

BEN WATSON

C# 4.0

HOW-TO

Real Solutions for C# 4.0 Programmers

SAMS

C# 4.0 How-To

Copyright © 2010 by Pearson Education, Inc.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-672-33063-6

ISBN-10: 0-672-33063-6

Library of Congress Cataloging-in-Publication Data

Watson, Ben, 1980–

C# 4.0 how-to / Ben Watson.
p. cm.

Includes index.

ISBN 978-0-672-33063-6 (pbk. : alk. paper) 1. C# (Computer program language) I. Title.

QA76.73.C154W38 2010

005.13'3—dc22

2010002735

Printed in the United States of America

First Printing March 2010

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Bulk Sales

Sams Publishing offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales
1-800-382-3419
corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact

International Sales
international@pearson.com

Editor-in-Chief

Karen Gettman

Executive Editor

Neil Rowe

Acquisitions Editor

Brook Farling

Development Editor

Mark Renfrow

Managing Editor

Kristy Hart

Project Editor

Lori Lyons

Copy Editor

Bart Reed

Indexer

Brad Herriman

Proofreader

Sheri Cain

Technical Editor

Mark Strawmyer

Publishing**Coordinator**

Cindy Teeters

Designer

Gary Adair

Compositor

Nonie Ratcliff

INTRODUCTION

Overview of C# 4.0 How-To

This book is very different from a typical “bible” approach to a topic. By structuring the book as a “how-to,” it presents the material by scenario in steps that are easily followed. Throughout, I have tried to keep the explanatory text to the minimum necessary and keep the focus on the code itself. Often, you will find comments embedded in the code to explain non-obvious bits.

This book is not strictly a language/library book. Besides covering the language features themselves, it dives into practical examples of application patterns, useful algorithms, and handy tips that are applicable in many situations.

Developers, both beginner and advanced, will find hundreds of useful topics in this book. Whether it’s a section on lesser-known C# operators, how to sort strings that contain numbers in them, or how to implement Undo, this book contains recipes that are useful in a wide variety of situations, regardless of skill level.

In short, this is the book I wish I had on my desk when I was first learning programming and C# as well as now, whenever I need a quick reference or reminder about how to do something.

How-To Benefit from This Book

We designed this book to be easy to read from cover to cover. The goal is to gain a full understanding of C# 4.0. The subject matter is divided into four parts with easy-to-navigate and easy-to-use chapters.

Part I, “C# Fundamentals,” covers the common C# functionality that you will use in every type of programming. While it may seem basic, there are a lot of tips to help you get the most of these fundamental topics.

- ▶ Chapter 1, “Type Fundamentals”
- ▶ Chapter 2, “Creating Versatile Types”
- ▶ Chapter 3, “General Coding”
- ▶ Chapter 4, “Exceptions”
- ▶ Chapter 5, “Numbers”

- ▶ Chapter 6, “Enumerations”
- ▶ Chapter 7, “Strings”
- ▶ Chapter 8, “Regular Expressions”
- ▶ Chapter 9, “Generics”

Part II, “Handling Data,” discusses how to store and manipulate data, including Internet-based data.

- ▶ Chapter 10, “Collections”
- ▶ Chapter 11, “Files and Serialization”
- ▶ Chapter 12, “Networking and the Web”
- ▶ Chapter 13, “Databases”
- ▶ Chapter 14, “XML”

Part III “User Interaction,” covers the most popular user interface paradigms in .Net, whether you work on the desktop, the Web, or both.

- ▶ Chapter 15, “Delegates, Events, and Anonymous Methods”
- ▶ Chapter 16, “Windows Forms”
- ▶ Chapter 17, “Graphics with Windows Forms and GDI+”
- ▶ Chapter 18, “WPF”
- ▶ Chapter 19, “ASP.NET”
- ▶ Chapter 20, “Silverlight”

Part IV, “Advanced C#,” has the advanced stuff to really take your applications to the next level in terms of performance, design patterns, useful algorithms, and more.

- ▶ Chapter 21, “LINQ”
- ▶ Chapter 22, “Memory Management”
- ▶ Chapter 23, “Threads, Asynchronous, and Parallel Programming”
- ▶ Chapter 24, “Reflection and Creating Plugins”
- ▶ Chapter 25, “Application Patterns and Tips”
- ▶ Chapter 26, “Interacting with the OS and Hardware”
- ▶ Chapter 27, “Fun Stuff and Loose Ends”
- ▶ Appendix A, “Essential Tools”

All of the code was developed using prerelease versions of Visual Studio 2010, but you can use earlier versions in many cases, especially for code that does not require .NET 4. If you do not have Visual Studio, you can download the Express edition from www.microsoft.com/express/default.aspx. This version will enable you to build nearly all the code samples in this book.

You can access the code samples used in this book by registering on the book's website at informit.com/register. Go to this URL, sign in, and enter the ISBN to register (free site registration required). After you register, look on your Account page, under Registered Products, for a link to Access Bonus Content.

How-To Continue Expanding Your Knowledge

No book can completely cover C#, the .NET Framework, or probably even hope to cover a small topic within that world. And if there were, you probably couldn't lift it, let alone read it in your lifetime.

Once you've mastered the essentials, there are plenty of resources to get your questions answered and dive deeply into .NET.

Thankfully, the MSDN documentation for .NET (located at <http://msdn.microsoft.com/en-us/library/aa139615.aspx>) is top-notch. Most topics have code samples and an explanation use. An added bonus is the ability at the bottom of every topic for anyone to add useful content. There are many good tips found here from other .NET developers.

The .NET Development forums (<http://social.msdn.microsoft.com/Forums/en-US/category/netdevelopment>) are an excellent place to get your questions answered by the experts, who, in many cases, were involved in the development and testing of .NET.

I have also found StackOverflow.com a good place to get questions answered.

The best advice I can give on how to continue expanding your knowledge is to just write software. Keep at it, think of new projects, use new technologies, go beyond your abilities. This and other books are very useful, to a point. After that, you just need to dive in and start coding, using the book as a faithful reference when you don't know how to approach a topic.

Happy coding!

CHAPTER 2

Creating Versatile Types

IN THIS CHAPTER

- ▶ Format a Type with `ToString()`
- ▶ Make Types Equatable
- ▶ Make Types Hashable with `GetHashCode()`
- ▶ Make Types Sortable
- ▶ Give Types an Index
- ▶ Notify Clients when Changes Happen
- ▶ Overload Appropriate Operators
- ▶ Convert One Type to Another
- ▶ Prevent Inheritance
- ▶ Allow Value Type to Be Null

Whenever you create your own classes, you need to consider the circumstances under which they could be used. For example, will two instances of your `Item` struct ever be compared for equality? Will your `Person` class need to be serializable, or sortable?

NOTE Versatility means being able to do many things well. When you're creating your own types, it means outfitting your objects with enough "extra" stuff that they can easily be used in a wide variety of situations.

This chapter is all about making your own objects as useful and versatile as possible. In many cases, this means implementing the standard interfaces that .NET provides or simply overriding base class methods.

Format a Type with `ToString()`

Scenario/Problem: You need to provide a string representation of an object for output and debugging purposes.

Solution: By default, `ToString()` will display the type's name. To show your own values, you must override the method with one of your own. To illustrate this, let's continue our `Vertex3d` class example from the previous chapter.

Assume the class initially looks like this:

```
struct Vertex3d
{
    private double _x;
    private double _y;
    private double _z;

    public double X
    {
        get { return _x; }
        set { _x = value; }
    }

    public double Y
    {
        get { return _y; }
        set { _y = value; }
    }
}
```

```
public double Z
{
    get { return _z; }
    set { _z = value; }
}

public Vertex3d(double x, double y, double z)
{
    this._x = x;
    this._y = y;
    this._z = z;
}
}
```

Override ToString() for Simple Output

To get a simple string representation of the vertex, override ToString() to return a string of your choosing.

```
public override string ToString()
{
    return string.Format("{0}, {1}, {2}", X, Y, Z);
}
```

The code

```
Vertex3d v = new Vertex3d(1.0, 2.0, 3.0);
Trace.WriteLine(v.ToString());
```

produces the following output:

```
(1, 2, 3)
```

Implement Custom Formatting for Fine Control

Scenario/Problem: You need to provide consumers of your class fine-grained control over how string representations of your class look.

Solution: Although the ToString() implementation gets the job done, and is especially handy for debugging (Visual Studio will automatically call ToString() on objects in the debugger windows), it is not very flexible. By implementing IFormattable on your type, you can create a version of ToString() that is as flexible as you need.

Let's create a simple format syntax that allows us to specify which of the three values to print. To do this, we'll define the following format string:

```
"X, Y"
```

This tells `Vertex3d` to print out `X` and `Y`. The comma and space (and any other character) will be output as-is.

The struct definition will now be as follows:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace VertexDemo
{
    struct Vertex3d : IFormattable
    {
        ...
        public string ToString(string format, IFormatProvider formatProvider)
        {
            //"G" is .Net's standard for general formatting--all
            //types should support it
            if (format == null) format = "G";

            // is the user providing their own format provider?
            if (formatProvider != null)
            {
                ICustomFormatter formatter =
                    formatProvider.GetFormat(this.GetType())
                        as ICustomFormatter;
                if (formatter != null)
                {
                    return formatter.Format(format, this, formatProvider);
                }
            }

            //formatting is up to us, so let's do it
            if (format == "G")
            {
                return string.Format("{0}, {1}, {2}", X, Y, Z);
            }

            StringBuilder sb = new StringBuilder();
            int sourceIndex = 0;
```

```
while (sourceIndex < format.Length)
{
    switch (format[sourceIndex])
    {
        case 'X':
            sb.Append(X.ToString());
            break;
        case 'Y':
            sb.Append(Y.ToString());
            break;
        case 'Z':
            sb.Append(Z.ToString());
            break;
        default:
            sb.Append(format[sourceIndex]);
            break;
    }
    sourceIndex++;
}
return sb.ToString();
}
}
```

The `formatProvider` argument allows you to pass in a formatter that does something different from the type's own formatting (say, if you can't change the implementation of `ToString()` on `Vertex3d` for some reason, or you need to apply different formatting in specific situations). You'll see how to define a custom formatter in the next section.

Formatting with `ICustomFormatter` and `StringBuilder`

Scenario/Problem: You need a general-purpose formatter than can apply custom formats to many types of objects.

Solution: Use `ICustomFormatter` and `StringBuilder`. This example prints out type information, as well as whatever the custom format string specifies for the given types.

