

A Brain-Friendly Guide

Covers
C# 3.0 and
Visual Studio 2008

Head First C#



Boss your
data around
with LINQ

Build a fully
functional
retro classic
arcade game



Learn how
extension
methods helped
Sue bend the
rules in Objectville

**A Learner's Guide to
Real-World Programming
with C# and .NET**

Discover the
secrets of
abstraction and
inheritance



See how Jim used
generic collections to
wrangle his data

O'REILLY®

Andrew Stellman
& Jennifer Greene

Advance Praise for *Head First C#*

“I’ve never read a computer book cover to cover, but this one held my interest from the first page to the last. If you want to learn C# in depth and have fun doing it, this is THE book for you.”

— **Andy Parker, fledgling C# programmer**

“Head First C# is a great book for hobbyist programmers. It provides examples and guidance on a majority of the things [those] programmers are likely to encounter writing applications in C#.”

— **Peter Ritchie, Microsoft MVP (2006-2007), Visual Developer, C#**

“With Head First C#, Andrew and Jenny have presented an excellent tutorial on learning C#. It is very approachable while covering a great amount of detail in a unique style. If you’ve been turned off by more conventional books on C#, you’ll love this one.”

— **Jay Hilyard, Software Developer, co-author of *C# 3.0 Cookbook***

“Head First C# is perfect blend of unique and interesting ways covering most of the concepts of programming. Fun excercises, bullet points, and even comic strips are some of the catchy and awesome works that this book has. The game-based labs are something that you really don’t want to miss. [This book is] a great work... the novice as [well as the] well-experienced will love this book. GREAT JOB!”

— **Aayam Singh, .NET professional**

“Head First C# is a highly enjoyable tutorial, full of memorable examples and entertaining exercises. Its lively style is sure to captivate readers—from the humorously annotated examples, to the Fireside Chats, where the abstract class and interface butt heads in a heated argument! For anyone new to programming, there’s no better way to dive in.”

— **Joseph Albahari, C# Design Architect at Egton Medical Information Systems, the UK’s largest primary healthcare software supplier, co-author of *C# 3.0 in a Nutshell***

“[Head First C#] was an easy book to read and understand. I will recommend this book to any developer wanting to jump into the C# waters. I will recommend it to the advanced developer that wants to understand better what is happening with their code. [I will recommend it to developers who] want to find a better way to explain how C# works to their less-seasoned developer friends.”

— **Giuseppe Turitto, C# and ASP.NET developer for Cornwall Consulting Group**

“Andrew and Jenny have crafted another stimulating Head First learning experience. Grab a pencil, a computer, and enjoy the ride as you engage your left brain, right brain, and funny bone.”

— **Bill Mietelski, Software Engineer**

“Going through this Head First C# book was a great experience. I have not come across a book series which actually teaches you so well... This is a book I would definitely recommend to people wanting to learn C#”

— **Krishna Pala, MCP**

Praise for other *Head First* books

“Kathy and Bert’s *Head First Java* transforms the printed page into the closest thing to a GUI you’ve ever seen. In a wry, hip manner, the authors make learning Java an engaging ‘what’re they gonna do next?’ experience.”

—**Warren Keuffel, Software Development Magazine**

“Beyond the engaging style that drags you forward from know-nothing into exalted Java warrior status, *Head First Java* covers a huge amount of practical matters that other texts leave as the dreaded “exercise for the reader...” It’s clever, wry, hip and practical—there aren’t a lot of textbooks that can make that claim and live up to it while also teaching you about object serialization and network launch protocols. ”

—**Dr. Dan Russell, Director of User Sciences and Experience Research
IBM Almaden Research Center (and teaches Artificial Intelligence at Stanford University)**

“It’s fast, irreverent, fun, and engaging. Be careful—you might actually learn something!”

—**Ken Arnold, former Senior Engineer at Sun Microsystems
Co-author (with James Gosling, creator of Java), *The Java Programming Language***

“I feel like a thousand pounds of books have just been lifted off of my head.”

—**Ward Cunningham, inventor of the Wiki and founder of the Hillside Group**

“Just the right tone for the geeked-out, casual-cool guru coder in all of us. The right reference for practical development strategies—gets my brain going without having to slog through a bunch of tired stale professor-speak.”

—**Travis Kalanick, Founder of Scour and Red Swoosh
Member of the MIT TR100**

“There are books you buy, books you keep, books you keep on your desk, and thanks to O’Reilly and the Head First crew, there is the penultimate category, Head First books. They’re the ones that are dog-eared, mangled, and carried everywhere. Head First SQL is at the top of my stack. Heck, even the PDF I have for review is tattered and torn.”

— **Bill Sawyer, ATG Curriculum Manager, Oracle**

“This book’s admirable clarity, humor and substantial doses of clever make it the sort of book that helps even non-programmers think well about problem-solving.”

— **Cory Doctorow, co-editor of Boing Boing
Author, *Down and Out in the Magic Kingdom*
and *Someone Comes to Town, Someone Leaves Town***

Praise for other *Head First* books

“I received the book yesterday and started to read it...and I couldn’t stop. This is definitely très ‘cool.’ It is fun, but they cover a lot of ground and they are right to the point. I’m really impressed.”

— **Erich Gamma, IBM Distinguished Engineer, and co-author of *Design Patterns***

“One of the funniest and smartest books on software design I’ve ever read.”

— **Aaron LaBerge, VP Technology, ESPN.com**

“What used to be a long trial and error learning process has now been reduced neatly into an engaging paperback.”

— **Mike Davidson, CEO, Newsvine, Inc.**

“Elegant design is at the core of every chapter here, each concept conveyed with equal doses of pragmatism and wit.”

— **Ken Goldstein, Executive Vice President, Disney Online**

“I ♥ Head First HTML with CSS & XHTML—it teaches you everything you need to learn in a ‘fun coated’ format.”

— **Sally Applin, UI Designer and Artist**

“Usually when reading through a book or article on design patterns, I’d have to occasionally stick myself in the eye with something just to make sure I was paying attention. Not with this book. Odd as it may sound, this book makes learning about design patterns fun.

“While other books on design patterns are saying ‘Buehler... Buehler... Buehler...’ this book is on the float belting out ‘Shake it up, baby!’”

— **Eric Wuehler**

“I literally love this book. In fact, I kissed this book in front of my wife.”

— **Satish Kumar**

Other related books from O'Reilly

Programming C# 3.0

C# 3.0 in a Nutshell

C# 3.0 Cookbook™

C# 3.0 Design Patterns

C# Essentials

C# Language Pocket Reference

Other books in O'Reilly's *Head First* series

Head First Java

Head First Object-Oriented Analysis and Design (OOA&D)

Head Rush Ajax

Head First HTML with CSS and XHTML

Head First Design Patterns

Head First Servlets and JSP

Head First EJB

Head First PMP

Head First SQL

Head First Software Development

Head First JavaScript

Head First Ajax

Head First Statistics

Head First Physics (2008)

Head First Programming (2008)

Head First Ruby on Rails (2008)

Head First PHP & MySQL (2008)

Head First C#

Wouldn't it be dreamy
if there was a *C#* book that
was more fun than endlessly
debugging code? It's probably
nothing but a fantasy...



Andrew Stellman
Jennifer Greene

O'REILLY®

Beijing • Cambridge • Köln • Sebastopol • Taipei • Tokyo

Head First C#

by Andrew Stellman and Jennifer Greene

Copyright © 2008 O'Reilly Media, Inc. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly Media books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*safari.oreilly.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Series Creators:	Kathy Sierra, Bert Bates
Series Editor:	Brett D. McLaughlin
Design Editor:	Louise Barr
Cover Designers:	Louise Barr, Steve Fehler
Production Editor:	Sanders Kleinfeld
Proofreader:	Colleen Gorman
Indexer:	Julie Hawks
Page Viewers:	Quentin the whippet and Tequila the pomeranian

Printing History:

November 2007: First Edition.



The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. The *Head First* series designations, *Head First C#*, and related trade dress are trademarks of O'Reilly Media, Inc.

Microsoft, Windows, Visual Studio, MSDN, the .NET logo, Visual Basic and Visual C# are registered trademarks of Microsoft Corporation.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc., was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and the authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

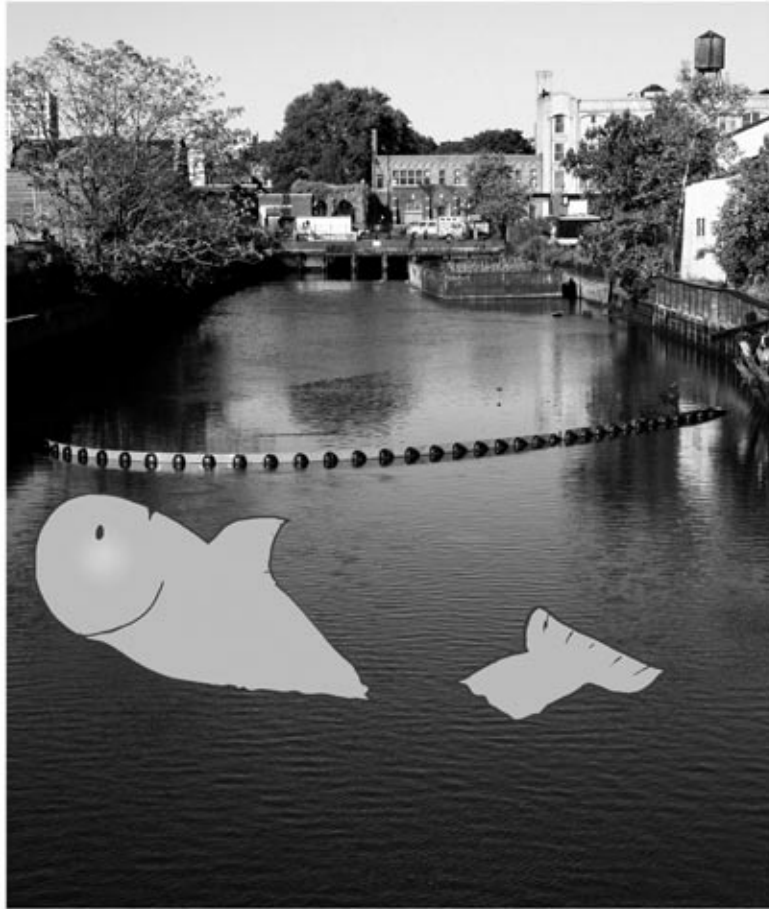
No bees, space aliens, or comic book heroes were harmed in the making of this book.



This book uses RepKover™, a durable and flexible lay-flat binding.

ISBN: 978-0-596-51482-2

*This book is dedicated to the loving memory of Sludgie the Whale,
who swam to Brooklyn on April 17, 2007.*



*You were only in our canal for a day,
but you'll be in our hearts forever.*

Thanks for buying our book! We really love writing about this stuff, and we hope you get a kick out of reading it...

Andrew

This photo (and the photo of the Gowanus Canal) by Nisha Sondhe



... because we know you're going to have a great time learning C#.

Jenny

Andrew Stellman, despite being raised a New Yorker, has lived in Pittsburgh *twice*. The first time was when he graduated from Carnegie Mellon's School of Computer Science, and then again when he and Jenny were starting their consulting business and writing their first book for O'Reilly.

When he moved back to his hometown, his first job after college was as a programmer at EMI-Capitol Records—which actually made sense, since he went to LaGuardia High School of Music and Art and the Performing Arts to study cello and jazz bass guitar. He and Jenny first worked together at that same financial software company, where he was managing a team of programmers. He's had the privilege of working with some pretty amazing programmers over the years, and likes to think that he's learned a few things from them.

When he's not writing books, Andrew keeps himself busy writing useless (but fun) software, playing music (but video games even more), studying taiji and aikido, having a girlfriend named Lisa, and owning a pomeranian.

Jenny and Andrew have been building software and writing about software engineering together since they first met in 1998. Their first book, *Applied Software Project Management*, was published by O'Reilly in 2005. They published their first book in the Head First series, *Head First PMP*, in 2007.

They founded Stellman & Greene Consulting in 2003 to build a really neat software project for scientists studying herbicide exposure in Vietnam vets. When they're not building software or writing books, they do a lot of speaking at conferences and meetings of software engineers, architects and project managers.

Check out their blog, *Building Better Software*: <http://www.stellman-greene.com>

Jennifer Greene studied philosophy in college but, like everyone else in the field, couldn't find a job doing it. Luckily, she's a great software tester, so she started out doing it at an online service, and that's the first time she really got a good sense of what project management was.

She moved to New York in 1998 to test software at a financial software company. She managed a team of testers at a really cool startup that did artificial intelligence and natural language processing.

Since then, she's traveled all over the world to work with different software teams and build all kinds of cool projects.

She loves traveling, watching Bollywood movies, reading the occasional comic book, waiting for her Xbox to be repaired, drinking carloads of carbonated beverages, and owning a whippet.

Table of Contents (Summary)

	Intro	xxix
1	Get productive with C#: <i>Visual Applications, in 10 minutes or less</i>	1
2	It's All Just Code: <i>Under the hood</i>	43
3	Objects Get Oriented: <i>Making code make sense</i>	85
4	Types and References: <i>It's 10:00. Do you know where your data is?</i>	123
	C# Lab 1: <i>A Day at the Races</i>	163
5	Encapsulation: <i>Keep your privates... private</i>	173
6	Inheritance: <i>Your object's family tree</i>	205
7	Interfaces and abstract classes: <i>Making classes keep their promises</i>	251
8	Enums and collections: <i>Storing lots of data</i>	309
	C# Lab 2: <i>The Quest</i>	363
9	Reading and writing files: <i>Save the byte array, save the world</i>	385
10	Exception handling: <i>Putting Out Fires Gets Old</i>	439
11	Events and delegates: <i>What Your Code Does When You're Not Looking</i>	483
12	Review and preview: <i>Knowledge, Power, and Building Cool Stuff</i>	515
13	Controls and graphics: <i>Make it pretty</i>	563
14	Captain Amazing: <i>The Death of the Object</i>	621
15	LINQ: <i>Get control of your data</i>	653
	C# Lab 3: <i>Invaders</i>	681
i	Leftovers: <i>The top 5 things we wanted to include in this book</i>	703

Table of Contents (the real thing)

Intro

Your brain on C#. You're sitting around trying to *learn* something, but your *brain* keeps telling you all that learning *isn't important*. Your brain's saying, "Better leave room for more important things, like which wild animals to avoid and whether nude archery is a bad idea." So how *do* you trick your brain into thinking that your life really depends on learning C#?

Who is this book for?	xxx
We know what you're thinking	xxxi
Metacognition	xxxiii
Bend your brain into submission	xxxv
What you need for this book	xxxvi
Read me	xxxii
The technical review team	xxxiv
Acknowledgments	xxxv

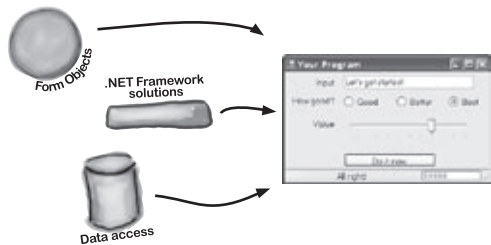
get productive with C#

1

Visual Applications, in 10 minutes or less

Want to build great programs really fast?

With C#, you've got a **powerful programming language** and a **valuable tool** at your fingertips. With the **Visual Studio IDE**, you'll never have to spend hours writing obscure code to get a button working again. Even better, you'll be able to **focus on getting your work done**, rather than remembering which method parameter was for the *name* for a button, and which one was for its *label*. Sound appealing? Turn the page, and let's get programming.



Why you should learn C#	2
C# and the Visual Studio IDE make lots of things easy	3
Help the CEO go paperless	4
Get to know your users' needs before you start building your program	5
Here's what you're going to build	6
What you do in Visual Studio...	8
What Visual Studio does for you...	8
Develop the user interface	12
Visual Studio, behind the scenes	14
Add to the auto-generated code	15
You can already run your application	16
We need a database to store our information	18
Creating the table for the Contact List	20
The blanks on contact card are columns in our People table	22
Finish building the table	25
Diagram your data so your application can access it	26
Insert your card data into the database	28
Connect your form to your database objects with a data source	30
Add database-driven controls to your form	32
Good apps are intuitive to use	34
How to turn YOUR application into EVERYONE'S application	37
Give your users the application	38
You're NOT done: test your installation	39
You built a complete data-driven application	40

2

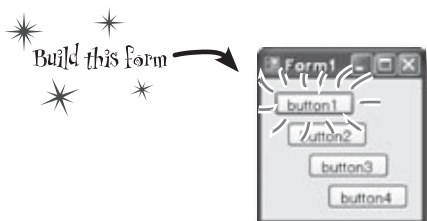
it's all just code

Under the Hood

You're a programmer, not just an IDE-user.

You can get a lot of work done using the IDE. But there's only so far it can take you. Sure, there are a lot of **repetitive tasks** that you do when you build an application. And the IDE is great at doing those things for you. But working with the IDE is *only the beginning*. You can get your programs to do so much more—and **writing C# code** is how you do it. Once you get the hang of coding, there's *nothing* your programs can't do.

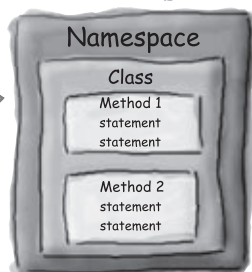
When you're doing this...	44
...the IDE does this	45
Where programs come from	46
The IDE helps you code	48
When you change things in the IDE, you're also changing your code	50
Anatomy of a program	52
Your program knows where to start	54
You can change your program's entry point	56
Two classes can be in the same namespace	61
Your programs use variables to work with data	62
C# uses familiar math symbols	64
Loops perform an action over and over again	65
Time to start coding	66
if/else statements make decisions	67
Set up conditions and see if they're true	68



Every time you make a new program, you define a namespace for it so that its code is separate from the .NET Framework classes.

A class contains a piece of your program (although some very small programs can have just one class).

