly inherits *System*.*Control*.

D

data binding

A way to link data (such as a data set) in your application to the properties of a control.

data consumer

This document is created with the unregistered version of CHM2PDF Pilot

A control that is bound to a data source.

Data Provider

A set of classes that work together to provide managed data access.

DataSet

A class that contains an application copy of data that has been retrieved from a data source.

delegate

An object that acts as a type-safe function pointer. Delegates can be used to invoke the method they point to.

deploying

Installing an application on the computer where it will run.

derived class

A class that is based on another class (called a base class) through inheritance. A derived class inherits the members of its base class and can override or shadow those members.

design time

Represents the state of the application while being designed. Applications at design time consist of noncompiled code that can be manipulated in the code editor or designer window.

destructor

The method that is run just before a class is removed from memory. Generally, a destructor contains any code required to "clean up" after the class. *See also* finalizer.

disconnected data access

Data access that uses a data connection only when executing database operations. This is the default kind of data access implemented by ADO.NET.

This document is created with the unregistered version of CHM2PDF Pilot

docking

In Windows Forms, attaching a control to the edge of the form. You can dock a control by setting the *Dock* property.

E

encapsulation

In component programming, separating the implementation of a component from the interface. Only the public interface of a component is made accessible to the rest of the application. Component data should never be accessible to outside callers.

event

A notification raised by a control or component that a specified condition has occurred.

event handler

A method that is executed when a particular event is raised.

exception

Any class that derives from *System.Exception*. Exceptions are thrown in response to run-time errors. *See also* throwing.

exception handling

The process of dealing with unusual occurrences within code so that it does not cause the program to crash or lose data.

explicit conversion

A conversion of one type to another that cannot be performed automatically. An explicit conversion usually presents some danger of a failed conversion or a loss of data.

extender provider

This document is created with the unregistered version of CHM2PDF Pilot

A specialized type of control that provides design time properties to other controls. *ErrorProvider* and *HelpProvider* are examples of extender providers.

F

field

A member variable of a class or structure.

finalizer

In Visual Basic .NET, the method that is called just before an instance of a class is destroyed. *See also* destructor.

floating-point numbers

A number that has a decimal component. *See also* floating-point types.

floating-point types

Types capable of representing floating-point numbers. Floating-point types include Single (float) and Double (double). *See also* floating-point numbers.

focus

When referring to a control, the condition of being able to receive user input through the keyboard or other input devices. A control that is capable of receiving input is said to have the focus.

foreign key

In a data table, a column that contains a key value that uniquely identifies a record in a related table.

form

The primary unit of the user interface in a Windows Forms application. A form serves as a nexus for data display and user input, and can host controls.

This document is created with the unregistered version of CHM2PDF Pilot

G

GAC

See Global Assembly Cache (GAC).

garbage collection

Automatic memory management provided by the common language runtime. Unused memory is automatically reclaimed by garbage collection without interaction with the application.

GDI+

See Graphic Device Interface (GDI).

getter

The code segment of a property that returns the value the property contains.

Global Assembly Cache (GAC)

A machine-wide cache that stores assemblies that have been specifically designated to be shared between several applications.

globalization

Applying culture-based formatting to existing data, for example, using culture-specific currency symbols and formatting.

Graphic Device Interface (GDI)

In Windows, a graphics display system used by applications to display or print bitmapped text (TrueType fonts), images, and other graphical elements. GDI+ is the name given to the .NET Framework's managed implementation of the GDI.

I

This document is created with the unregistered version of CHM2PDF Pilot

implicit conversion

A conversion between types that can be performed automatically by the common language runtime. An implicit conversion will always succeed and never pose a danger of data loss.

inheritance

In object-oriented programming, creating a new class that incorporates and extends all of the characteristics and functionality of a base class.

inherited control

A user-created control that inherits an existing control. An inherited control inherits all of the functionality of the base control and can incorporate custom functionality as well.

interface

A contract for behavior that can be implemented by a class. Classes that implement the same interface can be expected to provide the members defined by the interface.

L

localization

Providing multiple sets of data specific to different cultures and retrieving the data appropriate to the current culture setting.

logical error

An error that compiles and executes correctly but produces unexpected results. These can be the most difficult errors to detect in application testing.

M

This document is created with the unregistered version of CHM2PDF Pilot

managed code

Code that runs under the common language runtime. The common language runtime handles many tasks that would formerly have been handled in the application's executable. Managed code solves the Windows programming problems of component registration and versioning (sometimes called DLL Hell) because managed code contains all the versioning and type information that the common language runtime needs to run the application. The common language runtime handles registration dynamically at run time rather than statically through the system registry, as is done with applications based on the Component Object Model (COM).

metadata

Data about data. For example, the name, version, security information, and size of a file constitute metadata about the file.

Microsoft Intermediate Language (MSIL)

A low-level language that is just-in-time compiled to native code at run time. All .NET assemblies are represented in the MSIL.

MSIL

See Microsoft Intermediate Language (MSIL).

multidimensional array

An array that contains more than one dimension.

N

namespace

A logical organization of types that perform related functions.

.NET Framework

This document is created with the unregistered version of CHM2PDF Pilot

A managed environment for the development and execution of code. It consists of the common language runtime and .NET Framework base class library.

.NET Framework base class library

A collection of object-oriented types and interfaces that provides object models to assist in the development of your applications.