```
class TypeFormatter : IFormatProvider, ICustomFormatter
{
    public object GetFormat(Type formatType)
    {
```

```

        if (formatType == typeof(ICustomFormatter)) return this;
        return Thread.CurrentThread.CurrentCulture.GetFormat(formatType);
    }

    public string Format(string format, object arg, IFormatProvider
formatProvider)
    {
        string value;
        IFormattable formattable = arg as IFormattable;
        if (formattable == null)
        {
            value = arg.ToString();
        }
        else
        {
            value = formattable.ToString(format, formatProvider);
        }
        return string.Format("Type: {0}, Value: {1}", arg.GetType(),
value);
    }
}

```

The class can be used like this:

```

Vertex3d v = new Vertex3d(1.0, 2.0, 3.0);
Vertex3d v2 = new Vertex3d(4.0, 5.0, 6.0);
TypeFormatter formatter = new TypeFormatter();
StringBuilder sb = new StringBuilder();
sb.AppendFormat(formatter, "{0:(X Y)}; {1:[X, Y, Z]}", v, v2);
Console.WriteLine(sb.ToString());

```

The following output is produced:

```
Type: ch02.Vertex3d, Value: (1 2); Type: ch02.Vertex3d, Value: [4, 5, 6]
```

Make Types Equatable

Scenario/Problem: You need to determine if two objects are equal.

Solution: You should override `Object.Equals()` and also implement the `IEquatable<T>` interface.

By default, `Equals()` on a reference type checks to see if the objects refer to the same location in memory. This may be acceptable in some circumstances, but often, you'll

want to provide your own behavior. With value types, the default behavior is to reflect over each field and do a bit-by-bit comparison. This can have a very negative impact on performance, and in nearly every case you should provide your own implementation of `Equals()`.

```
struct Vertex3d : IFormattable, IEquatable<Vertex3d>
{
    ...
    public override bool Equals(object obj)
    {
        if (obj == null)
            return false;
        if (obj.GetType() != this.GetType())
            return false;
        return Equals((Vertex3d)obj);
    }

    public bool Equals(Vertex3d other)
    {
        /* If Vertex3d were a reference type you would also need:
        * if ((object)other == null)
        *     return false;
        *
        * if (!base.Equals(other))
        *     return false;
        */

        return this._x == other._x
            && this._y == other._y
            && this._z == other._z;
    }
}
```

NOTE Pay special attention to the note in `Equals(Vertex3d other)`. If `Vertex3d` was a reference type and `other` was null, the type-safe version of the function would be called, not the `Object` version. You also need to call all the base classes in the hierarchy so they have an opportunity to check their own fields.

There's nothing stopping you from also implementing `IEquatable<string>` (or any other type) on your type—you can define it however you want. Use with caution, however, because this may confuse people who have to use your code.

Make Types Hashable with GetHashCode()

Scenario/Problem: You want to use your class as the key part in a collection that indexes values by unique keys. To do this, your class must be able to convert the “essence” of its values into a semi-unique integer ID.

Solution: You almost always want to override `GetHashCode()`, especially with value types, for performance reasons. Generating a hash value is generally done by somehow distilling the data values in your class to an integer representation that is different for every value your class can have. You should override `GetHashCode()` whenever you override `Equals()`.

```
public override int GetHashCode()
{
    //note: This is just a sample hash algorithm.
    //picking a good algorithm can require some
    //research and experimentation
    return (((int)_x ^ (int)_z) << 16) |
           (((int)_y ^ (int)_z) & 0x0000FFFF);
}
```

NOTE Hash codes are not supposed to be unique for every possible set of values your type can have. This is actually impossible, as you can deduce from the previous code sample. For this reason, comparing hash values is not a good way to compute equality.

Make Types Sortable

Scenario/Problem: Objects of your type will be sorted in a collection or otherwise compared to each other.

Solution: Because you often don’t know how your type will be used, making the objects sortable is highly recommended whenever possible.

In the `Vector3d` class example, in order to make the objects comparable, we’ll add an `_id` field and implement the `IComparable<Vertex3d>` interface.

The `_id` field will be what determines the order (it doesn’t make much sense to sort on coordinates, generally).

The sorting function is simple. It takes an object of `Vertex3d` and returns one of three values:

< 0	this is less than other
0	this is same as other
> 0	this is greater than other

Within the `CompareTo` function, you can do anything you want to arrive at those values. In our case, we can do the comparison ourselves or just call the same function on the `_id` field.

```
struct Vertex3d : IFormattable, IEquatable<Vertex3d>,
                IComparable<Vertex3d>
{
    private int _id;

    public int Id
    {
        get
        {
            return _id;
        }
        set
        {
            _id = value;
        }
    }

    public Vertex3d(double x, double y, double z)
    {
        _x = x;
        _y = y;
        _z = z;

        _id = 0;
    }
    ...
    public int CompareTo(Vertex3d other)
    {
        if (_id < other._id)
            return -1;
        if (_id == other._id)
            return 0;
        return 1;
        /* We could also just do this:
        * return _id.CompareTo(other._id);
        */
    }
}
```

```

        * */
    }
}

```

Give Types an Index

Scenario/Problem: Your type has data values that can be accessed by some kind of index, either numerical or string based.

Solution: You can index by any type. The most common index types are `int` and `string`.

Implement a Numerical Index

You use the array access brackets to define an index on the `this` object, like this sample:

```

public double this[int index]
{
    get
    {
        switch (index)
        {
            case 0: return _x;
            case 1: return _y;
            case 2: return _z;
            default: throw new ArgumentOutOfRangeException("index",
                "Only indexes 0-2 valid!");
        }
    }
    set
    {
        switch (index)
        {
            case 0: _x = value; break;
            case 1: _y = value; break;
            case 2: _z = value; break;
            default: throw new ArgumentOutOfRangeException("index",
                "Only indexes 0-2 valid!");
        }
    }
}
}

```

Implement a String Index

Unlike regular arrays, however, you are not limited to integer indices. You can use any type at all, most commonly strings, as in this example:

```
public double this[string dimension]
{
    get
    {
        switch (dimension)
        {
            case "x":
            case "X": return _x;
            case "y":
            case "Y": return _y;
            case "z":
            case "Z": return _z;
            default: throw new ArgumentOutOfRangeException("dimension",
                "Only dimensions X, Y, and Z are valid!");
        }
    }
    set
    {
        switch (dimension)
        {
            case "x":
            case "X": _x = value; break;
            case "y":
            case "Y": _y = value; break;
            case "z":
            case "Z": _z = value; break;
            default: throw new ArgumentOutOfRangeException("dimension",
                "Only dimensions X, Y, and Z are valid!");
        }
    }
}
```

Sample usage:

```
Vertex3d v = new Vertex3d(1, 2, 3);
Console.WriteLine(v[0]);
Console.WriteLine(v["Z"]);
```

Output:

```
1
3
```

Notify Clients when Changes Happen

Scenario/Problem: You want users of your class to know when data inside the class changes.

Solution: Implement the `INotifyPropertyChanged` interface (located in `System.ComponentModel`).

```
using System.ComponentModel;
...
class MyDataClass : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    protected void OnPropertyChanged(string propertyName)
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new
PropertyChangeEventArgs(propertyName));
        }
    }

    private int _tag = 0;
    public int Tag
    {
        get
        { return _tag; }
        set
        {
            _tag = value;
            OnPropertyChanged("Tag");
        }
    }
}
```

The Windows Presentation Foundation (WPF) makes extensive use of this interface for data binding, but you can use it for your own purposes as well.

To consume such a class, use code similar to this:

```
void WatchObject(object obj)
{
    INotifyPropertyChanged watchableObj = obj as INotifyPropertyChanged;
```

```

    if (watchableObj != null)
    {
        watchableObj.PropertyChanged += new
            PropertyChangedEventHandler(data_PropertyChanged);
    }
}

void data_PropertyChanged(object sender, PropertyChangedEventArgs e)
{
    //do something when data changes
}

```

Overload Appropriate Operators

Scenario/Problem: You want to define what the +, *, ==, and != operators do when called on your type.

Solution: Operator overloading is like sugar: a little is sweet, but a lot will make you sick. Ensure that you only use this technique for situations that make sense.

Implement operator +

Notice that the method is `public static` and takes both operators as arguments.

```

public static Vertex3d operator +(Vertex3d a, Vertex3d b)
{
    return new Vertex3d(a.X + b.X, a.Y + b.Y, a.Z + b.Z);
}

```

The same principal can be applied to the -, *, /, %, &, |, <<, >>, !, ~, ++, and -- operators as well.

Implement operator == and operator !=

These should always be implemented as a pair. Because we've already implemented a useful `Equals()` method, just call that instead.

```

public static bool operator ==(Vertex3d a, Vertex3d b)
{
    return a.Equals(b);
}

public static bool operator !=(Vertex3d a, Vertex3d b)
{

```

```

    return !(a==b);
}

```

What if the type is a reference type? In this case, you have to handle null values for both a and b, as in this example:

```

public static bool operator ==(CatalogItem a, CatalogItem b)
{
    if ((object)a == null && (object)b == null)
        return true;
    if ((object)a == null || (object)b == null)
        return false;
    return a.Equals(b);
}

public static bool operator !=(CatalogItem a, CatalogItem b)
{
    return !(a == b);
}

```

Convert One Type to Another

Scenario/Problem: You need to convert one type to another, either automatically or by requiring an explicit cast.

Solution: Implement a conversion operator. There are two types of conversion operators: *implicit* and *explicit*. To understand the difference, we'll implement a new struct called `Vertex3i` that is the same as `Vertex3d`, except the dimensions are integers instead of doubles.

Explicit Conversion (Loss of Precision)

Explicit conversion is encouraged when the conversion will result in a loss of precision. When you're converting from `System.Double` to `System.Int32`, for example, all of the decimal precision is lost. You don't (necessarily) want the compiler to allow this conversion automatically, so you make it explicit. This code goes in the `Vertex3d` class:

```

public static explicit operator Vertex3i(Vertex3d vertex)
{
    return new Vertex3i((Int32)vertex._x, (Int32)vertex._y,
        (Int32)vertex._z);
}

```

To convert from `Vertex3d` to `Vertex3i` then, you would do the following:

```
Vertex3d vd = new Vertex3d(1.5, 2.5, 3.5);
Vertex3i vi = (Vertex3i)vd;
```

If you tried it without the cast, you would get the following:

```
//Vertex3i vi = vd;
Error: Cannot implicitly convert type 'Vertex3d' to 'Vertex3i'.
       An explicit conversion exists (are you missing a cast?)
```

Implicit Conversion (No Loss of Precision)

If there will not be any loss in precision, then the conversion can be implicit, meaning the compiler will allow you to assign the type with no explicit conversion. We can implement this type of conversion in the `Vertex3i` class because it can convert up to a double with no loss of precision.