A class has one or more methods. Your methods always have to live inside a class. And methods are made up of statements – like the ones you've already seen.



objects get oriented

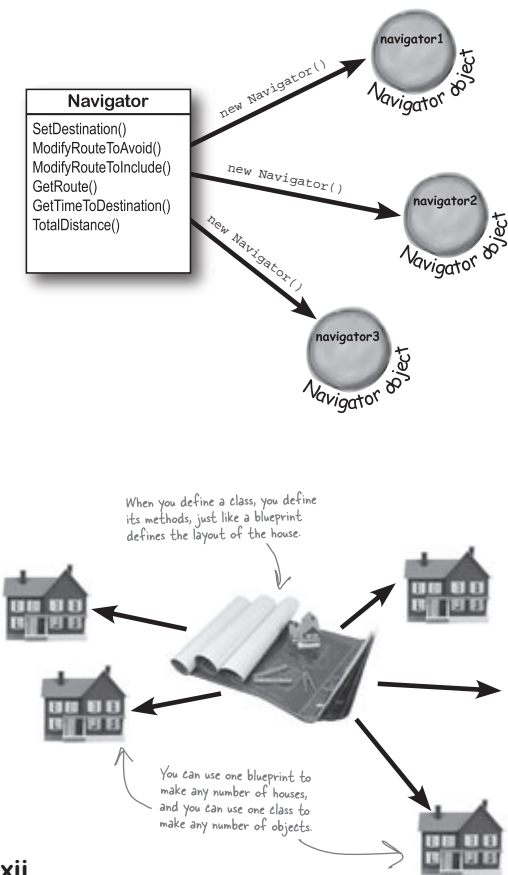
Making Code Make Sense

3

Every program you write solves a problem.

When you're building a program, it's always a good idea to start by thinking about what *problem* your program's supposed to solve. That's why **objects** are really useful. They let you structure your code based on the problem it's solving, so that you can spend your time *thinking about the problem* you need to work on rather than getting bogged down in the mechanics of writing code. When you use objects right, you end up with code that's *intuitive* to write, and easy to read and change.

How Mike thinks about his problems	86
How Mike's car navigation system thinks about his problems	87
Mike's Navigator class has methods to set and modify routes	88
Use what you've learned to build a simple application	89
Mike gets an idea	90
Mike can use objects to solve his problem	91
You use a class to build an object	92
When you create a new object from a class, it's called an instance of that class	93
A better solution... brought to you by objects!	94
An instance uses fields to keep track of things	98
Let's create some instances!	99
Thanks for the memory	100
What's on your program's mind	101
You can use class and method names to make your code intuitive	102
Give your classes a natural structure	104
Class diagrams help you organize your classes so they make sense	106
Build a class to work with some guys	110
Create a project for your guys	111
Build a form to interact with the guys	112
There's an even easier way to initialize objects	115
A few ideas for designing intuitive classes	116



types and references

It's 10:00. Do you know where your data is?

4

Data type, database, Lieutenant Commander Data...

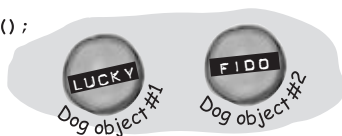
it's all important stuff. Without data, your programs are useless. You need **information** from your users, and you use that to look up or produce new information, to give back to them. In fact, almost everything you do in programming involves **working with data** in one way or another. In this chapter, you'll learn the ins and outs of C#'s **data types**, how to work with data in your program, and even figure out a few dirty secrets about **objects** (*psst... objects are data, too*).

The variable's type determines what kind of data it can store	124
A variable is like a data to-go cup	126
10 pounds of data in a 5 pound bag	127
Even when a number is the right size, you can't just assign it to any variable	128
When you cast a value that's too big, C# will adjust it automatically	129
C# does some casting automatically	130
When you call a method, the variables must match the types of the parameters	131
Combining = with an operator	136
Objects use variables, too	137
Refer to your objects with reference variables	138
References are like labels for your object	139
If there aren't any more references, your object gets garbage collected	140
Multiple references and their side effects	142
Two references means TWO ways to change an object's data	147
A special case: arrays	148
Arrays can contain a bunch of reference variables, too	149
Welcome to Sloppy Joe's Budget House o' Discount Sandwiches!	150
Objects use references to talk to each other	152
Where no object has gone before	153

```
Dog fido;
Dog lucky = new Dog();
```



```
fido = new Dog();
```



```
lucky = null;
```

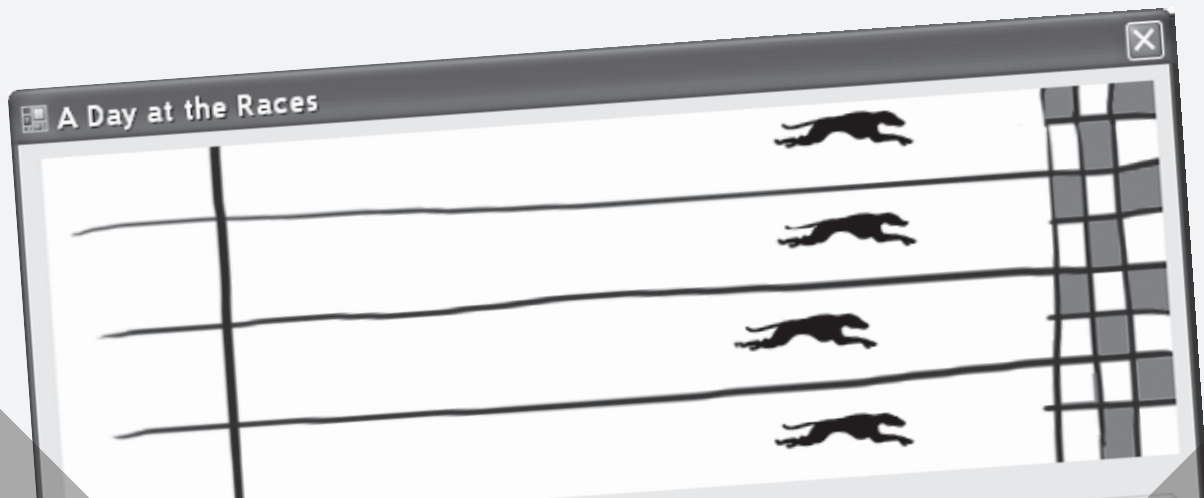


C# Lab 1

A Day at the Races

Joe, Bob, and Al love going to the track, but they're tired of losing all their money. They need you to build a simulator for them so they can figure out winners before they lay their money down. And, if you do a good job, they'll cut you in on their profits.

The Spec: Build a Racetrack Simulator	164
The Finished Product	172



encapsulation

5

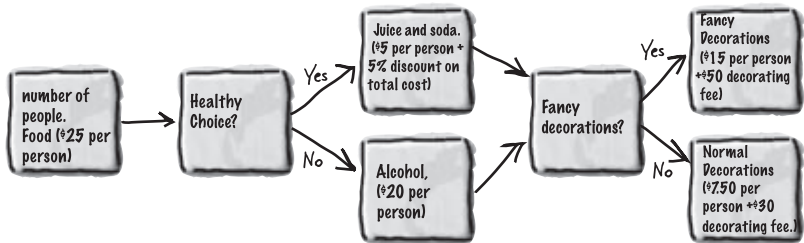
Keep your privates... private

Ever wished for a little more privacy?

Sometimes your objects feel the same way. Just like you don't want anybody you don't trust reading your journal, or paging through your bank statements, good objects don't let **other** objects go poking around their properties. In this chapter, you're going to learn about the power of **encapsulation**. You'll **make your object's data private**, and add methods to **protect how that data is accessed**.



Kathleen is an event planner	174
What does the estimator do?	175
Kathleen's Test Drive	180
Each option should be calculated individually	182
It's easy to accidentally misuse your objects	184
Encapsulation means keeping some of the data in a class private	185
Use encapsulation to control access to your class's methods and fields	186
But is the realName field REALLY protected?	187
Private fields and methods can only be accessed from inside the class	188
A few ideas for encapsulating classes	191
Encapsulation keeps your data pristine	192
Properties make encapsulation easier	193
Build an application to test the Farmer class	194
Use automatic properties to finish the class	195
What if we want to change the feed multiplier?	196
Use a constructor to initialize private fields	197



inheritance

Your object's family tree

6

Sometimes you *DO* want to be just like your parents.

Ever run across an object that *almost* does exactly what you want *your* object to do? Found yourself wishing that if you could just *change a few things*, that object would be perfect? Well that's just one reason that **inheritance** is one of the most powerful concepts and techniques in the C# language. Before you're through this chapter, you'll learn how to **subclass** an object to get its behavior, but keep the **flexibility** to make changes to that behavior. You'll **avoid duplicate code**, **model the real world** more closely, and end up with code that's **easier to maintain**.



Kathleen does birthday parties, too	206
We need a BirthdayParty class	207
One more thing... can you add a \$100 fee for parties over 12?	213
When your classes use inheritance, you only need to write your code once	214
Build up your class model by starting general and getting more specific	215
How would you design a zoo simulator?	216
Use inheritance to avoid duplicate code in subclasses	217
Different animals make different noises	218
Think about how to group the animals	219
Create the class hierarchy	220
Every subclass extends its base class	221
Use a colon to inherit from a base class	222
We know that inheritance adds the base class fields, properties, and methods to the subclass...	225
A subclass can override methods to change or replace methods it inherited	226
Any place where you can use a base class, you can use one of its subclasses instead	227
A subclass can access its base class using the base keyword	232
When a base class has a constructor, your subclass needs one too	233
Now you're ready to finish the job for Kathleen!	234
Build a beehive management system	239
First you'll build the basic system	240
Use inheritance to extend the bee management system	245

interfaces and abstract classes

Making classes keep their promises

7

Actions speak louder than words.

Sometimes you need to group your objects together based on the **things they can do** rather than the classes they inherit from. That's where **interfaces** come in—they let you work with any class that can do the job. But with **great power comes great responsibility**, and any class that implements an interface must promise to **fulfill all of its obligations...** or the compiler will break their kneecaps, see?

* Inheritance

Let's get back to bee-sics 252

We can use inheritance to create classes for different types of bees 253

An interface tells a class that it must implement certain methods and properties 254

Use the interface keyword to define an interface 255

Get a little practice using interfaces 256

Now you can create an instance of NectarStinger that does both jobs 257

Classes that implement interfaces have to include ALL of the interface's methods 258

You can't instantiate an interface, but you can reference an interface 260

Interface references work just like object references 261

You can find out if a class implements a certain interface with "is" 262

Interfaces can inherit from other interfaces 263

The RoboBee 4000 can do a worker bee's job without using valuable honey 264

is tells you what an object implements, as tells the compiler how to treat your object 265

A CoffeeMaker is also an Appliance 266

Upcasting works with both objects and interfaces 267

Downcasting lets you turn your appliance back into a coffee maker 268

Upcasting and downcasting work with interfaces, too 269

There's more than just public and private 273

Access modifiers change scope 274

Some classes should never be instantiated 277

An abstract class is like a cross between a class and an interface 278

Like we said, some classes should never be instantiated 280

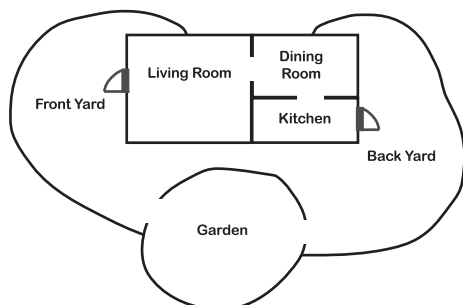
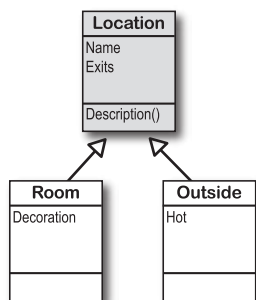
An abstract method doesn't have a body 281

Polymorphism means that one object can take many different forms 289

* Abstraction

* Encapsulation

* Polymorphism

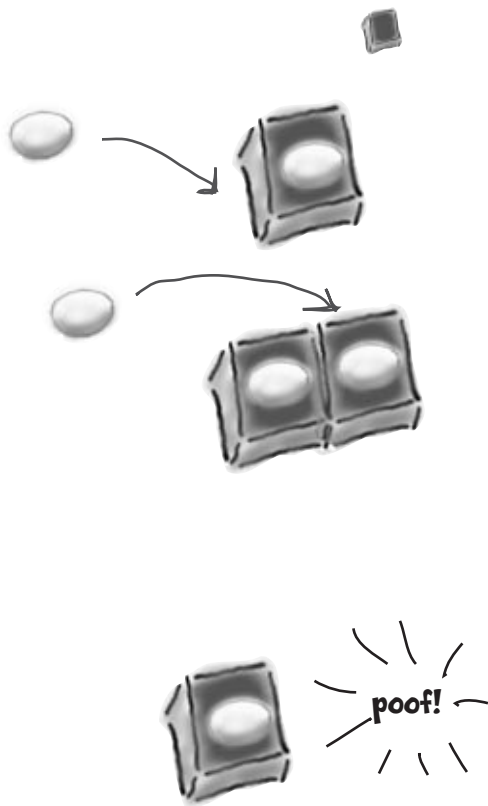


enums and collections

8 Storing lots of data

When it rains, it pours.

In the real world, you don't get to handle your data in tiny little bits and pieces. No, your data's going to come at you in **loads, piles and bunches**. You'll need some pretty powerful tools to organize all of it, and that's where **collections** come in. They let you **store, sort and manage** all the data that your programs need to pore through. That way you can think about writing programs to work with your data, and let the collections worry about keeping track of it for you.



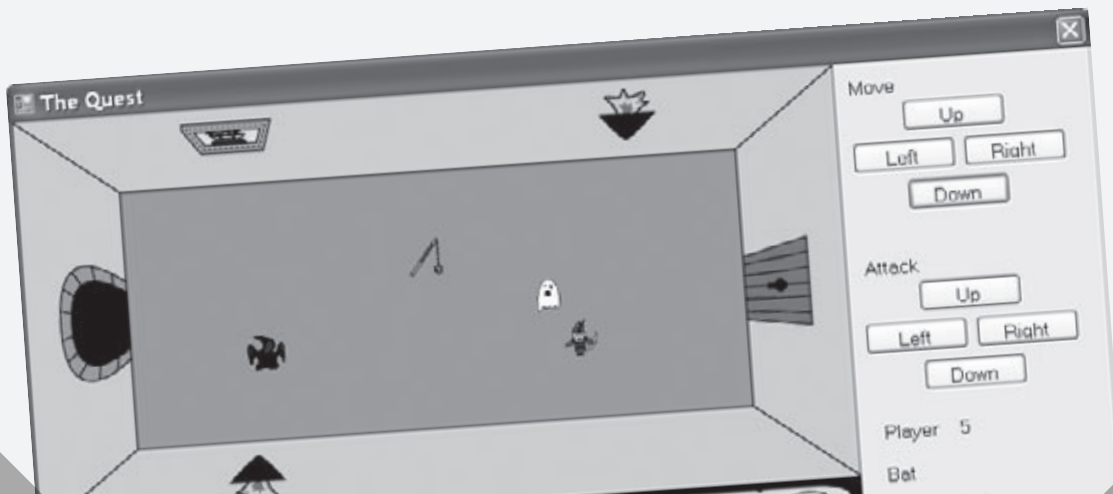
Strings don't always work for storing categories of data	310
Enums let you enumerate a set of valid values	311
Enums let you represent numbers with names	312
We could use an array to create a deck of cards...	315
Arrays are hard to work with	316
Lists make it easy to store collections of... anything	317
Lists are more flexible than arrays	318
Lists shrink and grow dynamically	321
List objects can store any type	322
Collection initializers work just like object initializers	326
Let's create a list of Ducks	327
Lists are easy, but SORTING can be tricky	328
Two ways to sort your ducks	329
Use IComparer to tell your List how to sort	330
Create an instance of your comparer object	331
IComparer can do complex comparisons	332
Use a dictionary to store keys and values	335
The Dictionary Functionality Rundown	336
Your key and value can be different types, too	337
You can build your own overloaded methods	343
And yet MORE collection types...	355
A queue is FIFO — First In, First Out	356
A stack is LIFO — Last In, First Out	357

C# Lab 2

The Quest

Your job is to build an adventure game where a mighty adventurer is on a quest to defeat level after level of deadly enemies. You'll build a turn-based system, which means the player makes one move and then the enemies make one move. The player can move or attack, and then each enemy gets a chance to move and attack. The game keeps going until the player either defeats all the enemies on all seven levels or dies.

The spec: build an adventure game	364
The fun's just beginning!	484



reading and writing files

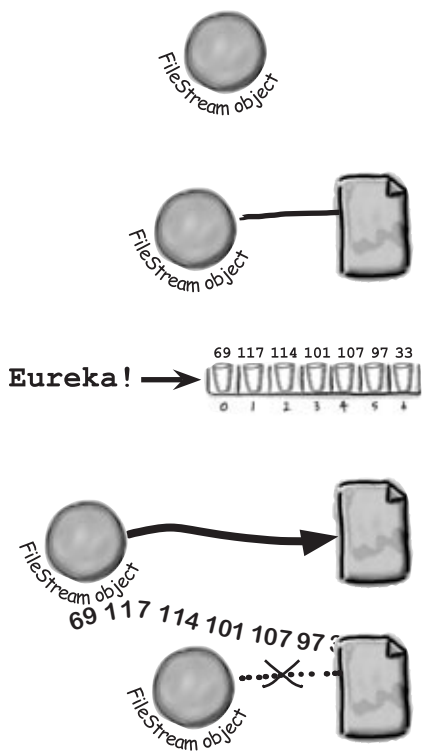
Save the byte array, save the world

9

Sometimes it pays to be a little persistent.

So far, all of your programs have been pretty short-lived. They fire up, run for a while, and shut down. But that's not always enough, especially when you're dealing with important information. You need to be able to **save your work**. In this chapter, we'll look at how to **write data to a file**, and then how to **read that information back in** from a file. You'll learn about the .NET **stream classes**, and also take a look at the mysteries of **hexadecimal** and **binary**.

C# uses streams to read and write data	386
Different streams read and write different things	387
A FileStream writes bytes to a file	388
Reading and writing takes two objects	393
Data can go through more than one stream	394
Use built-in objects to pop up standard dialog boxes	397
Dialog boxes are objects, too	399
Use the built-in File and Directory classes to work with files and directories	400
Use File Dialogs to open and save files	403
IDisposable makes sure your objects are disposed properly	405
Avoid file system errors with using statements	406
Writing files usually involves making a lot of decisions	412
Use a switch statement to choose the right option	413
Add an overloaded Deck() constructor that reads a deck of cards in from a file	415
What happens to an object when it's serialized?	417
But what exactly IS an object's state? What needs to be saved?	418
When an object is serialized, all of the objects it refers to get serialized too...	419
Serialization lets you read or write a whole object all at once	420
If you want your class to be serializable, mark it with the [Serializable] attribute	421
.NET converts text to Unicode automatically	425
C# can use byte arrays to move data around	426
Use a BinaryWriter to write binary data	427
You can read and write serialized files manually, too	429
StreamReader and StreamWriter will do just fine	433



10

exception handling
Putting out fires gets old**Programmers aren't meant to be firefighters.**

You've worked your tail off, waded through technical manuals and a few engaging Head First books, and you've reached the pinnacle of your profession: **master programmer**. But you're still getting pages from work because **your program crashes**, or **doesn't behave like it's supposed to**. Nothing pulls you out of the programming groove like having to fix a strange bug . . . but with **exception handling**, you can write code to **deal with problems** that come up. Better yet, you can even react to those problems, and **keep things running**.