O

object

The base class for all types in the .NET Framework. More generically, any instance of a class or structure.

object-oriented programming

The process of creating applications by using programmatic constructs to represent real-world objects.

operator overloading

In Visual C#, defining custom behavior for an operator when interacting with custom types.

optimization

The process of examining your application and modifying it to be more efficient in terms of either speed or size of the deployed application.

overloading

Creating multiple methods with the same name but different signatures.

overriding

When inheriting from a base class, providing a new method for one of the base class methods.

P

This document is created with the unregistered version of CHM2PDF Pilot

parent-child relationship

In data access, a relationship between two tables wherein a record from the parent table can be used to retrieve one or more related records from a child table. A parent-child relationship can be defined by using a *DataRelation* object.

polymorphism

The ability of classes to provide different implementations of the same public interface. Thus, two classes that exhibit polymorphism might contain different implementations, but because the interfaces are identical, they can be treated the same in code.

primary key

In a data table, the column that uniquely identifies a record in the table.

private assembly

An assembly that is used by only one application. In .NET Framework programming, most assemblies are private assemblies.

project

The collection of Visual Studio .NET source files that make up an application.

R

Reference type

A type, such as a class, whose instance data is allocated on the heap. Variables that represent Reference types contain a reference to the object on the heap as opposed to the object's data itself. *See also* Value type.

registry

The system repository for information about a computer's settings. The registry can contain configuration information

This document is created with the unregistered version of CHM2PDF Pilot

about programs and system components.

resource

Nonexecutable data, such as a string or image, that is deployed in an application.

role-based security

Code that ensures that users of your application have appropriate permission to access your application based on the identity of the user.

run time

Represents the state of the application while executing. At run time, the application consists of executing code.

run-time error

An error that causes an invalid result or attempts an invalid operation at run time.

S

satellite assembly

When localizing an application, assemblies that contain alternate sets of resources to be used in the application for different cultures.

setter

The code segment of a property that sets the property's value.

shadowing

Hiding a member in a base class with a member in an inherited class that has a different signature.

shared assembly

This document is created with the unregistered version of CHM2PDF Pilot

An assembly that is referenced by several applications on a single machine. A shared assembly must be installed to the Global Assembly Cache (GAC).

shared members

In Visual Basic .NET, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Shared members are called static members in Visual C#.

simple binding

In data binding, binding a control to a single data column. A *TextBox* is an example of a control that can be simply bound.

solution

A group of Visual Studio .NET projects that make up a single functional unit.

SQL

See Structured Query Language (SQL).

static members

In Visual C#, members that belong to the class instead of any one instance of the class. Shared members can be accessed before an instance of a class is created. Static members are called shared members in Visual Basic .NET.

Step Into

Moving from a calling procedure to a called procedure during debugging. Also used in reference to setting breakpoints to stop execution in a specific procedure during debugging.

Step Out

In Break mode, resuming execution of a function call until the call returns.

Step Over

In Break mode, executing a procedure call as a single statement during debugging.

This document is created with the unregistered version of CHM2PDF Pilot

strong name

A name that uniquely identifies an assembly. It consists of the assembly's name, version, and culture information as well as a public key encrypted by a private key. An assembly must have a strong name to be installed to the Global Assembly Cache (GAC).

strongly typed

Refers to types for all variables being explicitly declared. Strong typing eliminates the possibility of type mismatch errors, which are discovered at compile time. Visual C# is inherently strongly typed, and Visual Basic .NET is strongly typed when Option Strict is set to On.

Struct

In Visual C#, a user-defined Value type that serves as a template for an instance of that type. *See also* Structure.

Structure

In Visual Basic .NET, a user-defined Value type that serves as a template for an instance of that type. *See also* Struct .

structured exception handling

Using *Try...Catch...Finally* blocks to handle errors that might occur during code execution. When exceptions are thrown, they are handled in the *Catch* block before code in the *Finally* block is executed.

Structured Query Language (SQL)

The standard language used for issuing commands to databases.

syntax error

An error that occurs because of typographical errors or nonsensical code. In the .NET Framework, syntax errors prevent compilation and are thus discovered at compile time.

T

target audience

This document is created with the unregistered version of CHM2PDF Pilot

The group of end users for whom an application is designed.

threads

The basic unit to which the server allocates processor time. A single process can have multiple threads.

throwing

The act of passing an exception to the next caller in the call stack. *See also* exception.

tracing

A programming technique for recording events, such as exceptions, in an application. Tracing is used during debugging and in the testing phase of application deployment.

transaction

A group of commands (treated as a single unit) that change the data stored in a database. The transaction ensures that the commands are handled in an all-or-nothing fashion if one of the commands fails, all of the commands fail, and any data that was written to the database by the commands is backed out. In this way, transactions maintain the integrity of data in a database.

tuning

The process of making adjustments to a deployed application that do not affect code.

U

Unicode

The universal character encoding scheme for characters and text. Each character is represented by a unique numeric value.

unmanaged code

Code that is not managed by the common language runtime. Unmanaged code is not checked for type-safety and

This document is created with the unregistered version of CHM2PDF Pilot

must be used with extreme care.

user control

A user-defined control that encapsulates other Windows Forms controls with custom functionality in a common container.

V

Value type

A type, such as an Integer, Decimal, or Structure, whose instance data is allocated on the stack. Variables that represent Value types contain the object's instance data itself as opposed to a reference to that data. *See also* Reference type.

W

Watch window

When debugging, the window that allows you to examine the values of predetermined watch variables.

Web Service

A service or component that is implemented by a Web server and can be instantiated and accessed programmatically as a component over the World Wide Web.

X

XML Schema

A description of the data elements contained in an XML file. The XML Schema provides the names of the elements, their types, whether or not they are key fields, and other information.