```
public static implicit operator Vertex3d(Vertex3i vertex)
{
    return new Vertex3d(vertex._x, vertex._y, vertex._z);
}
```

Now we can assign without casting:

```
Vertex3i vi = new Vertex3i(1, 2, 3);
Vertex3d vd = vi;
```

Prevent Inheritance

Scenario/Problem: You want to prevent users of your class from inheriting from it.

Solution: Mark the class as sealed.

```
sealed class MyClass
{
    ...
}
```

Structs are inherently sealed.

Prevent Overriding of a Single Method

Scenario/Problem: You don't want to ban inheritance on your type, but you do want to prevent certain methods or properties from being overridden.

Solution: Put `sealed` as part of the method or property definition.

```
class ParentClass
{
    public virtual void MyFunc() { }
}

class ChildClass : ParentClass
{
    //seal base class function into this class
    public sealed override void MyFunc() { }
}

class GrandChildClass : ChildClass
{
    //yields compile error
    public override void MyFunc() { }
}
```

Allow Value Type to Be Null

Scenario/Problem: You need to assign null to a value type to indicate the lack of a value. This scenario often occurs when working with databases, which allow any data type to be null.

Solution: This isn't technically something you need to implement in your class. .NET 2.0 introduced the `Nullable<T>` type, which wraps any value type into something that can be null. It's useful enough that there is a special C# syntax shortcut to do this. The following two lines of code are semantically equivalent:

```
Nullable<int> _id;
int? _id;
```

Let's make the `_id` field in our `Vertex3d` class `Nullable<T>` to indicate the lack of a valid value. The following code snippet demonstrates how it works:

```
struct Vertex3d : IFormattable, IEquatable<Vertex3d>,
                IComparable<Vertex3d>
{
    private int? _id;

    public int? Id
    {
        get
        {
            return _id;
        }
        set
        {
            _id = value;
        }
    }
    ...
}
...
Vertex3d vn = new Vertex3d(1, 2, 3);
vn.Id = 3;    //ok
vn.Id = null; //ok
try
{
    Console.WriteLine("ID: {0}", vn.Id.Value); //throws
}
catch (InvalidOperationException)
{
    Console.WriteLine("Oops--you can't get a null value!");
}