Brian needs his excuses to be mobile	440
When your program throws an exception, .NET generates an Exception object.	444
Brian's code did something unexpected	446
All exception objects inherit from Exception	448
The debugger helps you track down and prevent exceptions in your code	449
Use the IDE's debugger to ferret out exactly what went wrong in the excuse manager	450
Uh-oh—the code's still got problems...	453
Handle exceptions with try and catch	455
What happens when a method you want to call is risky?	456
Use the debugger to follow the try/catch flow	458
If you have code that ALWAYS should run, use a finally block	460
Use the Exception object to get information about the problem	465
Use more than one catch block to handle multiple types of exceptions	466
One class throws an exception, another class catches the exception	467
Bees need an OutOfHoney exception	468
An easy way to avoid a lot of problems: using gives you try and finally for free	471
Exception avoidance: implement IDisposable to do your own clean up	472
The worst catch block EVER: comments	474
Temporary solutions are okay (temporarily)	475
A few simple ideas for exception handling	476
Brian finally gets his vacation...	481



11

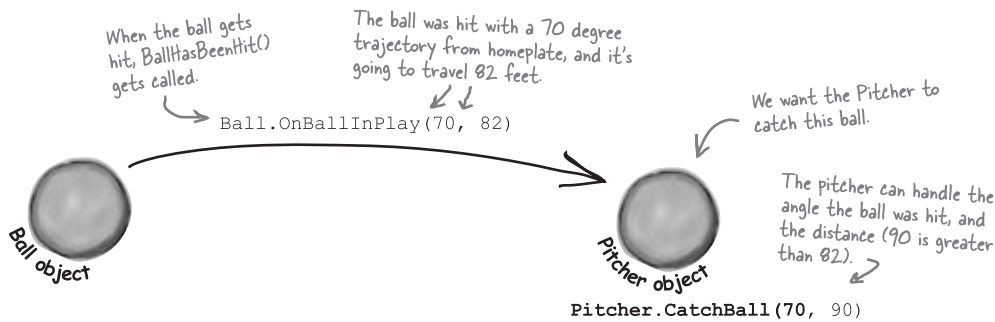
events and delegates

What your code does when you're not looking

Your objects are starting to think for themselves.

You can't always control what your objects are doing. Sometimes things...happen. And when they do, you want your objects to be smart enough to **respond to anything** that pops up. And that's what events are all about. One object *publishes* an event, other objects *subscribe*, and everyone works together to keep things moving. Which is great, until you've got too many objects responding to the same event. And that's when **callbacks** will come in handy.

Ever wish your objects could think for themselves?	484
But how does an object KNOW to respond?	484
When an EVENT occurs... objects listen	485
One object raises its event, others listen for it...	486
Then, the other objects handle the event	487
Connecting the dots	488
The IDE creates event handlers for you automatically	492
The forms you've been building all use events	498
Connecting event senders with event receivers	500
A delegate STANDS IN for an actual method	501
Delegates in action	502
Any object can subscribe to a public event...	505
Use a callback instead of an event to hook up exactly one object to a delegate	507
Callbacks use delegates, but NOT events	508



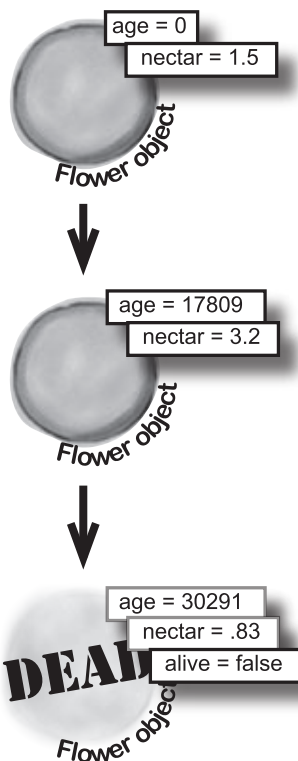
review and preview

Knowledge, power, and building cool stuff

12

Learning's no good until you BUILD something.

Until you've actually written working code, it's hard to be sure if you really *get* some of the tougher concepts in C#. In this chapter, we're going to learn about some new odds and ends: **timers** and dealing with collections using **LINQ** (to name a couple). We're also going to build phase I of a **really complex application**, and make sure you've got a good handle on what you've already learned from earlier chapters. So buckle up... it's time to build some **cool software**.

Life and death of a flower

You've come a long way, baby	516
We've also become beekeepers	517
The beehive simulator architecture	518
Building the beehive simulator	519
Life and death of a flower	523
Now we need a Bee class	524
Filling out the Hive class	532
The hive's Go() method	533
We're ready for the World	534
We're building a turn-based system	535
Giving the bees behavior	542
The main form tells the world to Go()	544
We can use World to get statistics	545
Timers fire events over and over again	546
The timer's using a delegate behind the scenes	547
Let's work with groups of bees	554
A collection collects... DATA	555
LINQ makes working with data in collections and databases easy	557

13

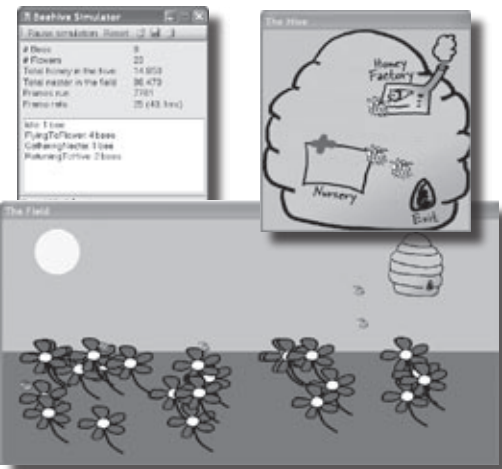
controls and graphics

Make it pretty

Sometimes you have to take graphics into your own hands.

We've spent a lot of time on relying on controls to handle everything visual in our applications. But sometimes that's not enough—like when you want to **animate a picture**. And once you get into animation, you'll end up **creating your own controls** for your .NET programs, maybe adding a little **double buffering**, and even **drawing directly onto your forms**. It all begins with the **Graphics** object, **Bitmaps**, and a determination to not accept the graphics status quo.

You've been using controls all along to interact with your programs	564
Form controls are just objects	565
Add a renderer to your architecture	568
Controls are well-suited for visual display elements	570
Build your first animated control	573
Your controls need to dispose their controls, too!	577
A UserControl is an easy way to build a control	578
Add the hive and field forms to the project	582
Build the Renderer	583
Let's take a closer look at those performance issues	590
You resized your Bitmaps using a Graphics object	592
Your image resources are stored in Bitmap objects	593
Use System.Drawing to TAKE CONTROL of graphics yourself	594
A 30-second tour of GDI+ graphics	595
Use graphics to draw a picture on a form	596
Graphics can fix our transparency problem...	601
Use the Paint event to make your graphics stick	602
A closer look at how forms and controls repaint themselves	605
Double buffering makes animation look a lot smoother	608
Double buffering is built into forms and controls	609
Use a Graphics object and an event handler for printing	614
PrintDocument works with the print dialog and print preview window objects	615

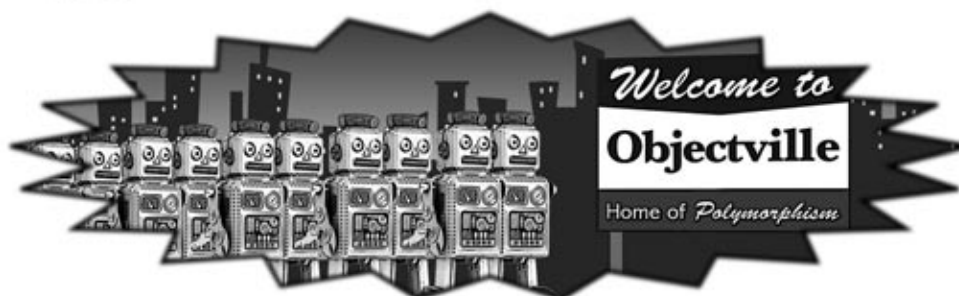


14

CAPTAIN AMAZING

THE DEATH OF THE OBJECT

Captain Amazing, Objectville's most amazing object pursues his arch-nemesis...	622
Your last chance to DO something... your object's finalizer	628
When EXACTLY does a finalizer run?	629
Dispose() works with using, finalizers work with garbage collection	630
Finalizers can't depend on stability	632
Make an object serialize itself in its Dispose()	633
Meanwhile, on the streets of Objectville...	636
A struct <i>looks</i> like an object...	637
..but <i>isn't</i> on the heap	637
Values get copied, references get assigned	638
Structs are value types; objects are reference types	639
The stack vs. the heap: more on memory	641
Captain Amazing... not so much	645
Extension methods add new behavior to EXISTING classes	646
Extending a fundamental type: string	648



15

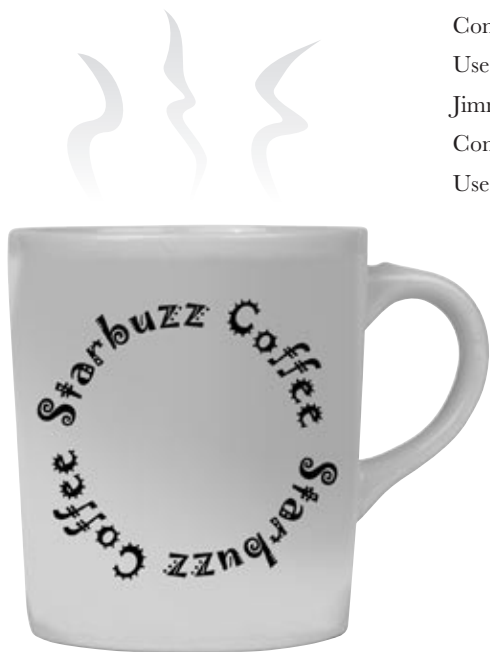
LINQ

Get control of your data

It’s a data-driven world... you better know how to live in it.

Gone are the days when you could program for days, even weeks, without dealing with **loads of data**. But today, **everything is about data**. In fact, you’ll often have to work with data from **more than one place**... and in more than one format. Databases, XML, collections from other programs... it’s all part of the job of a good C# programmer. And that’s where **LINQ** comes in. LINQ not only lets you **query data** in a simple, intuitive way, but it lets you **group data**, and **merge data from different data sources**.

An easy project...	654
...but the data’s all over the place	655
LINQ can pull data from multiple sources	656
.NET collections are already set up for LINQ	657
LINQ makes queries easy	658
LINQ is simple, but your queries don’t have to be	659
LINQ is versatile	662
LINQ can combine your results into groups	667
Combine Jimmy’s values into groups	668
Use join to combine two collections into one query	671
Jimmy saved a bunch of dough	672
Connect LINQ to a SQL database	674
Use a join query to connect Starbuzz and Objectville	678



C# Lab 3

Invaders

In this lab you'll pay homage to one of the most popular, revered and replicated icons in video game history, a game that needs no further introduction. It's time to build Invaders.

The grandfather of video games

682

And yet there's more to do...

701



leftovers




The top 5 things we wanted to include in this book

The fun’s just beginning!

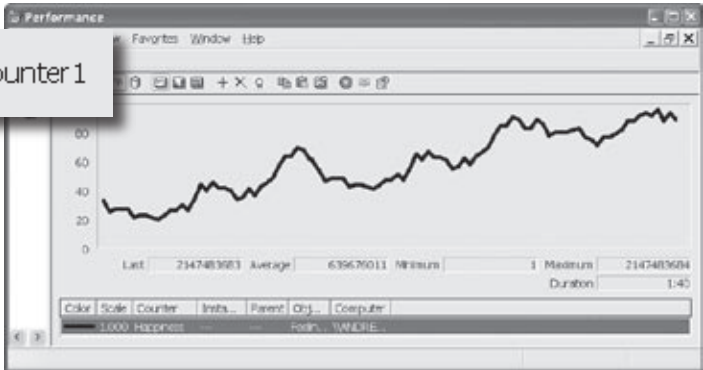
We’ve shown you a lot of great tools to build some really **powerful software** with C#. But there’s no way that we could include **every single tool, technology or technique** in this book—there just aren’t enough pages. We had to make some *really tough choices* about what to include and what to leave out. Here are some of the topics that didn’t make the cut. But even though we couldn’t get to them, we still think that they’re **important and useful**, and we wanted to give you a small head start with them.

#1 LINQ to XML	704
#2 Refactoring	706
#3 Some of our favorite Toolbox components	708
#4 Console Applications	710
#5 Windows Presentation Foundation	712
Did you know that C# and the .NET Framework can...	714

 backgroundWorker 1

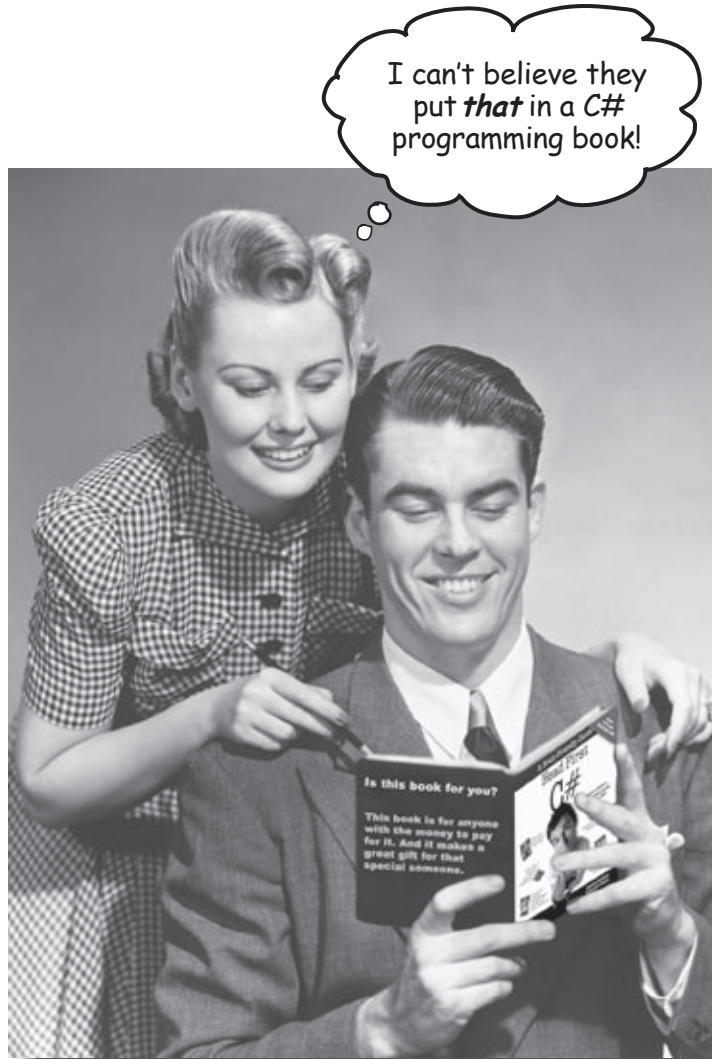
 fileSystemWatcher 1

 performanceCounter 1



how to use this book

Intro



In this section, we answer the burning question:
"So why DID they put that in a C# programming book?"

Who is this book for?

If you can answer “yes” to all of these:

- ① Do you want to **learn C#**?
- ② Do you like to tinker—do you learn by doing, rather than just reading?
- ③ Do you prefer **stimulating dinner party conversation** to **dry, dull, academic lectures**?

this book is for you.

Who should probably back away from this book?

If you can answer “yes” to any of these:

- ① Does the idea of writing a lot of code make you bored and a little twitchy?
- ② Are you a kick-butt C++ or Java programmer looking for a reference book?
- ③ Are you **afraid to try something different**? Would you rather have a root canal than mix stripes with plaid? Do you believe that a technical book can’t be serious if C# concepts are anthropomorphized?

this book is not for you.



[Note from marketing: this book is for anyone with a credit card.]

We know what you're thinking.

"How can *this* be a serious C# programming book?"

"What's with all the graphics?"

"Can I actually *learn* it this way?"

And we know what your *brain* is thinking.

Your brain craves novelty. It's always searching, scanning, *waiting* for something unusual. It was built that way, and it helps you stay alive.

So what does your brain do with all the routine, ordinary, normal things you encounter? Everything it *can* to stop them from interfering with the brain's *real* job—recording things that *matter*. It doesn't bother saving the boring things; they never make it past the "this is obviously not important" filter.

How does your brain *know* what's important? Suppose you're out for a day hike and a tiger jumps in front of you, what happens inside your head and body?

Neurons fire. Emotions crank up. *Chemicals surge*.

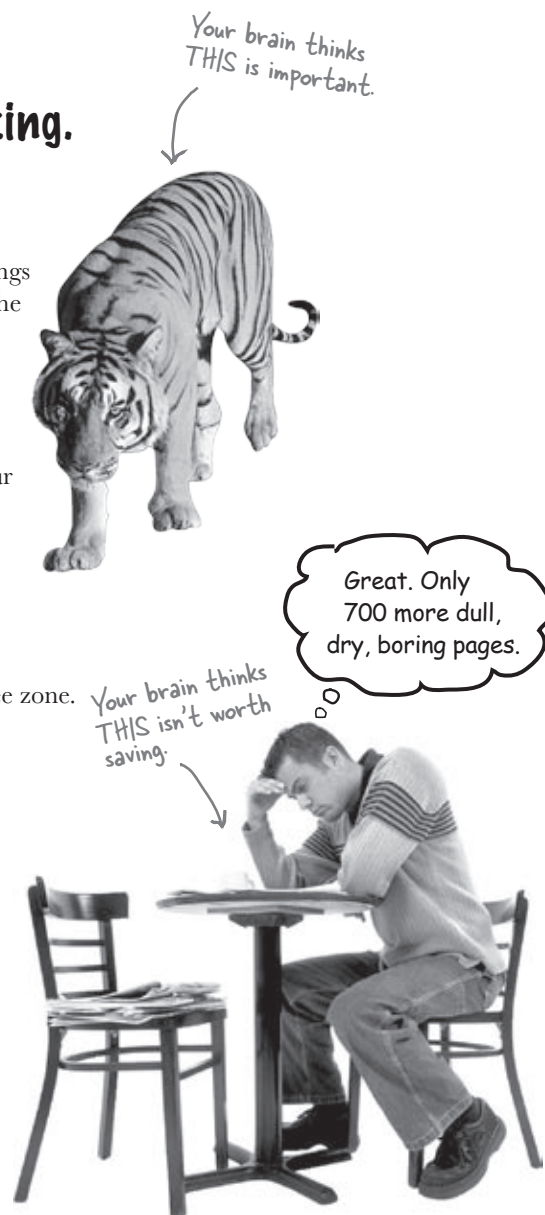
And that's how your brain knows...

This must be important! Don't forget it!

But imagine you're at home, or in a library. It's a safe, warm, tiger-free zone. You're studying. Getting ready for an exam. Or trying to learn some tough technical topic your boss thinks will take a week, ten days at the most.

Just one problem. Your brain's trying to do you a big favor. It's trying to make sure that this *obviously* non-important content doesn't clutter up scarce resources. Resources that are better spent storing the really *big* things. Like tigers. Like the danger of fire. Like how you should never have posted those "party" photos on your Facebook page.

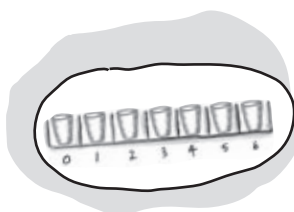
And there's no simple way to tell your brain, "Hey brain, thank you very much, but no matter how dull this book is, and how little I'm registering on the emotional Richter scale right now, I really *do* want you to keep this stuff around."



We think of a “Head First” reader as a learner.

So what does it take to *learn* something? First, you have to *get* it, then make sure you don't *forget* it. It's not about pushing facts into your head. Based on the latest research in cognitive science, neurobiology, and educational psychology, *learning* takes a lot more than text on a page. We know what turns your brain on.

Some of the Head First learning principles:



Make it visual. Images are far more memorable than words alone, and make learning much more effective (up to 89% improvement in recall and transfer studies). It also makes things more understandable. Put the words within or near the graphics they relate to, rather than on the bottom or on another page, and learners will be up to *twice* as likely to solve problems related to the content.

Use a conversational and personalized style. In recent studies, students performed up to 40% better on post-learning tests if the content spoke directly to the reader, using a first-person, conversational style rather than taking a formal tone. Tell stories instead of lecturing. Use casual language. Don't take yourself too seriously. Which would *you* pay more attention to: a stimulating dinner party companion, or a lecture?

Get the learner to think more deeply. In other words, unless you actively flex your neurons, nothing much happens in your head. A reader has to be motivated, engaged, curious, and inspired to solve problems, draw conclusions, and generate new knowledge. And for that, you need challenges, exercises, and thought-provoking questions, and activities that involve both sides of the brain and multiple senses.



Get—and keep—the reader's attention. We've all had the “I really want to learn this but I can't stay awake past page one” experience. Your brain pays attention to things that are out of the ordinary, interesting, strange, eye-catching, unexpected. Learning a new, tough, technical topic doesn't have to be boring. Your brain will learn much more quickly if it's not.



Touch their emotions. We now know that your ability to remember something is largely dependent on its emotional content. You remember what you care about. You remember when you *feel* something. No, we're not talking heart-wrenching stories about a boy and his dog. We're talking emotions like surprise, curiosity, fun, “what the...?”, and the feeling of “I Rule!” that comes when you solve a puzzle, learn something everybody else thinks is hard, or realize you know something that “I'm more technical than thou” Bob from engineering *doesn't*.



Metacognition: thinking about thinking

If you really want to learn, and you want to learn more quickly and more deeply, pay attention to how you pay attention. Think about how you think. Learn how you learn.

Most of us did not take courses on metacognition or learning theory when we were growing up. We were *expected* to learn, but rarely *taught* to learn.

But we assume that if you're holding this book, you really want to learn how to build programs in C#. And you probably don't want to spend a lot of time. If you want to use what you read in this book, you need to *remember* what you read. And for that, you've got to *understand* it. To get the most from this book, or *any* book or learning experience, take responsibility for your brain. Your brain on *this* content.

The trick is to get your brain to see the new material you're learning as Really Important. Crucial to your well-being. As important as a tiger. Otherwise, you're in for a constant battle, with your brain doing its best to keep the new content from sticking.

So just how *DO* you get your brain to treat C# like it was a hungry tiger?

There's the slow, tedious way, or the faster, more effective way. The slow way is about sheer repetition. You obviously know that you *are* able to learn and remember even the dullest of topics if you keep pounding the same thing into your brain. With enough repetition, your brain says, "This doesn't *feel* important to him, but he keeps looking at the same thing *over* and *over* and *over*, so I suppose it must be."

The faster way is to do **anything that increases brain activity**, especially different *types* of brain activity. The things on the previous page are a big part of the solution, and they're all things that have been proven to help your brain work in your favor. For example, studies show that putting words *within* the pictures they describe (as opposed to somewhere else in the page, like a caption or in the body text) causes your brain to try to make sense of how the words and picture relate, and this causes more neurons to fire. More neurons firing = more chances for your brain to *get* that this is something worth paying attention to, and possibly recording.

A conversational style helps because people tend to pay more attention when they perceive that they're in a conversation, since they're expected to follow along and hold up their end. The amazing thing is, your brain doesn't necessarily *care* that the "conversation" is between you and a book! On the other hand, if the writing style is formal and dry, your brain perceives it the same way you experience being lectured to while sitting in a roomful of passive attendees. No need to stay awake.

But pictures and conversational style are just the beginning.



Here's what WE did:

We used **pictures**, because your brain is tuned for visuals, not text. As far as your brain's concerned, a picture really *is* worth a thousand words. And when text and pictures work together, we embedded the text *in* the pictures because your brain works more effectively when the text is *within* the thing the text refers to, as opposed to in a caption or buried in the text somewhere.

We used **redundancy**, saying the same thing in *different* ways and with different media types, and *multiple senses*, to increase the chance that the content gets coded into more than one area of your brain.

We used concepts and pictures in **unexpected** ways because your brain is tuned for novelty, and we used pictures and ideas with at least *some emotional content*, because your brain is tuned to pay attention to the biochemistry of emotions. That which causes you to *feel* something is more likely to be remembered, even if that feeling is nothing more than a little **humor, surprise, or interest**.

We used a personalized, **conversational style**, because your brain is tuned to pay more attention when it believes you're in a conversation than if it thinks you're passively listening to a presentation. Your brain does this even when you're *reading*.

We included more than 80 **activities**, because your brain is tuned to learn and remember more when you *do* things than when you *read* about things. And we made the exercises challenging-yet-do-able, because that's what most people prefer.

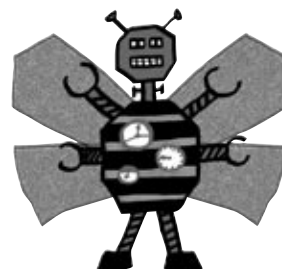
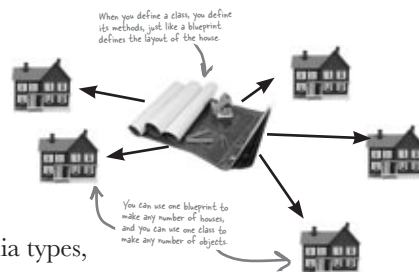
We used **multiple learning styles**, because *you* might prefer step-by-step procedures, while someone else wants to understand the big picture first, and someone else just wants to see an example. But regardless of your own learning preference, *everyone* benefits from seeing the same content represented in multiple ways.

We include content for **both sides of your brain**, because the more of your brain you engage, the more likely you are to learn and remember, and the longer you can stay focused. Since working one side of the brain often means giving the other side a chance to rest, you can be more productive at learning for a longer period of time.

And we included **stories** and exercises that present **more than one point of view**, because your brain is tuned to learn more deeply when it's forced to make evaluations and judgments.

We included **challenges**, with exercises, and by asking **questions** that don't always have a straight answer, because your brain is tuned to learn and remember when it has to *work* at something. Think about it—you can't get your *body* in shape just by *watching* people at the gym. But we did our best to make sure that when you're working hard, it's on the *right* things. That **you're not spending one extra dendrite** processing a hard-to-understand example, or parsing difficult, jargon-laden, or overly terse text.

We used **people**. In stories, examples, pictures, etc., because, well, because *you're* a person. And your brain pays more attention to *people* than it does to *things*.



BULLET POINTS

Fireside Chats





Cut this out and stick it
on your refrigerator.

Here's what YOU can do to bend your brain into submission

So, we did our part. The rest is up to you. These tips are a starting point; listen to your brain and figure out what works for you and what doesn't. Try new things.

-
- ① **Slow down.** The more you understand, the less you have to memorize.
Don't just *read*. Stop and think. When the book asks you a question, don't just skip to the answer. Imagine that someone really *is* asking the question. The more deeply you force your brain to think, the better chance you have of learning and remembering.
 - ② **Do the exercises.** Write your own notes. We put them in, but if we did them for you, that would be like having someone else do your workouts for you. And don't just *look* at the exercises. **Use a pencil.** There's plenty of evidence that physical activity *while* learning can increase the learning.
 - ③ **Read the "There are No Dumb Questions"** That means all of them. They're not optional sidebars—***they're part of the core content!*** Don't skip them.
 - ④ **Make this the last thing you read before bed.** Or at least the last challenging thing. Part of the learning (especially the transfer to long-term memory) happens *after* you put the book down. Your brain needs time on its own, to do more processing. If you put in something new during that processing time, some of what you just learned will be lost.
 - ⑤ **Drink water.** Lots of it. Your brain works best in a nice bath of fluid. Dehydration (which can happen before you ever feel thirsty) decreases cognitive function.
 - ⑥ **Talk about it.** Out loud. Speaking activates a different part of the brain. If you're trying to understand something, or increase your chance of remembering it later, say it out loud. Better still, try to explain it out loud to someone else. You'll learn more quickly, and you might uncover ideas you hadn't known were there when you were reading about it.
 - ⑦ **Listen to your brain.** Pay attention to whether your brain is getting overloaded. If you find yourself starting to skim the surface or forget what you just read, it's time for a break. Once you go past a certain point, you won't learn faster by trying to shove more in, and you might even hurt the process.
 - ⑧ **Feel something.** Your brain needs to know that this *matters*. Get involved with the stories. Make up your own captions for the photos. Groaning over a bad joke is *still* better than feeling nothing at all.
 - ⑨ **Write a lot of software!** There's only one way to learn to program: **writing a lot of code.** And that's what you're going to do throughout this book. Coding is a skill, and the only way to get good at it is to practice. We're going to give you a lot of practice: every chapter has exercises that pose a problem for you to solve. Don't just skip over them—a lot of the learning happens when you solve the exercises. We included a solution to each exercise—don't be afraid to **peek at the solution** if you get stuck! (It's easy to get snagged on something small.) But try to solve the problem before you look at the solution. And definitely get it working before you move on to the next part of the book.

What you need for this book:

We wrote this book using Visual C# 2008 Express Edition, which uses C# 3.0 and .NET Framework 3.5. All of the screenshots that you see throughout the book were taken from that edition, so we recommend that you use it. If you're using Visual Studio 2008 Standard, Professional, or Team System editions, you'll see some small differences, which we've pointed out wherever possible. You can download the Express Edition for free from Microsoft's website—it installs cleanly alongside other editions, as well as previous versions of Visual Studio.

SETTING UP VISUAL STUDIO 2008 EXPRESS EDITION

- It's easy enough to download and install Visual C# 2008 Express Edition. Here's the link to the Visual Studio 2008 Express Edition download page:

<http://www.microsoft.com/express/download/>

Make sure that you check all of the options when you install it.



If you absolutely must use an older version of Visual Studio, C# or the .NET Framework, then please keep in mind that you'll come across topics in this book that won't be compatible with your version. The C# team at Microsoft has added some pretty cool features to the language. We'll give you warnings when we talk about any of these topics. But definitely keep in mind that if you're not using the latest version, there will be some code in this book that won't work.

- Download the installation package for Visual C# 2008 Express Edition. Make sure you do a complete installation. That should install everything that you need: the IDE (which you'll learn about), SQL Server Express Edition, and .NET Framework 3.5.
- Once you've got it installed, you'll have a new Start menu option: *Microsoft Visual C# 2008 Express Edition*. Click on it to bring up the IDE, and you're all set.

Read me

This is a learning experience, not a reference book. We deliberately stripped out everything that might get in the way of learning whatever it is we're working on at that point in the book. And the first time through, you need to begin at the beginning, because the book makes assumptions about what you've already seen and learned.

The activities are NOT optional.

The exercises and activities are not add-ons; they're part of the core content of the book. Some of them are to help with memory, some for understanding, and some to help you apply what you've learned. **Don't skip the written problems.** The pool puzzles are the only things you don't *have* to do, but they're good for giving your brain a chance to think about twisty little logic puzzles.

The redundancy is intentional and important.

One distinct difference in a Head First book is that we want you to *really* get it. And we want you to finish the book remembering what you've learned. Most reference books don't have retention and recall as a goal, but this book is about *learning*, so you'll see some of the same concepts come up more than once.

Do all the exercises!

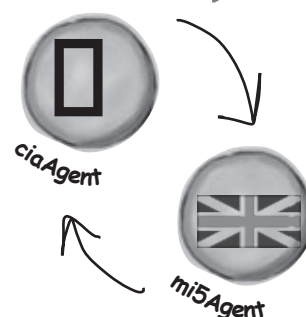
The one big assumption that we made when we wrote this book is that you want to learn how to program in C#. So we know you want to get your hands dirty right away, and dig right into the code. We gave you a lot of opportunities to sharpen your skills by putting exercises in every chapter. We've labeled some of them "Do this!"—when you see that, it means that we'll walk you through all of the steps to solve a particular problem. But when you see the Exercise logo with the running shoes, then we've left a big portion of the problem up to you to solve, and we gave you the solution that we came up with. Don't be afraid to peek at the solution—**it's not cheating!** But you'll learn the most if you try to solve the problem first.

We've also placed all the exercise solutions' source code on the web so you can download it. You'll find it at <http://www.headfirstlabs.com/books/hfcsharp/>

The "Brain Power" exercises don't have answers.

For some of them, there is no right answer, and for others, part of the learning experience of the Brain Power activities is for you to decide if and when your answers are right. In some of the Brain Power exercises you will find hints to point you in the right direction.

We use a lot of diagrams to make tough concepts easier to understand.



You should do ALL of the "Sharpen your pencil" activities



Activities marked with the Exercise (running shoe) logo are really important! Don't skip them if you're serious about learning C#.



If you see the Pool Puzzle logo, the activity is optional, and if you don't like twisty logic, you won't like these either.



The technical review team

Lisa Kellner



Joe Albahari



Jay Hilyard



Daniel Kinnaer



Not pictured (but
just as awesome):
Wayne Bradney,
Dave Murdoch,
and Bridgette
Julie Landers

Aayam Singh



Theodore Casser



Andy Parker



Peter Ritchie



Bill Mietelski



Technical Reviewers:

When we wrote this book, it had a bunch of mistakes, issues, problems, typos, and terrible arithmetic errors. Okay, it wasn't quite that bad. But we're still really grateful for the work that our technical reviewers did for the book. We would have gone to press with errors (including one or two big ones) had it not been for the most kick-ass review team EVER...

First of all, we really want to thank **Joe Albahari** for the enormous amount of technical guidance. He really set us straight on a few really important things, and if it weren't for him you'd be learning incorrect stuff. We also want to thank **Lisa Kellner**—this is our third book that she's reviewed for us, and she made a huge difference in the readability of the final product. Thanks, Lisa! And special thanks to **Jay Hilyard** and **Daniel Kinnaer** for catching and fixing a whole lot of our mistakes, and **Aayam Singh** for actually going through and doing every one of these exercises **before** we fixed them and corrected their problems. Aayam, you're really dedicated. Thanks!

And special thanks to our favorite readers, David Briggs and Jaime Moreno, for going above and beyond the call of duty by finding and reporting many errors that we didn't catch in the first printing, and to Jon Skeet for going through the whole book carefully and helping us fix a bunch of errors.

Krishna Pala



Giuseppe Turitto

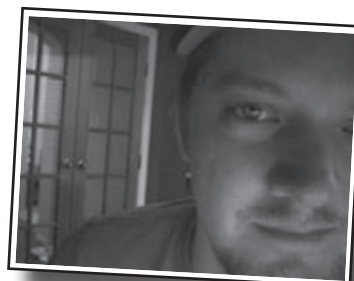


Acknowledgments

Our editor:

We want to thank our editor, **Brett McLaughlin**, for editing this book. He helped with a lot of the narrative, and the comic idea in Chapter 14 was completely his, and we think it turned out really well. Thanks, Brett!

Brett McLaughlin



The O'Reilly team:



Lou Barr

Lou Barr is an amazing graphic designer who went above and beyond on this one, putting in unbelievable hours and coming up with some pretty amazing visuals. If you see anything in this book that looks fantastic, you can thank her (and her mad InDesign skillz) for it. She did all of the monster and alien graphics for the labs, and the entire comic book. Thanks so much, Lou! You are our hero, and you're awesome to work with.



Sanders Kleinfeld

There are so many people at O'Reilly we want to thank that we hope we don't forget anyone. First of all, the Head First team rocks—**Laurie Petrycki**, **Catherine Nolan**, **Sanders Kleinfeld** (the most super production editor ever!), **Caitrin McCullough**, **Keith McNamara**, and **Brittany Smith**. Special thanks to **Colleen Gorman** for her sharp proofread, **Ron Bilodeau** for volunteering his time and preflighting expertise, and **Adam Witwer** for offering one last sanity check—all of whom helped get this book from production to press in record time. And as always, we love **Mary Treseler**, and can't wait to work with her again! And a big shout out to our other friends and editors, **Andy Oram**, **Isabel Kunkle**, and **Mike Hendrickson**. And if you're reading this book right now, then you can thank the greatest publicity team in the industry: **Marsee Henon**, **Sara Peyton**, **Mary Rotman**, **Jessica Boyd**, **Kathryn Barrett**, and the rest of the folks at Sebastopol.

Safari® Books Online



When you see a Safari® icon on the cover of your favorite technology book that means the book is available online through the O'Reilly Network Safari Bookshelf.

Safari offers a solution that's better than e-books. It's a virtual library that lets you easily search thousands of top tech books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it for free at <http://safari.oreilly.com>.

1 get productive with c#

Visual Applications, in 10 minutes or less

Don't worry, Mother. With Visual Studio and C#, you'll be able to program so fast that you'll never burn the pot roast again.



Want to build great programs really fast?

With C#, you've got a **powerful programming language** and a **valuable tool** at your fingertips. With the **Visual Studio IDE**, you'll never have to spend hours writing obscure code to get a button working again. Even better, you'll be able to **focus on getting your work done**, rather than remembering which method parameter was for the *name* for a button, and which one was for its *label*. Sound appealing? Turn the page, and let's get programming.

Why you should learn C#

C# and the Visual Studio IDE make it easy for you to get to the business of writing code, and writing it fast. When you're working with C#, the IDE is your best friend and constant companion.