if (vn.Id.HasValue)
{
    Console.WriteLine("ID: {0}", vn.Id.Value);
}
}
```

Symbols

- != operator, implementing, 39-40**
- + operator, implementing, 39**
- == operator, implementing, 39-40**
- 3D geometry, WPF, rendering, 389-392**
- 3D surfaces**
 - controls, Silverlight, 452-453
 - WPF
 - interactive controls, 395-398
 - video, 392-394
- 32-bit environments, applications, running in, 591-592**
- 64-bit environments, applications, running in, 591-592**

A

- abstract base classes**
 - instantiation, preventing, 23-24
 - interfaces, compared, 24-25
- access, arrays, accessing, 486-487**
- “access denied” errors, 190**
- accessibility modifiers, 8**
- Add Service Reference dialog box, 226**
- adding constructors, 11-12**
- administration privileges, requesting, UAC (User Access Control), 578-581**
- advanced text searches, regular expressions, 132**
- AJAX (Asynchronous JavaScript and XML), 423**
 - pages, creating, 423-425
- AJAX Demo—Default.aspx listing (19.11), 423-424**
- AJAX Demo—Default.aspx.cs listing (19.12), 424**
- aliases, namespaces, 51-52**
- allocating unchanged memory, 488-489**
- AllWidgetsView.xaml listing (25.1), 557-558**
- angle brackets, generics, 141**
- animating WPF element properties, 388-389**
- anonymous methods**
 - delegates, assigning to, 288
 - event handlers, using as, 288-290
 - lambda expression syntax, 290
- anonymous objects, LINQ, 466**
- anonymous types, creating, 22-23**
- anti-aliasing, 348-349**
- App.xaml listing (27.7), 611**

App.xaml.cs listing (27.8), 612-614**appending newline characters, strings, 120****application configuration values, Windows Forms, 314-316****application data, saving with restricted permissions, 198-200****application state, maintaining, ASP.NET, 429-430****ApplicationData folder, 194****applications**

32-bit environments, running in, 591-592

64-bit environments, running in, 591-592

asynchronous programming model, 515-516

command functionality, defining, 547-548, 551

command interface, defining, 545-546

custom attributes, adding, 521-522

deploying, ClickOnce, 572-573

events, writing to event logs, 581-583

history buffer, defining, 545-546

localization, 562-563

ASP.NET application, 564-565

Silverlight application, 570-572

Windows Forms application,

563-564

WPF application, 565-569

memory usage, measuring, 474-475

multiple database servers, working with, 242, 245

nonrectangular windows, creating, 598-601

notification icons, creating, 602-605

OS view, obtaining, 474-475

patterns, 530

Model-View-ViewModel pattern, 552-562

observer pattern, 536-539

plug-in architecture, implementing, 525-528

power state information, retrieving, 595

RSS content, parsing, 216, 219

screen locations, remembering, 543-544

screen savers, creating, 605-614

single instance, limiting, 505-506

sound files, playing, 619-620

splash screens, displaying, 614-619

starting, elevated privileges, 578-581

system configuration changes, responding to, 593

TextTokenizer, 515-516

undo commands, implementing, 545-552

web browsers

embedding, 214-216

running out of, 453-454

Windows services

creating, 585-588

managing, 584

WinForms applications, WPF, 398-400

architectures, plug-in architectures, implementing, 525-528**arrays**

access, speeding up, 486-487

declaring, 50

jagged arrays, 51

multidimensional arrays, creating, 50-51

objects, creating, 140

rectangular arrays, creating, 50-51

strings, splitting into, 121-122

values, reversing, 166-167

AsOrdered() method, 472**ASP.NET**

AJAX pages, creating, 423-425

application state, maintaining, 429-430

controls, binding data to, 256-257

data validation, 425-429

debugging information, viewing, 402-403

GridView, data binding, 412-414

master pages, 409-411

MVC (Model-View-Controller), 436-441

application creation, 436

controller creation, 437

model creation, 436

new records creation, 438-440

record editing, 439, 441

Views creation, 437-438

session state

restoring, 434-436

storing, 433-434

trace information, viewing, 402-403

UI state, maintaining, 430

UIs, creating, 418-422

user controls, creating, 414-417

user data, maintaining, 431-433

- user logins, authentication, 406-409
- users, redirecting to another page, 405-406
- web browser capabilities, determining, 404
- web sites, adding menus, 411-412

ASP.NET applications

- localization, 564-565
- unhandled exceptions, catching, 75

AsParallel() method, 472

assemblies

- plug-in assemblies, creating, 525-526
- shared assemblies, creating, 525
- types, enumerating, 520-521

Asynchronous Javascript and XML

(AJAX). *See* AJAX. *See* Asynchronous JavaScript and XML

asynchronous programming model, 515-516

Asynchronous Web Downloader listing (12.3), 211-213

- asynchronously calling methods, 496-497
- asynchronously downloading web content, 210-213

- auto-implemented properties, 10

- availability, database connections, 258

B

- banker's rounding, 93

- base class constructors, calling, 15

base classes, 24

- abstract base classes
 - instantiation prevention, 23-24
 - interfaces, 24-25
- constraining, 147
- methods, overriding, 16-17
- non-virtual methods, overriding, 17-19
- non-virtual properties, overriding, 17-19
- properties, overriding, 16-17

Base-64 encoding, 122-124

- BaseForm.cs listing (16.1), 304-306

- bases, numbers, converting, 87-89

- BigInteger class, 79-80

- binary data, strings, converting to, 122-124

- binary files, creating, 179

- binding data, controls, 250-257

- Bing.cs listing (21.1), 469-470

- bitmaps, pixels, accessing directly, 347-348

- bits, memory, locking, 348

- BookDetail.aspx listing (19.9), 417

- BookDetail.aspx.cs listing (19.10), 417-418
- BookEntryControl.ascx listing (19.7), 414-415

- BookEntryControl.ascx.cs listing (19.8), 415-417

- BookList.aspx listing (19.6), 412-413

- BooksApp—MasterPage.master, 410

- BookTransform.xslt listing (14.1), 274-275

- bound data, WPF, displaying, 385-386

- browser capabilities, determining, 404

- browsers, applications, running out of, 453-454

- brushes, creating, 339-341

- buffers, off-screen buffers, drawing to, 346-347

bytes

- numbers, converting to, 89-90

- strings

- converting to, 110-111

- translating to, 111, 114-115

C

- C functions, calling, C#, 589-590

- caches, garbage collection, creating, 482-485

calling

- C functions, C#, 589-590

- functions, timers, 313-314

- methods, asynchronously, 496-497

- multiple methods, delegates, 281-282

- native Windows functions, P/Invoke, 588-589

- captures, multiscreen captures, taking, 352-354

- capturing webcams, Silverlight, 455-457

- case, localized strings, changing, 116

catching

- exceptions, 64

- multiple exceptions, 65

- unhandled exceptions, 72-75

- circles, points, determining, 355-356

- classes. *See also* types

- base class constructors, calling, 15

- base classes, 24

- constraining, 147

- overriding methods, 16-17

- overriding properties, 16-17

- BigInteger, 79-80

- changes, notifications, 38

- collection classes, picking correctly, 156-157
- CommonOpenFileDialog, 594
- creating, 8-9, 28
- deriving from, 14-15
- dynamically instantiated classes, invoking methods on, 523-524
- exception classes, creating, 70-72
- formatting
 - ICustomFormatter, 31-32
 - StringBuilder, 31-32
 - ToString() method, 28-31
- generic classes, creating, 143
- hashable classes, creating, 34
- inheritance, preventing, 41-42
- instantiating, 523
- interface classes, constraining, 147
- Math, 94-95
- metadata, attaching, 521-522
- MFC (Microsoft Foundation Classes), 296
- OpenFileDialog, 594
- Parallel, 492-495
- proxy classes, 225-226
- String, 121
- System.Numerics.Complex, 80
- Vertex3d, 9
- XmlDocument, 268
- XmlTextReader, 269
- ClickOnce, applications, deploying, 572-573**
- clients**
 - changes, notifications, 38
 - dynamic clients, implementing, 235-236
 - TCP/IP clients, creating, 204-208
- Clipboard, Windows Forms, 323-327**
- Clipboard.SetText() method, 323**
- closing files, 179**
- clutres (.NET), 562-563**
- code**
 - obsolete code, marking, 531
 - profiling, stopwatch, 530-531
 - reflection, 520
 - instantiation, 523
 - reuse, multiple constructors, 14
- code contracts, enforcing, 58-60**
- code listings**
 - 7.1 (EncodeBase64Bad), 123
 - 7.2 (Reverse Words in a String), 124-125
 - 7.3 (Natural Sorting), 126-130
 - 10.1 (PriorityQueue.cs), 169-173
 - 11.1 (CompressFile.cs), 181-183
 - 11.2 (Searching for a File or Directory), 188-190
 - 12.1 (TCP Server), 205-206
 - 14.1 (BookTransform.xslt), 274-275
 - 16.1 (BaseForm.cs), 304-306
 - 16.2 (InheritedForm.cs), 306-308
 - 18.1 (ImageInfoViewModel.cs), 379-380
 - 18.2 (Window1.xaml.cs), 381-383
 - 19.1 (LoginForm.aspx), 407
 - 19.2 (LoginForm.aspx.cs), 407-408
 - 19.3 (Default.aspx), 409
 - 19.5 (Default.aspx), 411
 - 19.6 (BookList.aspx), 412-413
 - 19.7 (BookEntryControl.ascx), 414-415
 - 19.8 (BookEntryControl.ascx.cs), 415-417
 - 19.9 (BookDetail.aspx), 417
 - 19.10 (BookDetail.aspx.cs), 417-418
 - 19.11 (AJAX Demo—Default.aspx), 423-424
 - 19.12 (AJAX Demo—Default.aspx.cs), 424
 - 19.13 (Validation Demo—Default.aspx), 425-427
 - 19.14 (Validation Demo—Default.aspx.cs), 427-428
 - 19.15 (Session State Demo—Default.aspx), 431-432
 - 19.16 (Session State Demo—Default.aspx.cs), 432-433
 - 20.1 (MainPage.xaml), 445-448
 - 20.2 (MainPage.xaml.cs), 446-448
 - 20.3 (PlayDownloadProgressControl.xaml), 449
 - 20.4 (PlayDownloadProgressControl.xaml.cs), 449-450
 - 20.5 (MainPage.xaml), 455
 - 20.6 (MainPage.xaml.cs), 456-457
 - 21.1 (Bing.cs), 469-470
 - 21.2 (Program.cs), 471
 - 25.1 (AllWidgetsView.xaml), 557-558
 - 25.2 (WidgetGraphicView.xaml), 558
 - 25.3 (Mainwindow.xaml), 561-562
 - 26.1 (MyCDll.h), 589
 - 26.2 (MyCDll.cpp), 589
 - 26.3 (MyCDll.def), 590
 - 27.1 (Window1.xaml), 600-601
 - 27.2 (Window1.xaml.cs), 601

- 27.3 (OptionsWindow.xaml), 606
- 27.4 (OptionsWindow.xaml.cs), 606-607
- 27.5 (ScreenSaverWindow.xaml), 607
- 27.6 (ScreenSaverWindow.xaml.cs), 607-611
- 27.7 (App.xaml), 611
- 27.8 (App.xaml.cs), 612-614
- 27.9 (SplashScreen.xaml), 616-619
- collapsing controls, WPF, 375-376**
- collection classes, picking correctly, 156-157**
- collection items, concatenating, strings, 119-120**
- collections**
 - arrays, reversing, 166-167
 - concurrency-aware collections, 157
 - custom collections, creating, 159-163
 - custom iterators, creating, 163-166
 - data binding, WPF, 385
 - elements
 - counting, 168
 - obtaining, 168
 - shuffling, 620
 - filtering, LINQ, 464
 - generic collections, 156
 - initializing, 157-158
 - interfaces, 159
 - linked lists, reversing, 167
 - priority queues, implementing, 169
 - querying, LINQ, 462-463
 - trie structure, creating, 173-176
- color definitions, graphics, 330**
- color picker, Windows Forms, 330-331**
- colors, converting, 331-335**