Here's what the IDE automates for you...

Every time you want to get started writing a program, or just putting a button on a form, your program needs a whole bunch of repetitive code.

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
namespace A_New_Program
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

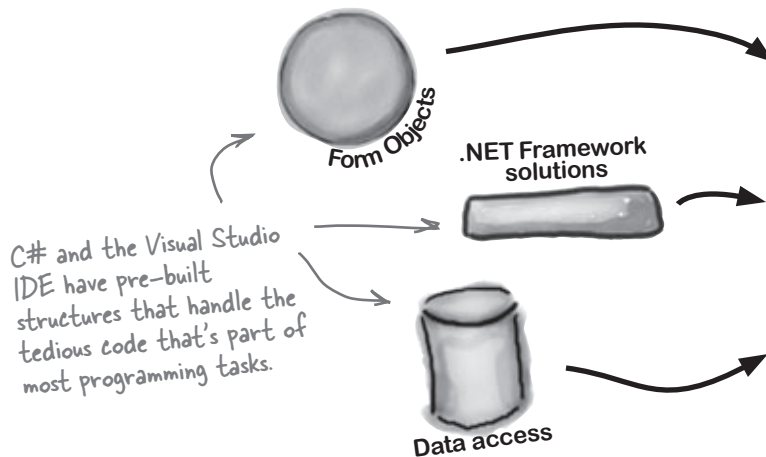
```
private void InitializeComponent()
{
    this.button1 = new System.Windows.Forms.Button();
    this.SuspendLayout();
    //
    // button1
    //
    this.button1.Location = new System.Drawing.Point(105, 56);
    this.button1.Name = "button1";
    this.button1.Size = new System.Drawing.Size(75, 23);
    this.button1.TabIndex = 0;
    this.button1.Text = "button1";
    this.button1.UseVisualStyleBackColor = true;
    this.button1.Click += new System.EventHandler(this.button1_Click);
    //
    // Form1
    //
    this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.None;
    this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
    this.ClientSize = new System.Drawing.Size(292, 267);
    this.Controls.Add(this.button1);
    this.Name = "Form1";
    this.Text = "Form1";
    this.ResumeLayout(false);
}
```

↖ The IDE—or Visual Studio Integrated Development Environment—is an important part of working in C#. It's a program that helps you edit your code, manage your files, and publish your projects.

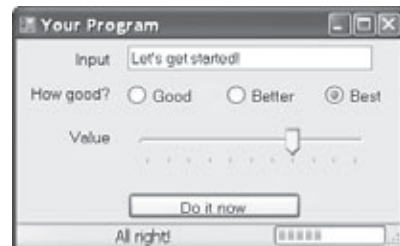
↖ It takes all this code just to draw a button on a form. Adding a few more visual elements to the form could take ten times as much code.

What you get with Visual Studio and C#...

With a language like C#, tuned for Windows programming, and the Visual Studio IDE, you can focus on what your program is supposed to **do** immediately:



↖ The result is a better looking application that takes less time to write.



C# and the Visual Studio IDE make lots of things easy

When you use C# and Visual Studio, you get all of these great features, without having to do any extra work. Together, they let you:

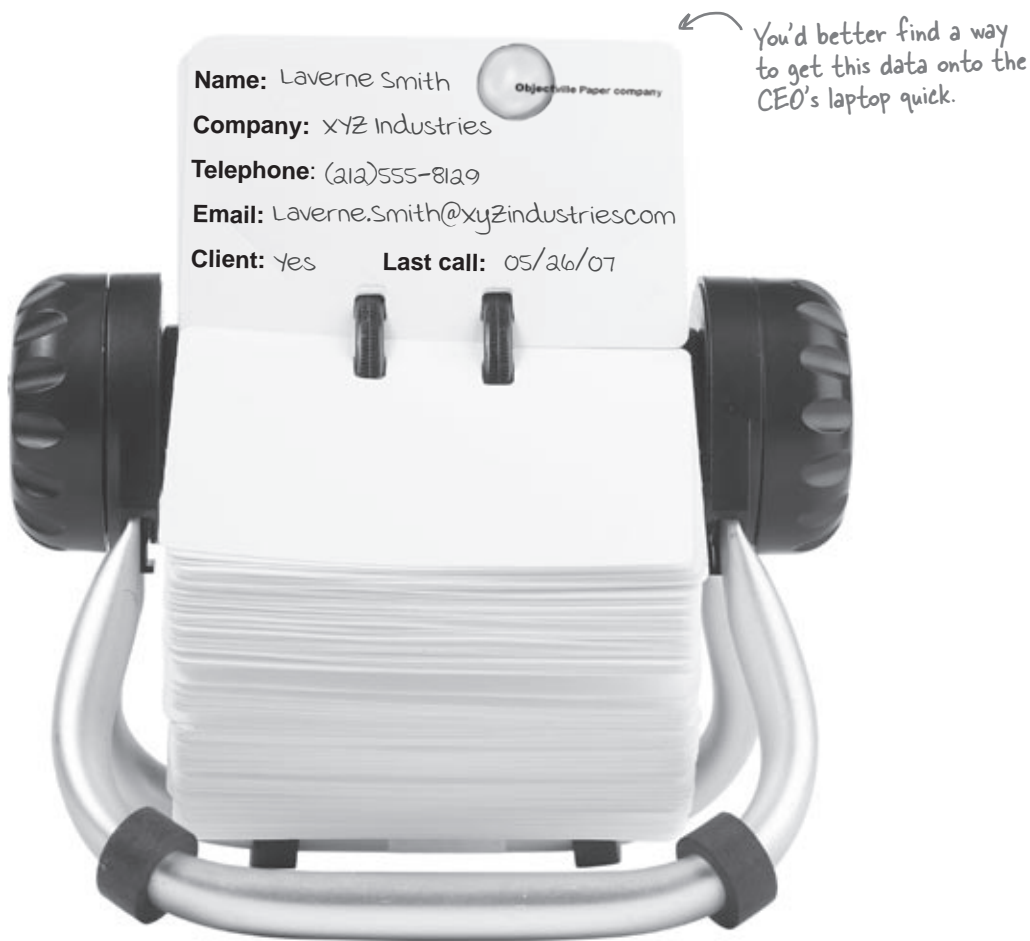
- ❶ **Build an application, FAST.** Creating programs in C# is a snap. The language is powerful and easy to learn, and the Visual Studio IDE does a lot of work for you automatically. You can leave mundane coding tasks to the IDE and focus on what your code should accomplish.
- ❷ **Design a great looking user interface.** The Form Designer in the Visual Studio IDE is one of the easiest design tools to use out there. It does so much for you that you'll find that making stunning user interfaces is one of the most satisfying parts of developing a C# application. You can build full-featured professional programs without having to spend hours writing a graphical user interface entirely from scratch.
- ❸ **Create and interact with databases.** The IDE includes a simple interface for building databases, and integrates seamlessly with SQL Server Express, as well as several other popular database systems.
- ❹ **Focus on solving your REAL problems.** The IDE does a lot for you, but *you* are still in control of what you build with C#. The IDE just lets you focus on your program, your work (or fun!), and your customers. But the IDE handles all the grunt work, such as:
 - ★ Keeping track of all of your projects
 - ★ Making it easy to edit your project's code
 - ★ Keeping track of your project's graphics, audio, icons, and other resources
 - ★ Managing and interacting with databases

All this means you'll have all the time you would've spent doing this routine programming to put into **building killer programs**.

↖ You're going to see exactly what we mean next.

Help the CEO go paperless

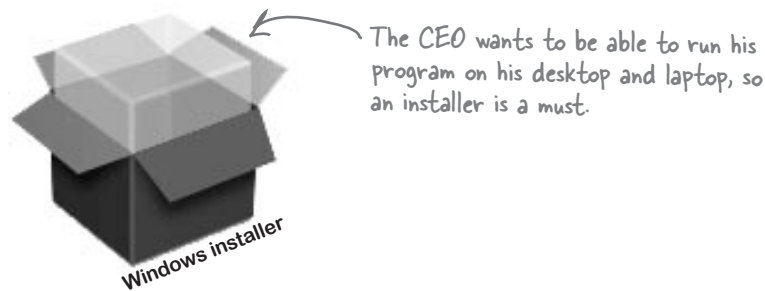
The Objectville Paper Company just hired a new CEO. He loves hiking, coffee, and nature... and he's decided that to help save forests. He wants to become a paperless executive, starting with his contacts. He's heading to Aspen to go ski for the weekend, and expects a new address book program by the time he gets back. Otherwise... well... it won't be just the old CEO who's looking for a job.



Get to know your users' needs before you start building your program

Before we can start writing the address book application—or *any* application—we need to take a minute and think about **who's going to be using it**, and **what they need** from the application.

- ① The CEO needs to be able to run his address book program at work and on his laptop too. He'll need an installer to make sure that all of the right files get onto each machine.



- ② The Objectville Paper company sales team wants to access his address book, too. They can use his data to build mailing lists and get client leads for more paper sales.

The CEO figures a database would be the best way that everyone in the company to see his data, and then he can just keep up with one copy of all his contacts.

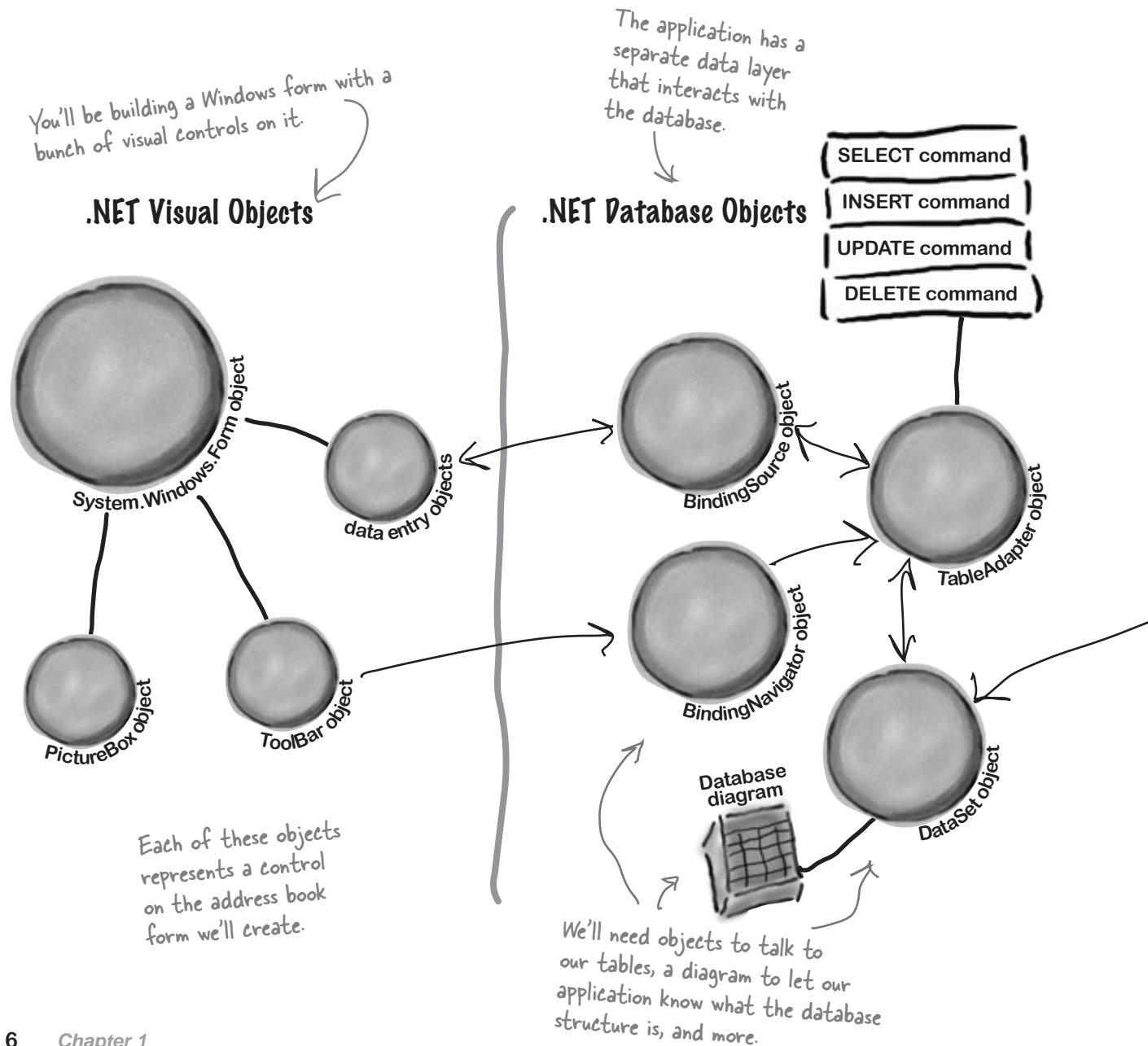
We already know that Visual C# makes working with databases easy. Having contacts in a database lets the CEO and the sales team all access the information, even though there's only one copy of the data.



Here's what you're going to build

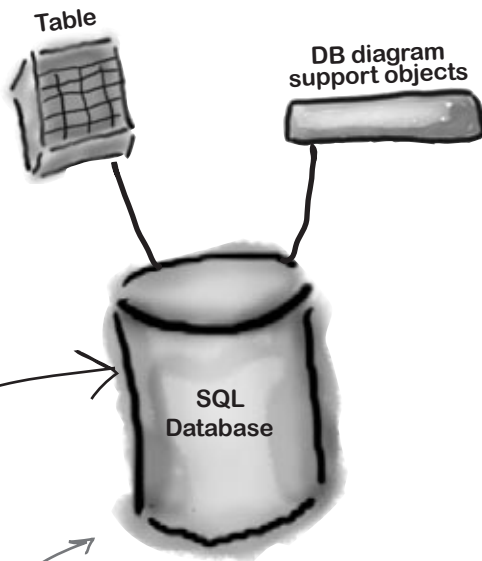
You're going to need an application with a graphical user interface, objects to talk to a database, the database itself, and an installer. It sounds like a lot of work, but you'll build all of this over the next few pages.

Here's the structure of the program we're going to create:



The data is all stored in a table in a SQL Server Express database.

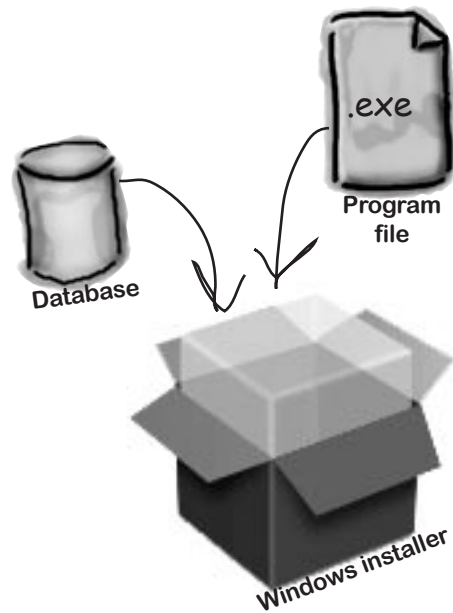
Data Storage



Here's the database itself, which Visual Studio will help us create and maintain.

Once the program's built, it'll be packaged up into a Windows installer.

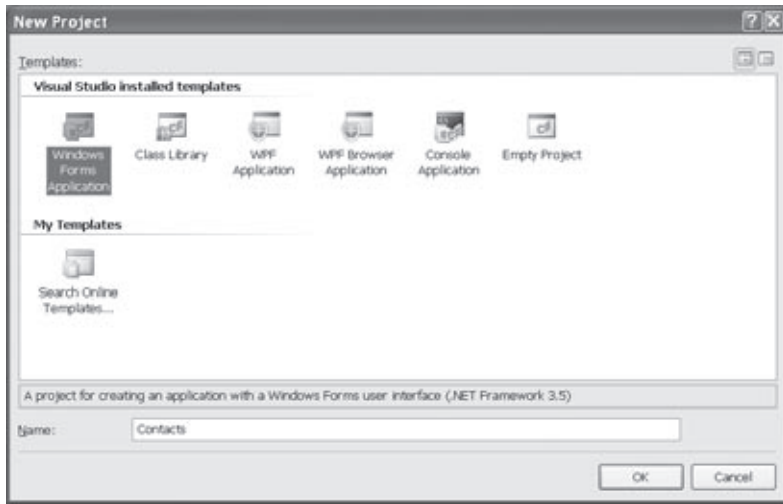
Deployment Package



The sales department will just need to point and click to install and then use his program.

What you do in Visual Studio...

Go ahead and start up Visual Studio, if you haven't already. Skip over the start page and select New Project from the **File** menu. Name your project "Contacts" and click OK.



Things may look a bit different in your IDE.

This is what the "New Project" window looks like in Visual Studio 2008 Express Edition. If you're using the Professional or Team Foundation edition, it might be a bit different. But don't worry, everything still works exactly the same.

What Visual Studio does for you...

As soon as you save the project, the IDE creates a Form1.cs, Form1.Designer.cs, and Program.cs file when you create a new project. It adds these to the Solution Explorer window, and by default, puts those files in My Documents\Visual Studio 2008\Projects\Contacts\.

Make sure that you save your project as soon as you create it by selecting "Save All" from the File menu—that'll save all of the project files out to the folder. If you select "Save", it just saves the one you're working on.

This file contains the C# code that defines the behavior of the form.



Form1.cs

This has the code that starts up the program and displays the form.



Program.cs

The code that defines the form and its objects lives here.



Form1.Designer.cs

Visual Studio creates all three of these files automatically.

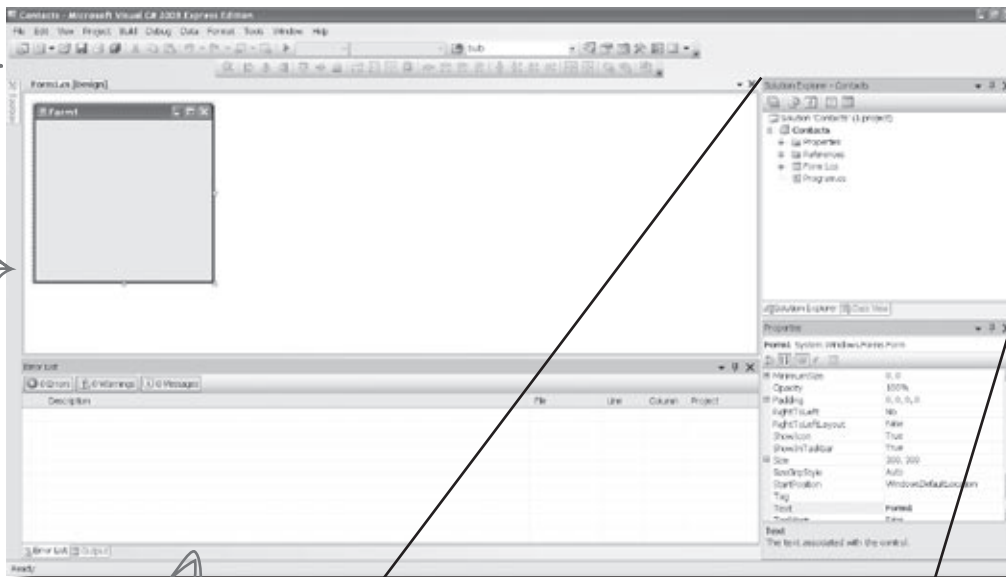


Below is what your screen probably looks like right now. You should be able to figure out what most of these windows and files are based on what you already know. In each of the blanks, try and fill in an annotation saying what that part of the IDE does. We've done one to get you started.