- COM interop, dynamic typing, simplifying, 49**
- Combining streams, 181-183**
- command functionality, defining, 547-548, 551**
- command interface, defining, 545-546**
- command objects, undo commands, implementing, 545-552**
- commands (WPF)**
 - custom commands, 371-373
 - enabling/disabling, 374
 - standard commands, 370-371
- CommonOpenFileDialog class, 594**
- complex numbers, formatting, 80-82**
- ComplexCriteria() method, 472**
- CompressFile.cs listing (11.1), 181-183**
- compression, files, 181-183**
- concatenating**
 - collection items into strings, 119-120
 - StringBuilder, 117-119
- concurrency-aware collections, 157**
- conditional operator, 52-53**
- configuration, Windows Forms, 314-316**
- connections, databases, 240-242, 245**
 - availability, 258
- console programs, unhandled exceptions, catching, 73**
- const fields, 13**
- constants, enumeration constants, duplicate values, 101**
- constraining, generic types, 146-149**
- constraints, methods, adding, 58-60**
- construction, properties, initialization, 12**
- constructors**
 - adding, 11-12
 - base class constructors, calling, 15
 - multiple constructors, code reuse, 14
- Contracts class, methods, constraints, 58-60**
- Contravariance, delegates, 291**
- controls**
 - 3D surfaces, Silverlight, 452-453
 - data, binding to, 250-257
 - DataGridView, 250-254
 - interactive controls, 3D surfaces, 395-398
 - ToolStrip, 297
 - user controls
 - creating, 414-417
 - Windows Forms, 308-313
 - windows, positioning, 367
 - WPF
 - appearance/functionality, 377
 - binding properties, 379-383
 - designing, 386-387
 - expanding/collapsing, 375-376
- conversion operators, implementing, 40-41**
- Convert.ToBase64String (), 122**
- converting**
 - binary data to strings, 122-124
 - bytes to strings, 110-111
 - numbers
 - bytes, 89-90
 - number bases, 87-89
 - strings
 - flags, 104
 - to bytes, 110-111

- to enumerations, 103-104
- to numbers, 86-87
- types, 40-41
- cookies, session state, restoring, 434-436**
- counting 1 bits, 92**
- CPUs, information, obtaining, 576-578**
- cryptographically secure random numbers, 97**
- cultures, numbers, formatting for, 82-83**
- current operating system, version information, obtaining, 576**
- cursor**
 - mouse cursor, distance, 354-355
 - wait cursors, resetting, 327-328
- custom attributes, applications, adding, 521-522**
- custom collections, creating, 159-160, 163**
- custom commands, WPF, 371-373**
- custom encoding schemes, strings, 111, 114-115**
- custom formatting, ToString() method, 29**
- custom iterators, collections, creating, 163-166**
- Custom web browser listing (12.4), 215-216**
- cut and paste operations, Windows Forms, 323-327**

D

- data**
 - exchanging, threads, 499-500
 - multiple threads, protecting, 500-502
 - protecting, multiple processes, 504-505
 - storing application-wide, 429-430
- data binding**
 - GridView control, 412-414
 - WPF
 - collections, 385
 - value conversions, 383-385
 - value formatting, 383
- data structures, multiple threads, 495**
- data types, forms, cutting and pasting, 323**
- database tables**
 - data
 - deleting, 246-247
 - inserting, 245-246
 - stored procedures, running, 247-248
- databases**
 - connecting to, 240-242, 245
 - connections, availability, 258
 - controls, binding data to, 250-257
 - creating, Visual Studio, 238-239
 - data, transforming to, 273-276
 - multiple tables, joins, 465-466
 - MySQL databases, connecting to, 241-242
 - objects, mapping data to, 259-260
 - tables
 - deleting data, 246-247
 - displaying data, 250-257
 - inserting data, 245-246
 - transactions, 248-250
 - updating, DataSet, 252-254
- DataGridView control, 250-254**
- DataSet**
 - controls, binding data to, 250-257
 - databases, updating, 252-254
- dates, validating, 136**
- dead code, marking, 531**
- debugging information, viewing, ASP.NET, 402-403**
- Decimal floating point types, 78**
- declaring**
 - delegates, 280
 - enumerations, 100-102
 - flags as enumerations, 101-102
 - objects, 50
 - variables, 46-47
- default constructors, types, constraining to, 148**
- default parameters, methods, calling, 55-56**
- Default.aspx listing (19.3), 409**
- Default.aspx listing (19.5), 411**
- deferring**
 - evaluations, values, 57-58
 - type checking, runtime, 47-49
- defining**
 - fields, 9-10
 - methods, 9-10
 - properties, 9-10
 - static members, 10-11
- degrees, radians, converting to, 93**
- delegates**
 - anonymous methods, assigning to, 288
 - contravariance, 291
 - declaring, 280
 - generic delegates, 145-146
 - multiple methods, calling to, 281-282
- deleting files, 180**

deploying applications, ClickOnce, 572-573
 diagonally drawing text, 344
 dialog boxes
 Add Service Reference, 226
 New Silverlight Application, 445
 directories
 browsing for, 187
 enumerating, 186-187
 existence, confirming, 185
 searching for, 188-190
 directory names, filenames, combining, 190-191
 disabling commands, WPF, 374
 discoverable hosts, implementing, 233-234
 displaying splash screens, 614
 Windows Forms, 614-616
 WPF, 616-619
 Dispose pattern, finalization, 479-482
 dispose pattern, managed resources, cleaning up, 477-482
 Dispose pattern, Windows Communication Framework, 479
 DLLs (dynamic link libraries), C functions, calling, 589-590
 DLR (Dynamic Language Runtime), 49
 documents
 printing, Silverlight, 457
 XML documents, validating, 270-271
 Double floating point types, 78
 download progress bars, video, Silverlight, 449-450
 downloading, web content, HTTP, 209-213
 drawing shapes, 335-337
 drives, enumerating, 185-186
 dynamic clients, implementing, 235-236
 dynamic keyword, 47-49
 Dynamic Language Runtime (DLR), 49
 dynamic typing, COM interop, simplifying, 49
 dynamically disabling, menu items, Windows Forms, 300
 dynamically instantiated classes, methods, invoking on, 523-524
 dynamically producing, RSS feeds, IIS (Internet Information Services), 220-222
 dynamically sized array of objects, creating, 140

E

element properties, WPF, animating, 388-389
 elements, collections
 counting, 168
 obtaining, 168
 shuffling, 620
 ellipse, points, determining, 356-357
 email, SMTP (Simple Mail Transport Protocol), sending via, 208-209
 email addresses, matching, 136
 embedding, web browsers, applications, 214-216
 empty strings, detecting, 117
 enabling commands, WPF, 374
 EncodeBase64Bad listing (7.1), 123
 encoding schemes, strings, 111, 114-115
 Encoding.GetString() method, 110
 enforcing code contracts, 58-60
 Entity Framework
 database objects, mapping data to, 259-260
 entities
 creating, 260
 deleting, 260
 listing, 259
 looking up, 260
 querying, LINQ, 467-469
 Enum values, metadata, attaching to, 104-106
 Enum.GetValues() method, 103
 enumerating
 directories, 186-187
 drives, 185-186
 files, 186-187
 enumerations, 100, 106
 declaring, 100-102
 external values, matching, 106
 flags, 107
 declaring as, 101-102
 integers, converting to, 102
 naming, 107
 None values, defining, 107
 strings, converting to, 103-104
 validity, determining, 103
 values, listing, 103
 equality, types, determining, 32-33
 Equals() method, objects, equality, 32-33
 evaluation, values, deferring, 57-58

event brokers, 540-543

event handlers, anonymous methods, using as, 288-290

event logs

events, writing to, 581-583

reading from, 582

events

event brokers, 540-543

event logs, writing to, 581-583

metadata, attaching, 521-522

multiple events, combining into one, 532-536

publishing, 283

signaling, threads, 509, 512

subscribing to, 282-283

WPF, responding to, 376-377

exceptions

catching, 64

multiple exceptions, 65

unhandled exceptions, 72-75

classes, creating, 70-72

handling, 76

information, extracting, 68-70

intercepting, 67

rethrowing, 66-67

throwing, 64

exchanging data, threads, 499-500

existingType.MyNewMethod() method,

types, adding methods to, 54-55

Exists() method, 185

expanding controls, WPF, 375-376

explicit conversions, types, 40-41

explicit values, enumerations,

declaring, 100

expressions, regular expressions, 132

advanced text searches, 132

extracting groups of text, 132-133

improving, 137

replacing text, 133-134

validating user input, 134-136

eXtensible Markup Language (XML). See

XML (eXtensible Markup Language)

extension methods, metadata, attaching to

Enum values, 104-106

F

fields

const, 13

defining, 9-10

metadata, attaching, 521-522

read only, 13

file dialogs, 594

filenames

directory names, combining, 190-191

temporary filenames, creating, 192

files

accessing, 590-591

closing, 179

compressing, 181-183

deleting, 180

enumerating, 186-187

existence, confirming, 185

FTP sites, uploading to, 213-214

memory-mapped files, 590-591

paths, manipulating, 190-191

searching for, 188-190

security information, retrieving, 183-184

sizes, retrieving, 183

text files, creating, 178-179

XML files

reading, 268-270

validating, 270-271

filtering object collections, LINQ, 464

finalization

Dispose pattern, 479-482

unmanaged resources, cleaning up, 475-477

flags

enumerations, 107

declaring, 101-102

strings, converting to, 104

floating-point types, choosing, 78

folders

paths, retrieving, 194

users, allowing access, 187

forcing garbage collection, 482

format strings, 84-85

formatting

complex numbers, 80-82

numbers, strings, 82-85

types

ICustomFormatter, 31-32

StringBuilder, 31-32

ToString() method, 28-31

forms

configuration, 314-316

data types, cutting and pasting, 323

horizontal tilt wheel, 319-323

images, cutting and pasting, 323

inheritance, 304-308

menu bars, adding, 297-299

- menu items, dynamically
 - disabling, 300
- modal forms, creating, 296
- modeless forms, creating, 296
- split window interfaces, creating, 302-303
- status bars, adding, 300
- text, cutting and pasting, 323
- timers, 313-314
- toolbars, adding, 301-302
- user controls, creating, 308-313
- user login, authentication, 406-409
- user-defined objects, cutting and pasting, 325-327
- wait cursors, resetting, 327-328

FTP sites, files, uploading to, 213-214

functions, calling, timers, 313-314

FXCop, 626-627

G

garbage collection

- caches, creating, 482-485
- forcing, 482

GDI (Graphics Device Interface), 330

GDI+, 330

- anti-aliasing, 348-349
- bitmap pixels, accessing directly, 347-348
- brushes, creating, 339-341
- color picker, 330-331
- colors, converting, 331-335
- flicker-free drawing, 349-350
- graphics
 - color definitions, 330
 - resizing, 350-351
 - thumbnails, 351-352
- images, drawing, 344-345
- mouse cursor, distance, 354-355
- multiscreen captures, taking, 352-354
- off-screen buffers, drawing to, 346-347
- pens, creating, 337-339
- points
 - circles, 355-356
 - ellipse, 356-357
 - mouse cursor, 354-355
 - rectangles, 355
- rectangles, intersection, 357
- shapes, drawing, 335-337
- text, drawing, 344

- transformations, 341
 - rotation, 342