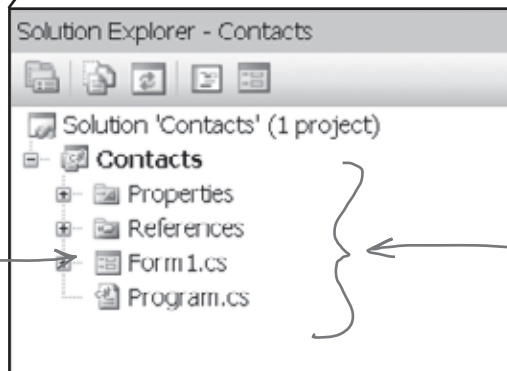
This toolbar has buttons that apply to what you're currently doing in the IDE.

If your IDE doesn't look exactly like this picture, you can select "Reset Window Layout" from the Window menu.

We've blown up this window below so you have more room.



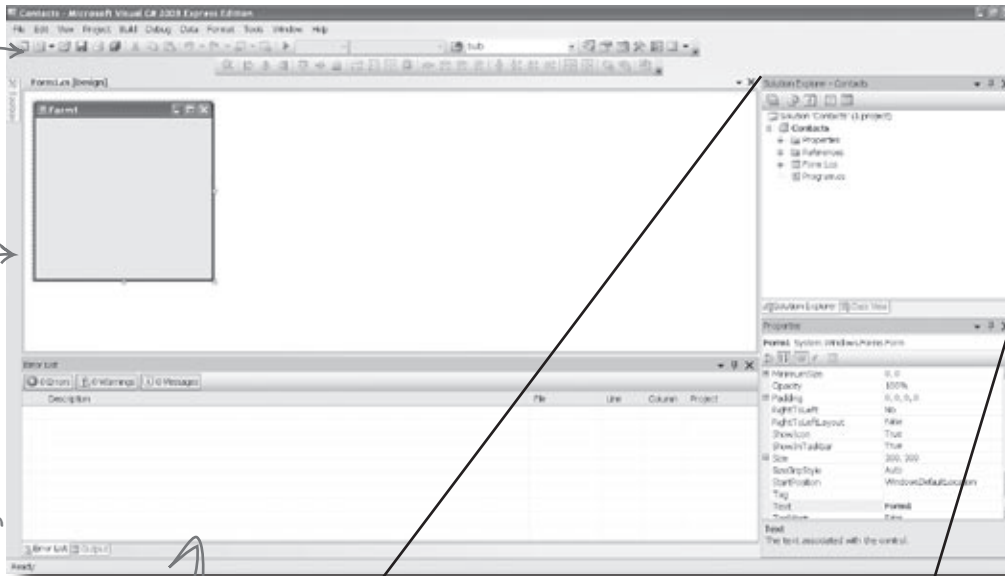
You can also bring up these windows by selecting Solution Explorer, Properties, or Error List from the View menu.



Sharpen your pencil Solution

We've filled in the annotations about the different sections of the Visual Studio C# IDE. You may have some different things written down, but you should have been able to figure out the basics of what each window and section of the IDE is used for.

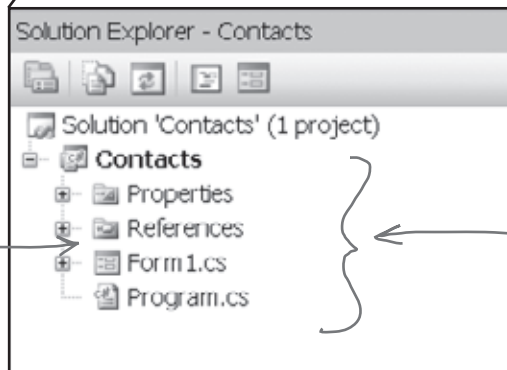
This toolbar has buttons that apply to what you're currently doing in the IDE.



This is the toolbox. It has a bunch of visual controls that you can drag onto your form.

This bottom pane is for debugging. It shows you when there are errors in your code.

The Form1.cs and Program.cs files that the IDE created for you when you added the new project appear in the Solution Explorer.



This window shows all of the properties of the controls on your form.

You can switch between files using the Solution Explorer in the IDE.

there are no Dumb Questions

Q: So if the IDE writes all this code for me, is learning C# just a matter of learning how to use the IDE?

A: No. The IDE is great at automatically generating some code for you, but it can only do so much. There are some things it's really good at, like setting up good starting points for you, and automatically changing properties of controls on your forms. But the hard part of programming—figuring what your program needs to do and making it do it—is something that no IDE can do for you. Even though the Visual Studio IDE is one of the most advanced development environments out there, it can only go so far. It's *you*—not the IDE—who writes the code that actually does the work.

Q: I created a new project in Visual Studio, but when I went into the “Projects” folder under My Documents, I didn't see it there. What gives?

A: First of all, you must be using Visual Studio 2008—in 2005, this doesn't happen. When you first create a new project in Visual Studio 2008, the IDE creates the project in your Local Settings\Application Data\Temporary Projects folder. When you save the project for the first time, it will prompt you for a new filename, and save it in the My Documents\Visual Studio 2008\Projects folder. If you try to open a new project or close the temporary one, you'll be prompted to either save or discard the temporary project.

Q: What if the IDE creates code I don't want in my project?

A: You can change it. The IDE is set up to create code based on the way the element you dragged or added is most commonly

used. But sometimes that's not exactly what you wanted. Everything the IDE does for you—every line of code it creates, every file it adds—can be changed, either manually by editing the files directly or through an easy-to-use interface in the IDE.

Q: Is it OK that I downloaded and installed Visual Studio Express? Or do I need to use one of the versions of Visual Studio that isn't free in order to do everything in this book?

A: There's nothing in this book that you can't do with the free version of Visual Studio (which you can download from Microsoft's website). The main differences between Express and the other editions (Professional and Team Foundation) aren't going to get in the way of writing C# and creating fully functional, complete applications.

Q: Can I change the names of the files the IDE generates for me?

A: Absolutely. When you create a new project, the IDE gives you a default form called Form1 (which has files called Form1.cs, Form1.Designer.cs and Form1.resx). But you can use the Solution Explorer to change the names of the files to whatever you want. By default, the names of the files are the same as the name of the form. If you change the names of the files, you'll be able to see in the Properties window that form will still be called Form1. You can change the name of the form by changing the “(Name)” line in the Properties window. If you do, the filenames won't change.

C# doesn't care what names you choose for your files or your forms (or any other part of the program). But if you choose good names, it makes your programs easier to work with. For now, don't worry about names—we'll talk a lot more about how to choose good names for parts of your program later on.

Q: I'm looking at the IDE right now, but my screen doesn't look like yours! It's missing some of the windows, and others are in the wrong place. What gives?

A: If you click on the “Reset Window Layout” command under the “Window” menu, the IDE will restore the default window layout for you. Then your screen will look just like the ones in this chapter.

Visual Studio will generate code you can use as a starting point for your applications.

Making sure the application does what it's supposed to do is still up to you.

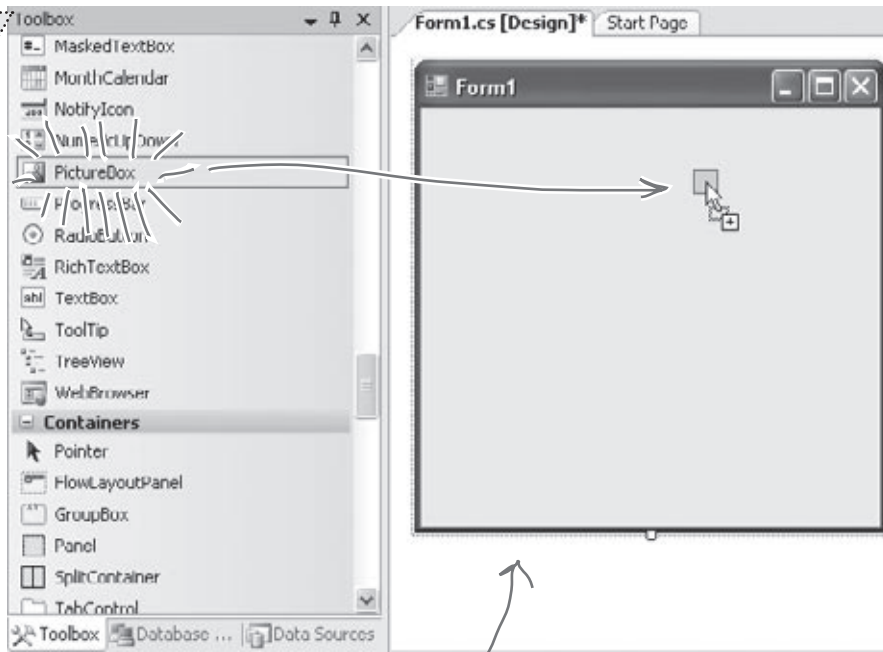
Develop the user interface

Adding controls and polishing the user interface is as easy as dragging and dropping with the Visual Studio IDE. Let's add a logo to the form:

1 Use the PictureBox control to add a picture.

Click on the PictureBox control in the Toolbox, and drag it onto your form. In the background, the IDE added code to `Form1.Designer.cs` for a new picture control.

If you don't see the toolbox, try hovering over the word "Toolbox" that shows up in the upper left-hand corner of the IDE. If it's not there, select "Toolbox" from the View menu to make it appear.



Every time you make a change to a control's properties on the form, the code in `Form1.Designer.cs` is getting changed by the IDE.

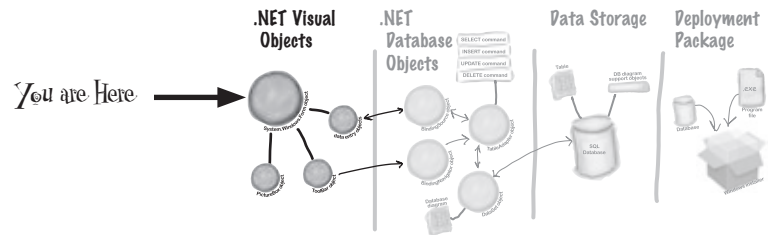


Form1.Designer.cs



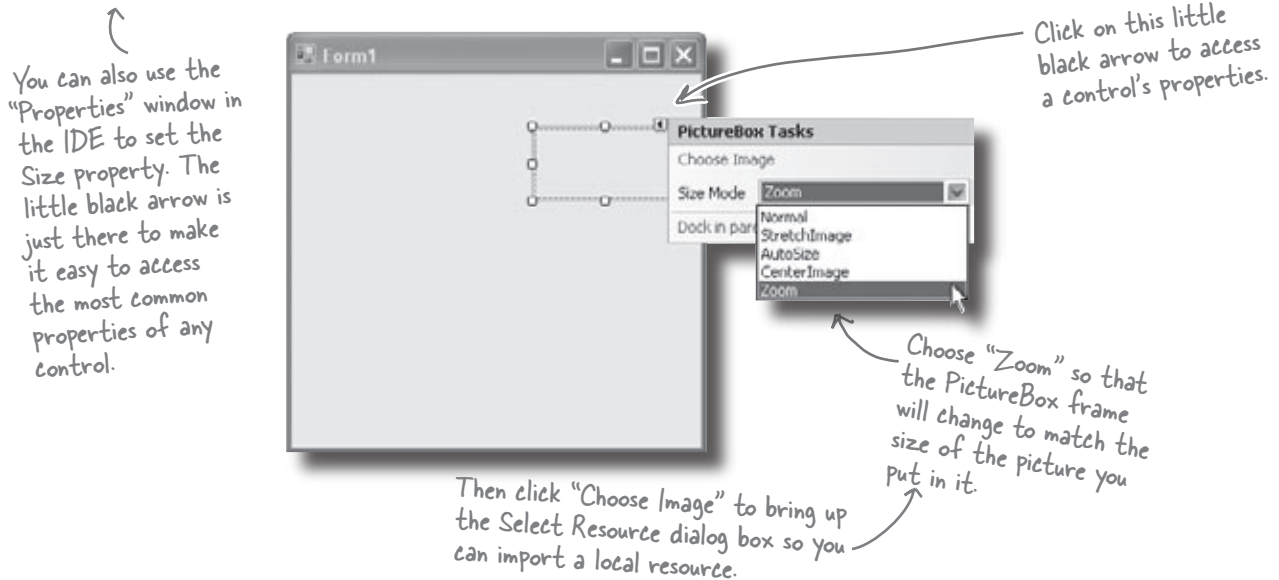
It's OK if you're not a pro at user interface design.

We'll talk a lot more about designing good user interfaces later on. For now, just get the logo and other controls on your form, and worry about **behavior**. We'll add some style later.



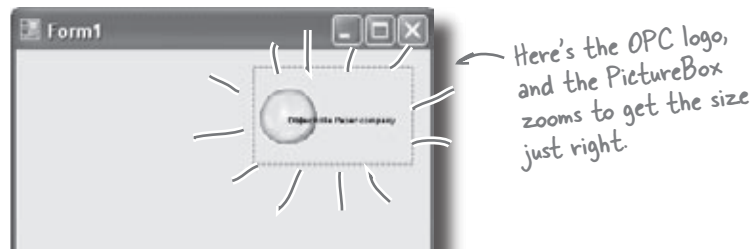
2 Set the PictureBox to Zoom mode.

Every control on your form has properties that you can set. Click the little black arrow for a control to access these properties. Change the PictureBox's Size property to "Zoom" to see how this works:



3 Download the Objectville Paper Company logo.

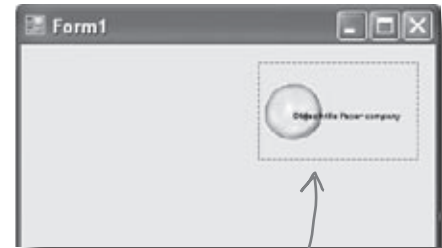
Download the Objectville Paper Co. logo from Head First Labs (<http://www.headfirstlabs.com/books/hfcsharp>) and save it to your hard drive. Then click the PictureBox properties arrow, and select Choose Image. You'll see a Select Resources window pop up. Click the "Local Resource" radio button to enable the "Import..." button at the top of the form. Click that button, find your logo, and you're all set.



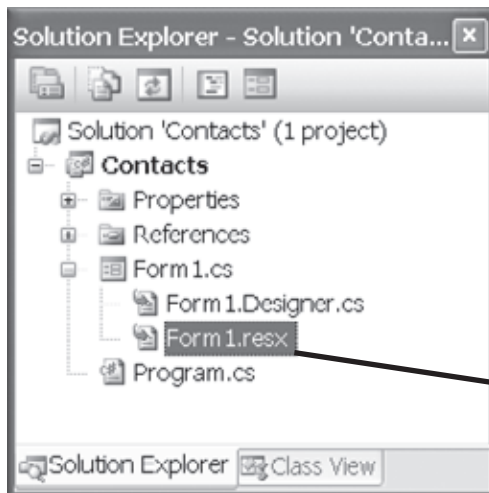
Visual Studio, behind the scenes

Every time you do something in the Visual Studio IDE, the IDE is **writing code for you**. When you created the logo and told Visual Studio to use the image you downloaded, Visual Studio created a resource and associated it with your application. A **resource** is any graphics file, audio file, icon, or other kind of data file that gets bundled with your application. The graphic file gets integrated into the program, so that when it's installed on another computer, the graphic is installed along with it and the PictureBox can use it.

When you dragged the PictureBox control onto your form, the IDE automatically created a resource file called Form1.resx to store that resource and keep it in the project. Double-click on this file, and you'll be able to see the newly imported image.



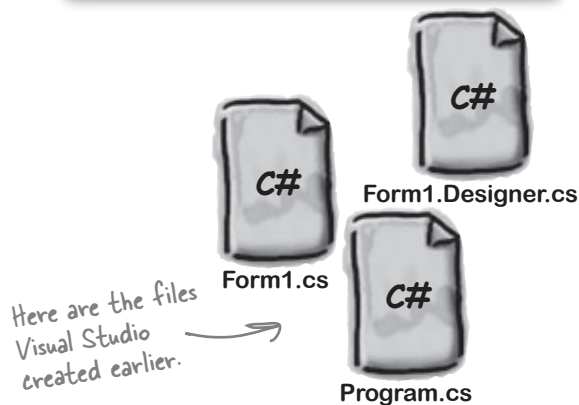
This image is now a resource of the Contact List application.



Go to the Solution Explorer and click on the plus icon next to Form1.cs to expand it (if it's not already expanded). This will display two files: Form1.Designer.cs and Form1.resx. Double-click on Form1.resx, click on the arrow next to "Strings", and select "Images" from the drop-down list (or hit Ctrl-2) to see the logo that you imported. That file is what links it to the PictureBox, and the IDE added code to do the linking.



If you chose the other 'Import...' button from the Select Resource dialog on the last page, then your image will show up in the Resources folder in the Solution Explorer instead. Don't worry—just go back to Select Resources, choose "Local Resource," and reimport the image into the resources, and it'll show up here.



When you imported the image, the IDE created this file for you. It contains all of the resources (graphics, video, audio and other stored data) associated with Form1.

Add to the auto-generated code

The IDE creates lots of code for you, but you'll still want to get into this code and add to it. Let's set the logo up to show an About message when the users run the program and click on the logo.

When you're editing a form in the IDE, double-clicking on any of the toolbox controls causes the IDE to automatically add code to your project. Make sure you've got the form showing in the IDE, and then double-click on the PictureBox control. The IDE will add code to your project that gets run any time a user clicks on the PictureBox. You should see some code pop up that looks like this:

```
public partial class Form1 : Form
{
```

```
    public Form1()
    {
```

```
        InitializeComponent();
    }
```

```
    private void pictureBox1_Click(object sender, EventArgs e)
    {
```

```
        MessageBox.Show("Contact List 1.0.\nWritten by: Your Name", "About");
    }
```

```
}
```

When you double-click on the PictureBox, it will open this code up with a cursor blinking right here. Ignore any windows the IDE pops up as you type; it's trying to help you, but we don't need that right now.

Type in this line of code. It causes a message box to popup with the text you provide. The box will be titled "About".

Once you've typed in the line of code, save it using the Save icon on the IDE toolbar or by selecting "Save" from the File menu. Get in the habit of doing "Save All" regularly!

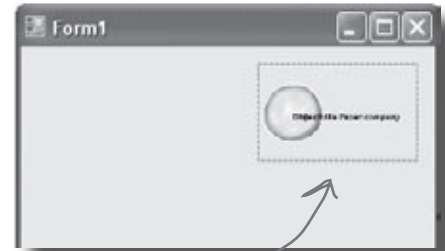
there are no Dumb Questions

Q: What's a method?

A: A **method** is just a *named block of code*. We'll talk a lot more about methods in Chapter 2.

Q: What does that \n thing do?


A: That's a line break. It tells C# to put "Contact List 1.0." on one line, and then start a new line for "Written by:".

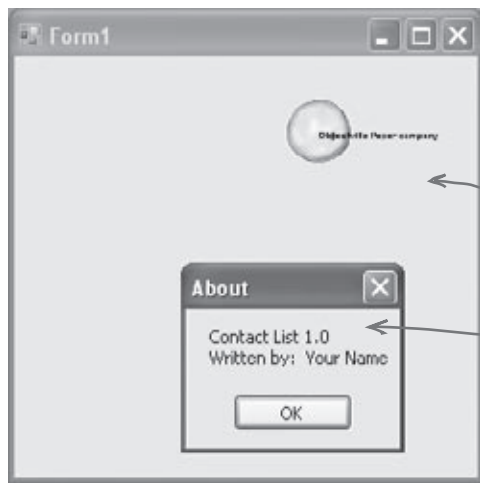


When you double-clicked on the PictureBox control, the IDE created this method. It will run every time a user clicks on the logo in the running application.

This method name gives you a good idea about when it runs: when someone clicks on this PictureBox control.

You can already run your application

Press the F5 key on your keyboard, or click the green arrow button (▶) on the toolbar to check out what you've done so far. (This is called "Debugging", which just means running your program using the IDE.) You can stop debugging by selecting "Stop Debugging" from the Debug menu or clicking this toolbar button: .



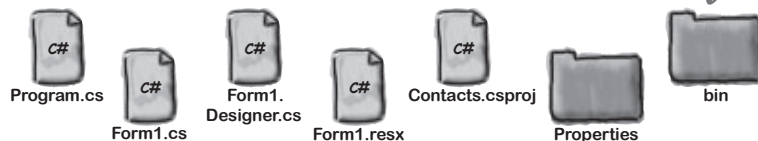
All three of these buttons work—and you didn't have to write any code to make them work.

Clicking on the OPC logo brings up the About box you just coded.

Where are my files?

When you run your program, Visual Studio copies all of your files to My Documents\Visual Studio 2008\Projects\Contacts\Contacts\bin\debug. You can even hop over to that directory and run your program by double-clicking on the .exe file the IDE creates.

C# turns your program into a file that you can run, called an **executable**. You'll find it in here, in the debug folder.



This isn't a mistake; there are two levels of folders. The inner folder has the actual C# code files.

there are no Dumb Questions

Q: In my IDE, the green arrow is marked as "Debug". Is that a problem?

A: No. Debugging, at least for our purposes right now, just means running your application inside the IDE. We'll talk a lot more about debugging later, but for now, you can simply think about it as a way to run your program.

Q: I don't see the Stop Debugging button on my toolbar. What gives?

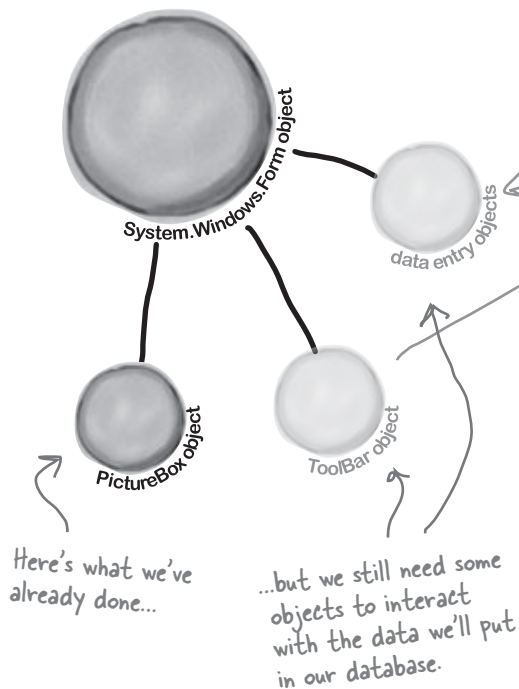
A: The Stop Debugging button only shows up in a special toolbar that **only shows up** when your program is running. Try starting the application again, and see if it appears.

Here's what we've done so far

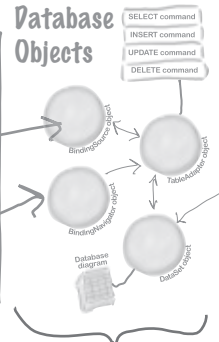
We've built a form and created a PictureBox object that pops up a message box when it's clicked on. Next, we need to add all the other fields from the card, like the contact's name and phone number.

Let's store that information in a database. Visual Studio can connect fields directly to that database for us, which means we don't have to mess with lots of database access code (which is good). But for that to work, we need to create our database so that the controls on the form can hook up to it. So we're going to jump from the .NET Visual Objects straight to the Data Storage section.

.NET Visual Objects

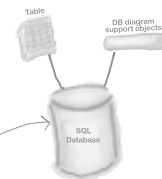


.NET Database Objects



This step is about connecting our form to the database, so we're not ready for it yet, since we don't have a database.

Data Storage



Deployment Package



So we need to focus on this step next: creating our database, and putting some initial data into it.

Visual Studio can generate code to connect your form to a database, but you need to have the database in place **BEFORE** generating that code.

We need a database to store our information

Before we add the rest of the fields to the form, we need to create a database to hook the form up to. The IDE can create lots of the code for connecting our form to our data, but we need to define the database itself first.

Make sure you've stopped debugging before you continue.

1 Add a new SQL database to your project.

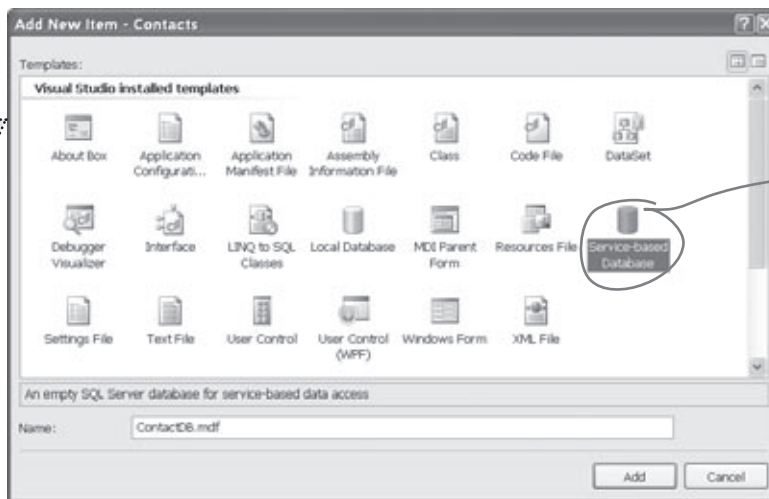
In the Solution Explorer, **right-click the Contacts project**, select **Add**, and then choose **New Item**. Choose the **SQL Database** icon, and name it **ContactDB.mdf**.

This file is our new database.



ContactDB.mdf

Pick the right icon for the version you're using. Choose **SQL Database** if you're using Visual Studio Express 2005 and **Service-Based Database** if you're using 2008.



The SQL Database icon only works if you have SQL Server Express installed. Flip back to the README if you're not sure how to do this.

2 Click on the Add button in the Add New Item window.

3 Cancel the Data Source Configuration Wizard.

For now, we want to skip configuring a data source, so click the **Cancel** button. We'll come back to this once we've set up our database structure.

4 View your database in the Solution Explorer.

Go to the Solution Explorer, and you'll see that **ContactDB** has been added to the file list. Double click **ContactDB.mdf** in the Solution Explorer and look at the left side of your screen. The Toolbox has changed to a **Database Explorer**.



Watch it!

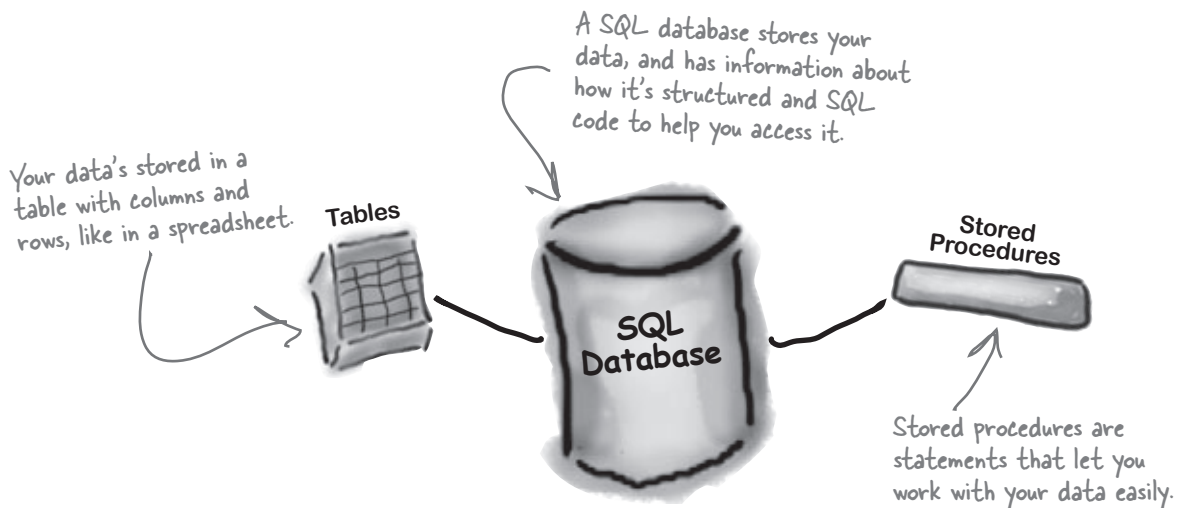
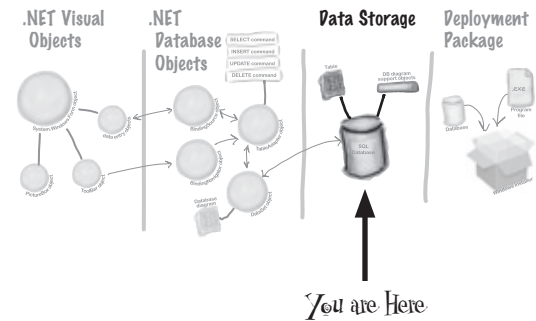
If you're not using the Express edition, you'll see "Server Explorer" instead of "Database Explorer".

The Visual Studio 2008 Professional and Team Foundation editions don't have a Database Explorer window. Instead, they have a Server Explorer window, which does everything the Database Explorer does, but also lets you explore data on your network.

The IDE created a database

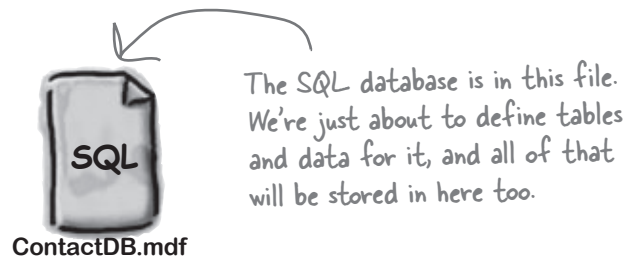
When you told the IDE to add a new SQL database to your project, the IDE created a new database for you. A **SQL database** is a system that stores data for you in an organized, interrelated way. The IDE gives you all the tools you need to maintain your data and databases.

Data in a SQL database lives in tables. For now, you can think of a table like a spreadsheet. It organizes your information into columns and rows. The columns are the data categories, like a contact's name and phone number, and each row is the data for one contact card.



SQL is its own language

SQL stands for **Structured Query Language**. It's a programming language for accessing data in databases. It's got its own syntax, keywords, and structure. SQL code takes the form of **statements** and **queries**, which access and retrieve the data. A SQL database can hold **stored procedures**, which are a bunch of SQL statements and queries that are stored in the database and can be run at any time. The IDE generates SQL statements and stored procedures for you automatically to let your program access the data in the database.

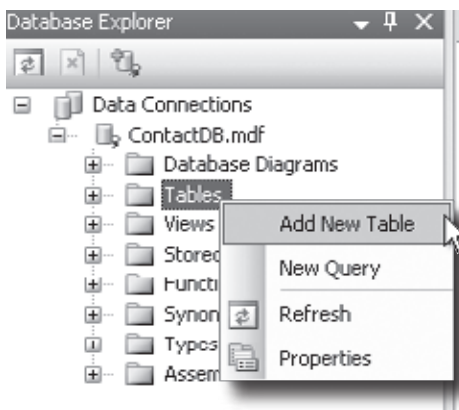


← [note from marketing: Can we get a plug for Head First SQL in here?]

Creating the table for the Contact List

We have a database, and now we need to store information in it. But our information actually has to go into a table, the data structure that databases use to hold individual bits of data. For our application, let's create a table called "People" to store all the contact information:

- 1 Add a table to the ContactDB database.**
Right click on Tables in the Database Explorer, and select Add New Table. This will open up a window where you can define the columns in the table you just created.



Now we need to add columns to our table. First, let's add a column called ContactID to our new People table, so that each Contact record has its own unique ID.

- 2 Add a ContactID column to the People table.**
Type "ContactID" in the Column Name field, and select Int from the Data Type dropdown box. Be sure to uncheck the Allow Nulls checkbox.

Finally, let's make this the primary key of our table. Highlight the ContactID column you just created, and click the Primary Key button. This tells the database that each entry will have a unique primary key entry.



↑
This is the Primary Key button. A primary key helps your database look up records quickly.

there are no Dumb Questions

Q: What's a column again?

A: A column is one field of a table. So in a People table, you might have a FirstName and LastName column. It will always have a data type, too, like String or Date or Bool.

Q: Why do we need this ContactID column?

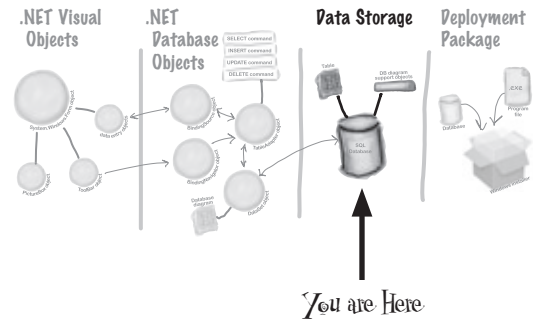
A: It helps to have a unique ID for each record in most database tables. Since we're storing contact information for individual people, we decided to create a column for that, and call it ContactID.

Q: What's that Int from Data Type mean?

A: The data type tells the database what type of information to expect for a column. Int stands for integer, which is just a whole number. So the ContactID column will have whole numbers in it.

Q: This is a lot of stuff. Should I be getting all of this?

A: No, it's OK if you don't understand everything right now. Focus on the basic steps, and we'll spend a lot more time on databases in the later chapters of the book. And if you're dying to know more right away, you can always pick up *Head First SQL* to read along with this book.



3 Tell the database to auto-generate IDs.

Since ContactID is a number for the database, and not our users, we can tell our database to handle creating and assigning IDs for us automatically. That way, we don't have to worry about writing any code to do this.

In the properties below your table, scroll down to Identity Specification, click the + button, and select Yes next to the (Is Identity) property.

This window is what you use to define your table and the data it will store.

The screenshot shows the 'Table Designer' window for a table named 'ContactID'. The table has one column, 'ContactID', with a data type of 'int' and 'Allow Nulls' set to 'No'. In the 'Column Properties' pane, the 'Identity Specification' property is expanded, showing a list of properties: 'DTS published' (No), 'Full-text Specification' (No), 'Has Non-SQL Server Subscriber' (No), 'Identity Specification' (Yes), '(Is Identity)' (Yes), 'Identity Increment' (Yes), and 'Identity Seed' (No). The '(Is Identity)' property is highlighted.

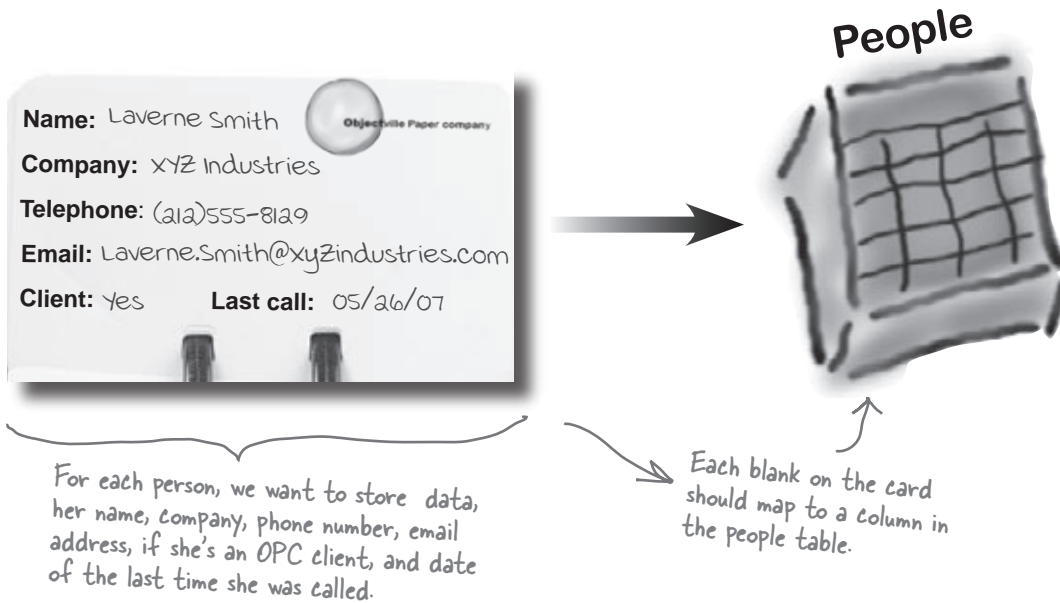
It's important that you leave this unchecked. Since the primary key is the main way your program will locate records, it always needs to have a value.

This will make it so that the ContactID field updates automatically whenever a new record is added.