 - scaling, 343
 - shearing, 343
 - translations, 342
- transparent images, drawing, 345

generating

- GUIDs (globally unique IDs), 97-98
- random numbers, 96-97

generic classes, creating, 143

generic collections, 156

- methods, passing to, 149-150

generics, 140

- constraining, 146-149
- generic classes, creating, 143
- generic collections, 156
 - passing to methods, 149-150
- generic delegates, creating, 145-146
- generic interfaces, creating, 142
- generic list, creating, 140
- generic methods, creating, 141-142
- generic types, constraining, 146-149
- multiple generic types, creating, 146

GetBytes() method, 110

GetHashCode() method, hashable types, creating, 34

GetPixel() method, 347

GetTempFileName() method, 192

GetTotalMemory() method, 474

graphics

- color definitions, 330
- resizing, 350-351
- text, drawing, 344
- thumbnails, creating, 351-352
- transformations, 341
 - rotation, 342
 - scaling, 343
 - shearing, 343
 - translations, 342

Graphics Device Interface (GDI), 330

GridView control, data, binding to, 412-414

group digits, 84

groups of text, extracting, regular expressions, 132-133

GUIDs (globally unique IDs), generating, 97-98

H

handling exceptions, 76

Hanselman, Scott, 631

hard drives, enumerating, 185-186

hardware information, obtaining, 576-578
HasFlag() method, 102
 hash codes, 34
 hashable types, creating, **GetHashCode()** method, 34
 hexadecimal numbers, printing in, 83
 history buffer, defining, 545-546
 horizontal tilt wheel, Windows Forms, 319-323
 hostnames, current machines, obtaining, 202
 hosts
 availability, detecting, 203
 discoverable hosts, implementing, 233-234
 HSV color format, RGB color format, converting between, 331-335
 HTML tags, stripping, 214
 HTTP, web content, downloading via, 209-213

I

IComparer, covering, 150-151
 icons, notification icons, creating, 602-605
ICustomFormatter, types, formatting, 31-32
IEnumerable, 50
 converting, 149
 IIS (Internet Information Services), RSS feeds, producing dynamically, 220-222
 ImageInfoViewModel.cs listing (18.1), 379-380
 images. *See also* graphics
 drawing, 344-345
 forms, cutting and pasting, 323
 resizing, 350-351
 thumbnails, creating, 351-352
 transparent images, drawing, 345
 implicit conversions, types, 41
 implicit typing, 46-47
 indexes, types, 36-37
 inference, types, 46-47
 information, exceptions, extracting, 68-70
 inheritance, forms, 304-308
 inheritances, classes, preventing, 41-42
 InheritedForm.cs listing (16.2), 306-308
 initialization
 collections, 157-158
 properties at construction, 12
 static data, 12
INotifyPropertyChanged interface, 38
 installation, **NUit**, 625

instantiation, abstract base classes, preventing, 23-24
 integers
 determining, 79, 82, 91-93, 96-97
 enumerations, converting to, 102
 large integers, **UInt64**, 79-80
 interactive controls, 3D surfaces, **WPF**, 395-398
 intercepting exceptions, 67
 interface classes, constraining, 147
 interfaces, 24
 abstract base classes, compared, 24-25
 collections, 159
 contracts, implementing on, 60
 creating, 19
 generic interfaces, creating, 142
 implementing, 19-21
 split window interfaces, creating, 302-303
 interlocked methods, locks, compared, 503
 Internal accessibility modifier, 8
 Internet, communication over, **WCF**, 231-232
IntersectsWith() method, 357
 IP addresses
 current machines, obtaining, 202
 hostnames, translating to, 202
ISerializable() interface, 196
IsPrime() method, 92
 iterators, collections, creating for, 163-166

J-K

jagged arrays, 51
 joins, multiple tables, **LINQ**, 465-466
 keywords
 dynamic, 47-49
 object, 49
 var, 22-23, 47

L

labels, type parameters, 146
 lambda expression syntax, anonymous methods, 290
 Language Integrated Query (**LINQ**). *See* **LINQ (Language Integrated Query)**
 layout method, **WPF**, choosing, 367
 leading zeros, printing, 84
 libraries, Windows 7, accessing, 594

limiting applications, single instance, 505-506

linked lists, reversing, 167

LINQ (Language Integrated Query), 462

anonymous objects, 466

Bing, 469-471

Entity Framework, querying, 467-469

multiple tables, joins, 465-466

object collections

filtering, 464

obtaining portions, 465

querying, 462-463

PLINQ (Parallel LINQ), 472

query results, ordering, 463

SQL, compared, 462

web services, querying, 469-471

XML, generating, 467

XML documents, querying, 466-467

LINQPad, 630-631

listing values, enumerations, 103

listings

7.1 (EncodeBase64Bad), 123

7.2 (Reverse Words in a String), 124-125

7.3 (Natural Sorting), 126-130

10.1 (PriorityQueue.cs), 169-173

11.1 (CompressFile.cs), 181-183

11.2 (Searching for a File or Directory), 188-190

12.1 (TCP Server), 205-206

14.1 (BookTransform.xslt), 274-275

16.1 (BaseForm.cs), 304-306

16.2 (InheritedForm.cs), 306-308

18.1 (ImageInfoViewModel.cs), 379-380

18.2 (Window1.xaml.cs), 381-383

19.1 (LoginForm.aspx), 407

19.2 (LoginForm.aspx.cs), 407-408

19.3 (Default.aspx), 409

19.5 (Default.aspx), 411

19.6 (BookList.aspx), 412-413

19.7 (BookEntryControl.ascx), 414-415

19.8 (BookEntryControl.ascx.cs), 415-417

19.9 (BookDetail.aspx), 417

19.10 (BookDetail.aspx.cs), 417-418

19.11 (AJAX Demo—Default.aspx), 423-424

19.12 (AJAX Demo—Default.aspx.cs), 424

19.13 (Validation Demo—Default.aspx), 425-427

19.14 (Validation Demo—Default.aspx.cs), 427-428

19.15 (Session State Demo—Default.aspx), 431-432

19.16 (Session State Demo—Default.aspx.cs), 432-433

20.1 (MainPage.xaml), 445-448

20.2 (MainPage.xaml.cs), 446-448

20.3 (PlayDownloadProgressControl.xaml), 449

20.4 (PlayDownloadProgressControl.xaml.cs), 449-450

20.5 (MainPage.xaml), 455

20.6 (MainPage.xaml.cs), 456-457

21.1 (Bing.cs), 469-470

21.2 (Program.cs), 471

25.1 (AllWidgetsView.xaml), 557-558

25.2 (WidgetGraphicView.xaml), 558

25.3 (Mainwindow.xaml), 561-562

26.1 (MyCDll.h), 589

26.2 (MyCDll.cpp), 589

26.3 (MyCDll.def), 590

27.1 (Window1.xaml), 600-601

27.2 (Window1.xaml.cs), 601

27.3 (OptionsWindow.xaml), 606

27.4 (OptionsWindow.xaml.cs), 606-607

27.5 (ScreenSaverWindow.xaml), 607

27.6 (ScreenSaverWindow.xaml.cs), 607-611

27.7 (App.xaml), 611

27.8 (App.xaml.cs), 612-614

27.9 (SplashScreen.xaml), 616-619

ListView (Windows Forms), virtual mode, 317-318

loading plugins, 526-528

LocalApplicationData folder, 194

localization, 562-563

ASP.NET application, 564-565

resource files, 568-569

Silverlight application, 570-572

Windows Forms application, 563-564

WPF application, 565-569

XAML localization, 566-568

localized strings

case, changing, 116

comparing, 115-116

locking bits, memory, 348

locks, 502

- interlocked methods, compared, 503
- multiple threads, 502
- reader-writer locks, 513-514

LoginForm.aspx listing (19.1), 407

LoginForm.aspx.cs listing (19.2), 407-408

M

MailMessage class, 209

MainPage.xaml listing (20.1), 445-448

MainPage.xaml listing (20.5), 455

MainPage.xaml.cs listing (20.2), 446-448

MainPage.xaml.cs listing (20.6), 456-457

Mainwindow.xaml listing (25.3), 561-562

managed resources, cleaning up, dispose pattern, 477-482

mapping data, database objects, 259-260

marking obsolete code, 531

master pages, ASP.NET, 409-411

Math class, numbers, rounding, 94-95

measuring memory usage, 474-475

memory

- bits, locking, 348
- fixed memory, 488
- objects, directly accessing, 485-486
- preventing being moved, 487-488
- unchanged memory, allocating, 488-489

memory streams, serializing, 198

memory usage, measuring, 474-475

memory-mapped files, 590-591

menu bars

- windows, adding to, 367-368
- Windows Forms, adding, 297-299

menu items, Windows Forms, disabling dynamically, 300

menus, websites, adding, 411-412

metadata

- Enum values, attaching to, 104-106
- method arguments, attaching, 521-522

method arguments, metadata, attaching, 521-522

methods

- anonymous methods
 - as event handlers, 288-290
 - assigning to delegates, 288
 - lambda expression syntax, 290
- AsOrdered(), 472
- AsParallel(), 472

calling

- default parameters, 55-56
- named parameters, 56
- specific intervals, 512

calling asynchronously, 496-497

Clipboard.SetText(), 323

ComplexCriteria(), 472

constraints, adding to, 58-60

defining, 9-10

delegates, calling multiple to, 281-282

dynamically instantiated classes, invoking on, 523-524

Encoding.GetString(), 110

Enum.GetValues(), 103

Equals(), 32-33

existingType.MyNewMethod(), 54-55

Exists(), 185

extension methods, 104-106

generic collections, passing to, 149-150

generic methods, creating, 141-142

GetBytes(), 110

GetHashCode(), 34

GetPixel(), 347

GetTempFileName(), 192

GetTotalMemory(), 474

HasFlag(), 102

IntersectsWith(), 357

IsPrime(), 92

metadata, attaching, 521-522

Monitor.Enter(), 501

Monitor.Exit(), 501

non-virtual methods, overriding, 17-19

Object.Equals(), 32

OrderBy(), 472

overriding, base classes, 16-17

Parse(), 97

ParseExact(), 97

runtime, choosing, 280-282

SetPixel(), 347

String.Concat(), 119

String.IsNullOrEmpty(), 117

String.IsNullOrEmptyOrWhitespace(), 117

ToString(), 28-31, 79, 82, 104, 385

TryParse(), 97

TryParseExact(), 97

types, adding to, 54-55

MFC (Microsoft Foundation Classes), 296

modal forms, creating, 296

Model-View-ViewModel pattern, WPF, 552-562
 defining model, 553-554
 defining view, 557-558
 defining ViewModel, 555-556

modeless forms, creating, 296

modifiers, accessibility modifiers, 8

Monitor.Enter() method, 501

Monitor.Exit() method, 501

monitoring system changes, 192-194

mouse cursor, distance, 354-355

multidimensional arrays, creating, 50-51

multiple constructors, code reuse, 14

multiple events, single event, combining into, 532-536

multiple exceptions, catching, 65

multiple generic types, creating, 146

multiple interfaces, implementing, 20-21

multiple processes, data, protecting, 504-505

multiple tables, joins, LINQ, 465-466

multiple threads
 data, protecting, 500-502
 data structures, 495
 locks, 502

multiples, numbers, rounding to, 94-95

multiscreen captures, taking, 352-354

multithreaded timers, 512

mutexes, naming, 505

MVC (Model-View-Controller), 436-441
 application creation, 436
 controller creation, 437
 model creation, 436
 new records creation, 438-440
 records, editing, 439-441
 Views creation, 437-438

My Documents, paths, retrieving, 194

MyCDII.cpp listing (26.2), 589

MyCDII.def listing (26.3), 590

MyCDII.h listing (26.1), 589

MySQL databases, connecting to, 241-242

N

named parameters, methods, calling, 56

namespaces
 aliasing, 51-52
 System.Text, 110

naming
 enumerations, 107
 mutexes, 505

native Windows functions, calling, P/Invoke, 588-589

Natural Sorting listing (7.3), 126-130

naturally sorting, number strings, 125, 128-130

NDepend, 626

.NET
 cultures, 562-563
 MSDN documentation, 3
 objects, storing in binary form, 194-197
 printing, 358-363
 Win32 API functions, calling, 588-589

network cards, information, obtaining, 204

networks, availability, detecting, 203

New Silverlight Application dialog box, 445

newline characters, strings, appending to, 120

non-virtual methods and properties, overriding, 17-19

None values, enumerations, defining, 107

nonrectangular windows, creating, 598-601

notification icons, creating, 602-605

notifications, changes, 38

null values, checking for, 53

null-coalescing operator, 53

nulls, value types, assigning to, 42

number format strings, 85

number strings, sorting naturally, 125, 128-130

numbers
 1 bits, counting, 92
 bytes, converting to, 89-90
 complex numbers, formatting, 80-82
 degrees, converting, 93
 enumerations, 100, 106
 converting to integers, 102
 declaring, 100-102
 external values, 106
 flags, 107
 naming, 107
 None values, 107
 strings, 103-104
 validity, 103
 values, 103

floating-point types, 78

group digits, 84

GUIDs (globally unique IDs), generating, 97-98

hexadecimal, printing in, 83

- integers
 - converting to enumerations, 102
 - determining, 79, 82, 91-93, 96-97
 - large integers, 79-80
- leading zeros, printing, 84
- number bases, converting, 87-89
- prime numbers, determining, 92
- pseudorandom numbers, 96
- radians, converting, 93
- random numbers, generating, 96-97
- rounding, 94-95
- strings
 - converting, 86-87
 - formatting in, 82-85

numerical indexes, types, 36

NUnit, 623-625

O

object collections

- filtering, LINQ, 464
- querying, LINQ, 462-463

object keyword, 49

Object.Equals() method, 32

objects

- arrays
 - declaring, 50
 - jagged arrays, 51
 - multidimensional arrays, 50-51
 - rectangular arrays, 50-51
- collections
 - concurrency-aware collections, 157
 - counting elements, 168
 - custom collection creation, 159-160, 163
 - custom iterator creation, 163-166
 - initializing, 157-158
 - interfaces, 159
 - obtaining elements, 168
 - picking correctly, 156-157
 - priority queues, 169
 - reversing arrays, 166-167
 - reversing linked lists, 167
 - trie structure, 173-176

databases, mapping data to, 259-260

enumerations, 100, 106

- converting to integers, 102

- declaring, 100-102

- external values, 106

- flags, 107

- naming, 107

- None values, 107

- strings, 103-104

- validity, 103

- values, 103

- equality, determining, 32-33

- memory, directly accessing, 485-486

- serializing, 194-197

- sortable objects, creating, 34-35

- user-defined objects, forms, 325-327

- XML, serialization, 262-266

observer pattern, implementing, 536-539

obsolete code, marking, 531

off-screen buffers, drawing to, 346-347

OpenFileDialog class, 594

operating systems, current operating system, version information, 576

operators

- != operator, implementing, 39-40

- + operator, implementing, 39

- == operator, implementing, 39-40

- conditional operators, 52-53

- conversion operators, implementing, 40-41

- null-coalescing operator, 53

- overloading, 39-40

OptionsWindow.xaml listing (27.3), 606

OptionsWindow.xaml.cs listing (27.4), 606-607

OrderBy() method, 472

ordering query results, LINQ, 463

overloading operators, 39-40

overriding

- non-virtual methods, base classes, 17-19

- non-virtual properties, 17-19

- ToString() method, 29

P

P/Invoke, native Windows functions, calling, 588-589

Parallel class

- data, processing in, 492-494

- tasks, running in, 494-495

parameters

- default values, specifying, 55-56

- named parameters, called methods, 56

- types, labels, 146

Parse (), 86

parse hexadecimal number strings, converting, 87

Parse() method, 97

ParseExact() method, 97

paths
files, manipulating, 190-191
user folders, retrieving, 194

patterns (applications), 530
Model-View-ViewModel pattern, 552-562
defining model, 553-554
defining view, 557-558
defining ViewModel, 555-556
observer pattern, implementing, 536-539

pens, creating, 337-339

phone numbers, validating, 135

pinging machines, 203

playback progress bars, video, Silverlight, 449-450

PlayDownloadProgressControl.xaml
listing (20.3), 449

PlayDownloadProgressControl.xaml.cs
listing (20.4), 449-450

playing sound files, 619-620

PLINQ (Parallel LINQ), 472

plugin architecture, implementing, 525-528

plugin assemblies, creating, 525-526

plugins, loading and searching for, 526-528

pointers, using, 485-486

points
circles, determining, 355-356
ellipse, determining, 356-357
mouse cursor, distance, 354-355
rectangles, determining, 355

power state information, retrieving, 595

prefix trees, 173-176

prime numbers, determining, 91-92

Print Preview, 363

printing
.NET, 358-363
documents, Silverlight, 457
numbers
hexidecimals, 83
leading zeros, 84

priority queues, collections, implementing, 169

PriorityQueue.cs listing (10.1), 169-173

Private accessibility modifier, 8

Process Explorer, 628-629

Process Monitor, 628-629

processes, communicating between, 222-229

processing data in Parallel class, 492-494

processors, tasks, splitting among, 492-495

profiling code, stopwatch, 530-531

Program.cs listing (21.2), 471

progress bars, video, Silverlight, 449-450

projects (Silverlight), creating, 444-445

properties
auto-implemented properties, 10
defining, 9-10
initialization at construction, 12
metadata, attaching, 521-522
non-virtual properties, overriding, 17-19
overriding base classes, 16-17

Protected accessibility modifier, 8

Protected internal accessibility modifier, 8

protecting data, multiple threads, 500-502

proxy classes, generating, Visual Studio, 225-226

pseudorandom numbers, 96

Public accessibility modifier, 8

publishing events, 283

Q

query results, ordering, LINQ, 463

querying
Entity Framework, LINQ, 467-469
object collections, LINQ, 462-463
web services, LINQ, 469-471
XML documents
LINQ, 466-467
XPath, 271-272

R

radians, degrees, converting to, 93

random numbers
cryptographically secure random numbers, 97
generating, 96-97

reader-writer locks, 513-514

reading
binary files, 179
files, XML files, 268-270
text files, 178-179

readonly fields, 13

rectangles
intersection, determining, 357
points, determining, 355

rectangular arrays, creating, 50-51

reference types, constraining, 147

references, weak references, 484

reflection, 520

- code, instantiating, 523
- types, discovering, 520-521

Reflector, 622**RegexBuddy, 630****registry**

- accessing, 583-584
- XML configuration files, compared, 584

regular expressions, 132

- advanced text searches, 132
- improving, 137
- text
 - extracting groups, 132-133
 - replacing, 133-134
 - user input, validating, 134-136

rendering 3D geometry, WPF, 389-392**replacing text, regular expressions, 133-134****resetting wait cursor, Windows Forms, 327-328****resizing graphics, 350-351****resource files, localization, 568-569****resources, thread access, limiting, 506-508****restoring session state, cookies, 434-436****restricted permissions, application data, saving, 198-200****rethrowing exceptions, 66-67****retrieving power state information, 595****reuse, code, multiple constructors, 14****Reverse Words in a String listing (7.2), 124-125****reversing**

- linked lists, 167
- values, arrays, 166-167
- words, strings, 124-125

RGB color format, HSV color format,

- converting between, 331-335

rotation, graphics, 342**rounding numbers, 93-95****RoutedEvents, WPF, 377****RSS feeds**

- consuming, 216, 219
- producing dynamically in IIS, 220-222

running

- stored procedures, databases, 247-248
- tasks in Parallel, 494-495

running code, timing, 530-531**runtime**

- methods, choosing, 280-282
- type checking, deferring, 47-49

S**saving application data, restricted permissions, 198-200****scaling graphics, 343****screen locations, applications,**

- remembering, 543-544

screen savers, WPF, creating in, 605-614**ScreenSaverWindow.xaml**

- listing (27.5), 607

ScreenSaverWindow.xaml.cs

- listing (27.6), 607-611

searches

- directories and files, 188-190
- plugins, 526-528

Searching for a File or Directory

- listing (11.2), 188-190

security information, files, retrieving, 183-184**SerializableAttribute (), 195-196****Serialization, objects, XML, 262-266****serializing**

- memory streams, 198
- objects, 194-197

servers

- data validation, ASP.NET, 425-429
- SQL Server, connecting to, 240-241
- TCP/IP servers, creating, 204-208

services, discovering, during runtime, 233-236**session state, ASP.NET, storing and restoring, 433-436****Session State Demo—Default.aspx listing (19.15), 431-432****Session State Demo—Default.aspx.cs listing (19.16), 432-433****SetPixel() method, 347****shapes**

- circles, points, 355-356
- drawing, 335, 337
- ellipse, points, 356-357
- rectangles
 - intersection, 357
 - points, 355

shared assemblies, creating, 525**shared integer primitives, manipulating, 503****shearing graphics, 343****shuffling elements, collections, 620****signaling events, threads, 509-512**

Silverlight, 444

- 3D surfaces, controls, 452-453
- browsers, running applications out of, 453-454
- documents, printing, 457
- projects, creating, 444-445
- UI threads, timer events, 451-452
- versions, 444
- videos
 - playing over web, 445-448
 - progress bar, 449-450
 - webcams, capturing, 455-457

Silverlight applications, localization, 570-572**single event, multiple events, combining into, 532-536****single instance, applications, limiting, 505-506****sizes, files, retrieving, 183****Smart Tags, Visual Studio, 20****SMTP (Simple Mail Transport Protocol), email, sending via, 208-209****social security numbers, validating, 134****sortable types, creating, 34-35****sorting number strings naturally, 125, 128-130****sound files, playing, 619-620****splash screens, displaying, 614**

- Windows Forms, 614-616
- WPF, 616-619

SplashScreen.xaml listing (27.9), 616-619**Split (), 121****split window interfaces, Windows Forms, creating for, 302-303****splitting strings, 121-122****SQL, LINQ, compared, 462****SQL objects, external resources, wrapping, 246****SQL Server, connecting to, 240-241****SqlCommand object, 246****StackOverflow.com, 3****standard commands, WPF, 370-371****statements, values, choosing, 52-53****static constructors, adding, 12****static members, defining, 10-11****status bars**

- windows, adding to, 369
- Windows Forms, adding to, 300

stopwatch, code, profiling, 530-531**stored procedures, databases, running, 247-248****streams, combining, 181-183****String class, 121****string indexes, types, 37****String.Concat() method, 119****String.IsNullOrEmpty() method, 117****String.IsNullOrWhiteSpace() method, 117****StringBuilder, types, formatting, 31-32****StringBuilder (), strings, concatenating, 117-119****strings, 110**

- binary data, converting to, 122-124
- bytes

- converting to, 110-111

- translating to, 111, 114-115

- case, changing, 116

- comparing, 115-116

- concatenating

- collection items, 119-120

- StringBuilder, 117-119

- custom encoding scheme, 111, 114-115

- empty strings, detecting, 117

- enumerations, converting to, 103-104

- flags, converting to, 104

- format strings, 84-85

- newline characters, appending to, 120

- number format strings, 85

- number strings, sorting naturally, 125, 128-130

- numbers

- converting, 86-87

- formatting in, 82-85

- splitting, 121-122

- tokens, reversing, 124-125

- Unicode, 111

- words, reversing, 124-125

stripping HTML of tags, 214**structs, 8**

- creating, 21-22

styles, WPF, triggers, 378**subscriber pattern, implementing, 536-539****subscriptions, events, 282-283****system changes, monitoring, 192-194****system configuration changes, responding to, 593****System.Numerics.Complex class, 80****System.Text namespace, 110**

T**tables (databases), data**

- deleting, 246-247
- displaying, 250-257
- inserting, 245-246

tags, HTML, stripping, 214**tasks**

- processors, splitting among, 492-495
- running in Parallel class, 494-495

TCP Server listing (12.1), 205-206**TCP/IP clients and servers, creating, 204-208****templates, controls, designing, 386-387****temporary filenames, creating, 192****text**

- drawing, 344
- extracting groups, regular expressions, 132-133
- forms, cutting and pasting, 323
- replacing, regular expressions, 133-134

text files, creating, 178-179**text searches, regular expressions, 132****TextTokenizer application, 515-516****thread pools, 497-499****threads, 492**

- creating, 498-499
- culture settings, 562-563
- data, exchanging, 499-500
- multiple threads
 - data protection, 500-502
 - data structures, 495
- resource access, limiting number, 506-508
- signaling events, 509, 512

throwing exceptions, 64

- rethrowing, 66-67

thumbnail graphics, creating, 351-352**timer events, UI threads, Silverlight, 451-452****timers**

- functions, calling, 313-314
- multithreaded timers, 512

timing code, 530-531**tokens, strings, reversing, 124-125****toolbars**

- windows, adding to, 369-370
- Windows Forms, adding to, 301-302

tools

- finding, 631
- FXCop, 626-627

LINQPad, 630-631

NDepend, 626

NUnit, 623-625

Process Explorer, 628-629

Process Monitor, 628-629

Reflector, 622

RegexBuddy, 630

Virtual PC, 627-628

ToolStrip controls, 297**ToolStripItem, 297****ToolStripMenuItem, 299****ToString() method, 79, 82, 104, 385**

overriding, 29

types, formatting, 28-31

trace information, viewing, ASP.NET, 402-403**transactions, databases, 248-250****transformations (graphics), 341**

rotation, 342

scaling, 343

shearing, 343

translations, 342

translating hostnames to

IP addresses, 202

translations, graphics, 342**transparent images, drawing, 345****trie structures, creating, 173-176****triggers, WPF, style changes, 378****TryParse() method, 86, 97****TryParseExact() method, 97****tuples, creating, 151****type checking, 524**

deferring to runtime, 47-49

types. See also classes

anonymous types, creating, 22-23

converting, 40-41

creating, 28

discovering, 520-521

dynamically sized array of objects, creating, 140

enumerating, assemblies, 520-521

equality, determining, 32-33

floating-point types, choosing, 78

formatting

ICustomFormatter, 31-32

StringBuilder, 31-32

ToString() method, 28-31

generic types, constraining, 146-149

hashable types, creating, 34

implicit typing, 46-47

indexes, 36-37

inference, 46-47

- methods, adding, 54-55
- multiple generic types, creating, 146
- operators, overloading, 39-40
- parameters, labels, 146
- reference types, constraining, 147
- sortable types, creating, 34-35
- value types
 - constraining, 147
 - nulls, 42

U

- UAC (User Access Control), administration**
 - privileges, requesting, 578-581
- UI state, maintaining, ASP.NET, 430**
- UI threads**
 - timer events, Silverlight, 451-452
 - updates, ensuring, 285-287
- Uls, websites, creating, 418-422**
- Ultimate Developer and Power Users Tool List for Windows, 631**
- unchanged memory, allocating, 488-489**
- undo commands, implementing, command objects, 545-552**
- unhandled exceptions, catching, 72-75**
- Unicode strings, 111**
- unmanaged resources, cleaning up, finalization, 475-477**
- unrecoverable errors, indicating, 64**
- updates, UI threads, ensuring, 285-287**
- updating, databases, DataSet, 252-254**
- uploading files, FTP sites, 213-214**
- user configuration values, Windows Forms, 314-316**
- user controls**
 - ASP.NET, creating, 414-417
 - Windows Forms, creating, 308-313
- user date, maintaining, ASP.NET, 431-433**
- user folders, paths, retrieving, 194**
- user input**
 - data validation, ASP.NET, 425-429
 - validating, regular expressions, 134-136
- user logins, authentication, 406-409**
- user-defined objects, forms, cutting and pasting, 325-327**
- users**
 - session state, storing, 433-434
 - web pages, redirecting to, 405-406

V

- validation**
 - user input
 - ASP.NET, 425-429
 - regular expressions, 134-136
 - XML documents, 270-271
- Validation Demo—Default.aspx listing (19.13), 425-427**
- Validation Demo—Default.aspx.cs listing (19.14), 427-428**
- validity, enumerations, determining, 103**
- value types**
 - constraining, 147
 - nulls, assigning to, 42
- values**
 - arrays, reversing, 166-167
 - enumerations, listing, 103
 - evaluation, deferring, 57-58
 - explicit values, enumerations, 100
 - statements, choosing, 52-53
- var keyword, 22-23, 47**
- variables, declaring, 46-47**
- versatility, 28**
- Vertex3d class, 9**
 - formatting
 - ICustomFormatter, 31-32
 - StringBuilder, 31-32
 - ToString() method, 28-31
- video**
 - 3D surfaces, WPF, 392-394
 - playing over web, Silverlight, 445-448
 - progress bars, Silverlight, 449-450
- virtual mode, Windows Forms ListView, 317-318**
- Virtual PC, 627-628**
- Vista file dialogs, 594**
- Visual Studio**
 - databases, creating, 238-239
 - proxy classes, generating, 226
 - Smart Tags, 20

W

- wait cursor, Windows Forms, resetting, 327-328**
- WCF (Windows Communication Foundation), 208**
 - Internet, communication over, 231-232
 - multiple machines, communication, 229-230

- processes, communicating between, 222-229
- service interface, defining, 223-224
- services, discovering, 233-236
- weak references, 484**
- web browser capabilities, determining, 404**
- web browsers, applications**
 - embedding, 214-216
 - running out of, 453-454
- web pages**
 - AJAX pages, creating, 423-425
 - master pages, ASP.NET, 409-411
 - users, redirecting to, 405-406
- web services, querying, LINQ, 469-471**
- websites**
 - menus, adding, 411-412
 - Uls, creating, 418-422
- webcams, capturing, Silverlight, 455-457**
- WidgetGraphicView.xaml listing (25.2), 558**
- Win32 API functions, calling, .NET, 588-589**
- Window1.xaml listing (27.1), 600-601**
- Window1.xaml.cs listing (18.2), 381-383**
- Window1.xaml.cs listing (27.2), 601**
- windows**
 - nonrectangular windows, creating, 598-601
 - WPF (Windows Presentation Foundation)
 - displaying, 366-367
 - menu bars, 367-368
 - positioning controls, 367
 - status bars, 369
 - toolbars, 369-370
- Windows 7**
 - file dialogs, 594
 - functionality, accessing, 593-594
 - libraries, accessing, 594
- Windows Forms, 296, 330**
 - anti-aliasing, 348-349
 - bitmap pixels, accessing directly, 347-348
 - brushes, creating, 339-341
 - clipboard, 323-327
 - color picker, 330-331
 - colors, converting, 331-335
 - configuration values, 314-316
 - controls, binding data, 250-252
 - flicker-free drawing, 349-350
 - forms, inheritance, 304-308
 - graphics
 - color definitions, 330
 - resizing, 350-351
 - thumbnails, 351-352
 - horizontal tilt wheel, 319-323
 - images, drawing, 344-345
 - ListView, virtual mode, 317-318
 - menu bars, adding, 297-299
 - menu items, disabling dynamically, 300
 - modal forms, creating, 296
 - modeless forms, creating, 296
 - mouse cursor, distance, 354-355
 - multiscreen captures, taking, 352-354
 - nonrectangular windows, creating, 598-600
 - off-screen buffers, drawing to, 346-347
 - pens, creating, 337, 339
 - points
 - circles, 355-356
 - ellipse, 356-357
 - mouse cursor, 354-355
 - rectangles, 355
 - rectangles, intersection, 357
 - shapes, drawing, 335-337
 - splash screens, displaying, 614-616
 - split window interface, creating, 302-303
 - status bars, adding, 300
 - text, drawing, 344
 - timers, 313-314
 - toolbars, adding, 301-302
 - transformations, 341
 - rotation, 342
 - scaling, 343
 - shearing, 343
 - translations, 342
 - transparent images, drawing, 345
 - unhandled exceptions, catching, 73
 - user controls, creating, 308-313
 - wait cursor, resetting, 327-328
- Windows Forms applications, localization, 563-564**
- Windows Internals, 475**
- Windows Presentation Foundation (WPF). See WPF (Windows Presentation Foundation)**
- Windows services**
 - creating, 585-588
 - managing, 584

WinForms

- UI threads, updates, 286

- WPF applications, 400

WinForms applications, WPF, 398-399**words, strings, reversing, 124-125****WPF (Windows Presentation****Foundation), 366**

- 3D geometry, rendering, 389-392

- 3D surfaces

- interactive controls, 395-398

- video, 392-394

- bound data, displaying, 385-386

- commands, enabling/disabling, 374

- controls

- appearance/functionality, 377

- binding data to, 254-256

- binding properties, 379-383

- designing, 386-387

- expanding/collapsing, 375-376

- custom commands, 371-373

- data binding

- collections, 385

- value conversions, 383-385

- value formatting, 383

- element properties, animating, 388-389

- events, responding to, 376-377

- layout method, choosing, 367

- Model-View-ViewModel pattern, 552-562

- defining model, 553-554

- defining view, 557-558

- defining ViewModel, 555-556

- nonrectangular windows, creating, 600-601

- RoutedEvents, 377

- screen savers, creating, 605-614

- splash screens, displaying, 616-619

- standard commands, 370-371

- triggers, style changes, 378

- UI threads, updates, 286

- windows

- displaying, 366-367

- menu bars, 367-368

- positioning controls, 367

- status bars, 369

- toolbars, 369-370

- WinForms, applications in, 398-400

WPF applications

- localization, 565-569

- unhandled exceptions, catching, 74

writing

- binary files, 179

- events, event logs, 581-583

- text files, 178-179

- XML, 266-267

X-Z**XAML, localization, 566-568****XML (eXtensible Markup Language), 262**

- database data, transforming to, 273-276

- generating, LINQ, 467

- objects, serialization, 262-266

- querying, XPath, 271-272

- writing, 266-267

- XML documents, validating, 270-271

- XML files, reading, 268-270

XML configuration files, registry, compared, 584**XML documents**

- querying, LINQ, 466-467

- validating, 270-271

XML files, reading, 268-270**XmlDocument class**

- XML, writing, 266-267

- XML files, reading, 268

XmlTextReader class, XML files, reading, 269-270**XmlWriter, XML, writing, 267****XPath, XML, querying, 271-272**

- zip codes, validating, 135