You'll need to click on the right column and select Yes from the dropdown next to IsIdentity to designate ContactID as your record Identifier.

The blanks on contact card are columns in our People table

Now that you've created a primary key for the table, you need to define all of the fields you're going to track in the database. Each field on our written contact card should become a column in the People table.



What kinds of problems could result from having multiple rows stored for the same person?

WHO DOES WHAT?

Now that you've created a People table and a primary key column, you need to add columns for all of the data fields. See if you can work out which data type goes with each of the columns in your table, and also match the data type to the right description.

Column Name	Data Type	Description
Last Call	int	This type stores a date and time
Name	bit	A Boolean true/false type
ContactID	nvarchar(50)	A string of letters, numbers and other characters with a maximum length of 50
Client?	datetime	A whole number

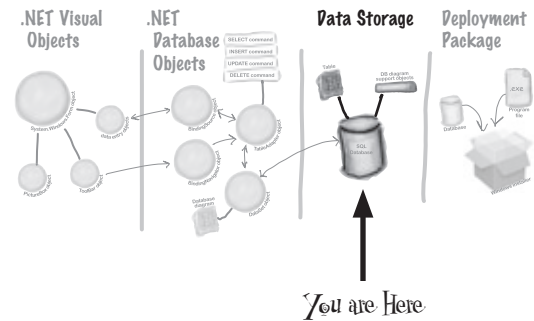
WHO DOES WHAT?

Now that you've created a People table and a primary key column, you need to add columns for all of the data fields. See if you can work out which data type goes with each of the columns in your table, and also match the data type to the right description.

Column Name	Data Type	Description
Last Call	int	This type stores a date and time
Name	bit	A Boolean true/false type
ContactID	nvarchar(50)	A string of letters, numbers and other characters with a maximum length of 50
Client?	datetime	A whole number

Finish building the table

Go back to where you entered the ContactID column and add the other five columns from the contact card. Here's what your database table should look like when you're done:



Bit fields hold True or False values and can be represented as a checkbox.

If you uncheck Allow Nulls, the column must have a value.

Some cards might have some missing information, so we'll let certain columns be blank.

Column Name	Data Type	Allow Nulls
ContactID	int	<input type="checkbox"/>
Name	nvarchar(50)	<input checked="" type="checkbox"/>
Company	nvarchar(50)	<input checked="" type="checkbox"/>
Telephone	nvarchar(50)	<input checked="" type="checkbox"/>
Email	nvarchar(50)	<input checked="" type="checkbox"/>
Client	bit	<input checked="" type="checkbox"/>
LastCall	datetime	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

Click on the Save button on the toolbar to save your new table. You'll be asked for a name. Call it "People" and click OK.

We've been talking about this table as the "People" table, but it's not until this step that you give it an official name.

This creates a People table, which goes in the ContactDB database.

you are here ▶

Diagram your data

The Visual Studio IDE is built to work with databases, and it comes with a lot of built-in tools that help you when you're handling a lot of data. One of the most powerful tools you have is the **database diagram**, which you can use to view and edit complex relationships between the tables in your database. So let's go ahead and build a database diagram for your database.

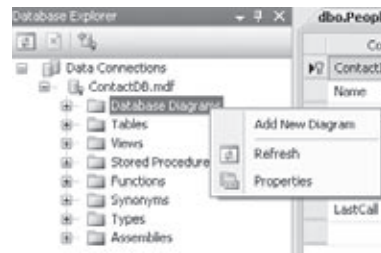


In very rare cases, a few people sometimes have problems getting the SQL database to work.

If you run into any trouble, don't worry—go to the Head First C# forum at <http://www.headfirstlabs.com/> for help troubleshooting the problem.

1 Create a new database diagram.

Go to the Database Explorer window and right-click on the Database Diagrams node. Select Add New Diagram.



2 Let the IDE generate access code.

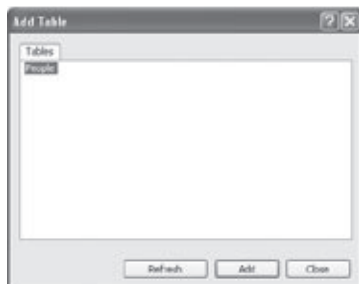
Before you tell the IDE about your specific table, it needs to create some basic stored procedures for interacting with your database. Just click Yes here, and let the IDE go to work.



Remember, these options are all under ContactDB, so they all apply to that specific database.

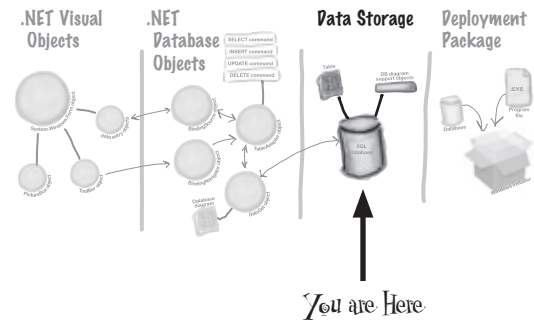
3 Select the tables you want to work with.

Select the People table from the window that pops up, and click Add. Now the IDE is ready to generate code specific to your table.



When you have databases with multiple tables, each table will show up as an entry on this window.

The IDE creates several stored procedures that allow your code to interact with the database you created.



4 Name your diagram PeopleDiagram.

Select File>Save Diagram. You'll be asked to name your new database diagram. Call it PeopleDiagram, and you're all set.

If you're using Visual Studio 2005, select File>Save All instead.

The database diagram is shown here visually. It's a very simple representation of your table.

People	
PK	ContactID
	Name
	Company
	Telephone
	Email
	Client
	LastCall

This is just a picture of the database design you've just done. It marks the ContactID field as your primary key and lists off all of the columns in the table.

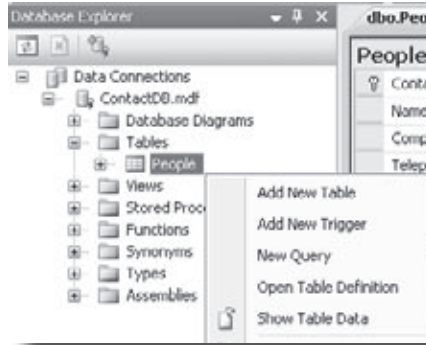
If you had any other tables in the database you wanted diagrammed, they would appear here, too.

A database diagram describes your tables to the Visual Studio IDE. The IDE will read your database and build a database diagram for you automatically.

Insert your card data into the database

Now you're ready to start entering cards into the database. Here are some of the boss's contacts—we'll use those to set up the database with a few records.

- 1 Expand Tables and then right click on the People Table in the Database Explorer (or Server Explorer) and select Show Table Data.
- 2 Once you see the Table grid in the main window, go ahead and add all of the data below. (You'll see all NULL values at first—just type over them when you add your first row. And ignore the exclamation points that appear next to the data.) You don't need to fill in the ContactID column, that happens automatically.



Your job is to enter the data from all six of these cards into the People table.

Type "True" or "False" in the Client column. That's how SQL stores yes or no info.

Name: Liz Nelson
Company: JTP
Telephone: (419)555-2578
Email: LizNelson@JTP.ORG
Client: Yes **Last call:** 03/04/06

Name: Lloyd Jones
Company: Black Box inc.
Telephone: (718)555-5638
Email: LJones@xblackboxinc.com
Client: Yes **Last call:** 05/26/07

Name: Lucinda Ericson
Company: Ericson Events
Telephone: (212)555-9523
Email: Lucy@ericsonerevents.info
Client: NO **Last call:** 05/17/07

Name: matt Franks
Company: xyz Industries
Telephone: (a1a)555-81a5
Email: matt.Franks@xyzindustries.com
Client: yes **Last call:** 05/26/07

Name: Sarah Kalter
Company: Kalter, Riddle, and Stoft
Telephone: (614)555-5641
Email: Sarah@KRS.org
Client: no **Last call:** 12/10/05

Name: Laverne Smith
Company: xyz Industries
Telephone: (a1a)555-81a9
Email: Laverne.Smith@xyzindustries.com
Client: yes **Last call:** 05/26/07

Objectville Paper Company is in the United States, so the CEO writes dates so that 05/26/07 means May 26, 2007. If your machine is set to a different location, you may need to enter dates differently; you might need to use 26/05/07 instead.

- ③ Once you've entered all six records, select Save All from the File menu again. That should save the records to the database.

"Save All" tells the IDE to save everything in your application. That's different from "Save", which just saves the file you're working on.

there are no Dumb Questions

Q: So what happened to the data after I entered it? Where did it go?

A: The IDE automatically stored the data you entered into the People table in your database. The table, its columns, the data types, and all of the data inside it is all stored in the SQL Server Express file, ContactDB.mdf. That file is stored as part of your project, and the IDE updates it just like it updates your code files when you change them.

Q: Okay, I entered these six records. Will they be part of my program forever?

A: Yes, they're as much a part of the program as the code that you write and the form that you're building. The difference is that instead of being compiled into an executable program, the ContactDB.mdf file is copied and stored along with the executable. When your application needs to access data, it reads and writes to ContactDB.mdf, in the program's output directory.

This file is actually a SQL database, and your program can use it with the code the IDE generated for you.



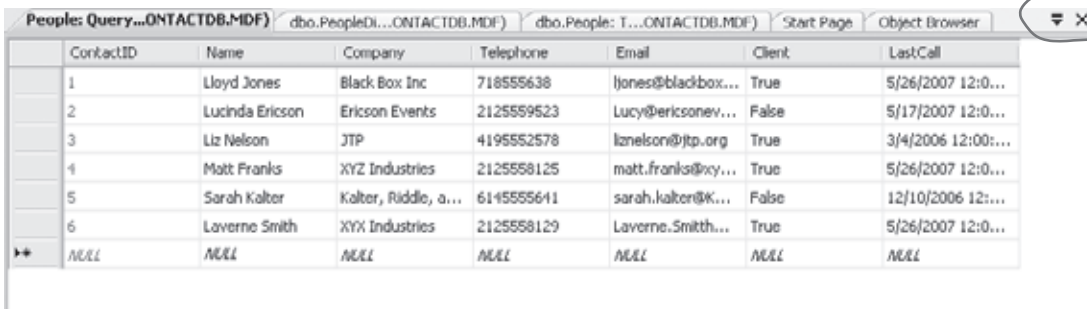
Connect your form to your database objects with a data source

We're finally ready to build the .NET database objects that our form will use to talk to your database. We need a **data source**, which is really just a collection of SQL statements your program will use to talk to the ContactDB database.

1 Go back to your application's form.

Close out the People table and the ContactDB database diagram. You should now have the Form1.cs [Design] tab visible.

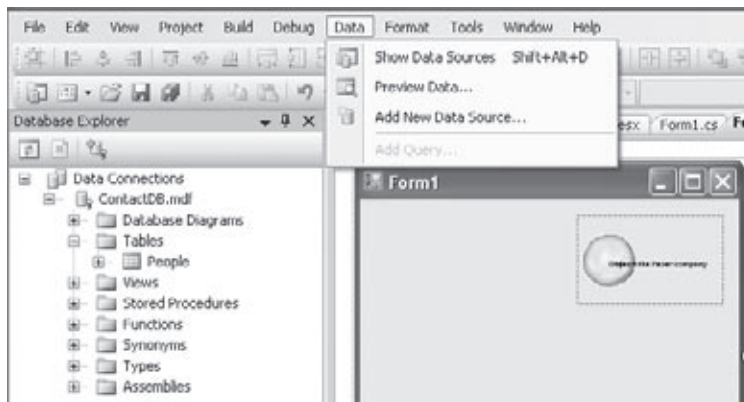
You need to close both the data grid and the diagram to get back to your form.



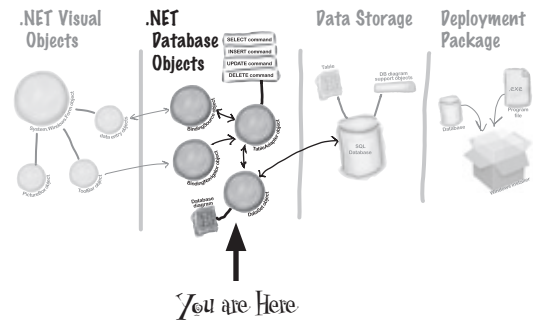
ContactID	Name	Company	Telephone	Email	Client	LastCall
1	Lloyd Jones	Black Box Inc	718555638	ljones@blackbox...	True	5/26/2007 12:0...
2	Lucinda Ericson	Ericson Events	2125559523	Lucy@ericson-ev...	False	5/17/2007 12:0...
3	Liz Nelson	JTP	4195552578	liznelson@jtp.org	True	3/4/2006 12:00...
4	Matt Franks	XYZ Industries	2125558125	matt.franks@xyz...	True	5/26/2007 12:0...
5	Sarah Kalter	Kalter, Riddle, a...	6145555641	sarah.kalter@K...	False	12/10/2006 12:...
6	Laverne Smith	XYZ Industries	2125558129	Laverne.Smith...	True	5/26/2007 12:0...
NULL	NULL	NULL	NULL	NULL	NULL	NULL

2 Add a new data source to your application.

This should be easy by now. Click the Data menu, and then select Add New Data Source... from the drop down.



The data source you're creating will handle all the interactions between your form and your database.



3 Configure your new data source.

Now you need to setup your data source to use the ContactDB database. Here's what to do:

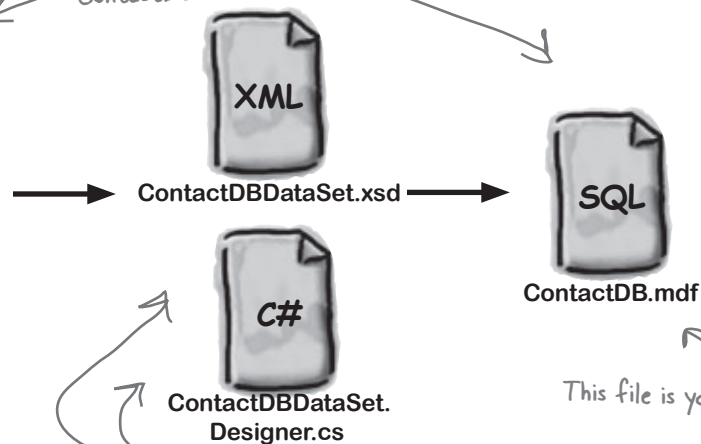
- ★ Select Database and click the Next button.
- ★ Click Next in the "Choose your Data Connection" screen.
- ★ Make sure the Save the connection checkbox is checked in the "Save the Connection" screen that follows and click Next.
- ★ In the "Choose Your Objects" screen, click the Table checkbox.
- ★ In the Dataset Name field, make sure it says "ContactDBDataSet" and click Finish.

These steps connect your new data source with the People table in the ContactDB database.



Here's your existing form.

Now your form can use the data source to interact with the ContactDB database.



These files are what's generated by the data source you just setup.

This file is your database.

Add database-driven controls to your form

Now we can go back to our form, and add some more controls. But these aren't just any controls, they are controls that are *bound* to our database, and the columns in the People table. That just means that a change to the data in one of the controls on the form automatically changes the data in the matching column in the database.

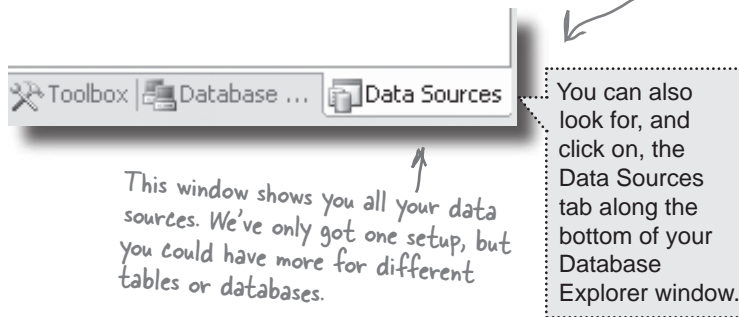
Here's how to create several database-driven controls:

1 Select the data source you want to use.

Select Show Data Sources from the Data pull down menu. This will bring up the Data Sources window, showing the sources you have setup for your application.

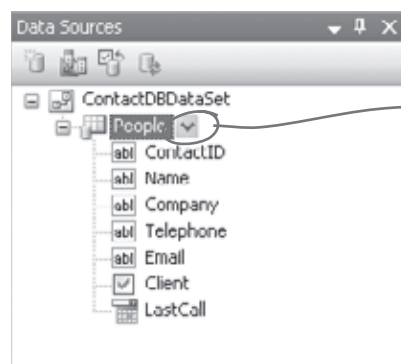
It took a little work, but now we're back to creating form objects that interact with our data storage.

If you don't see this tab, select "Show Data Sources" from the Data menu.



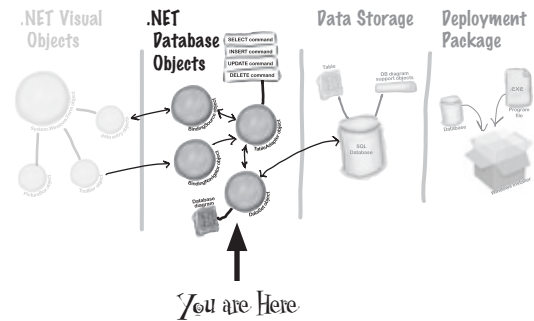
2 Select the People table.

Under the ContactDBDataSet, you should see the People table and all of the columns in it. Click the plus sign next to the People table to expand it—you'll see the columns that you added to your table. When you click on the People table in the Data Sources window and drag it onto your form, the IDE automatically adds data controls to your form that the user can use to browse and enter data. By default it adds a DataGridView, which lets the user work with the data using one big spreadsheet-like control. Click the arrow next to the People table and select Details—that tells the IDE to add individual controls to your form for each column in the table.



Click this arrow and choose Details to tell the IDE to add individual controls to your form rather than one large spreadsheet-like data control.

All of the columns you created should show up here.



3 Create controls that bind to the People table.

Drag and drop the People table onto your form. You should see controls appear for each column in your database. Don't worry too much about how they look right now; just make sure that they all appear on the form.

If you accidentally click out of the form you're working on, you can always get back to it by clicking the "Form1.cs [Design]" tab, or opening Form1.cs from the Solution Explorer.

The IDE creates this toolbar for navigating through the People table.

When you dragged the People table onto the form, a control was created for each column in the table.

These won't show up on your form, but represent the data set the IDE created to interact with the People table and ContactDB database.

This object connects the form to your People table.

This adapter allows your controls to interact with SQL commands that the IDE and data source generated for you.

The binding navigator connects the toolbar controls to your table.

Good programs are intuitive to use

Right now, the form works. But it doesn't look that great. Your application has to do more than be functional. It should be easy to use. With just a few simple steps, you can make the form look a lot more like the paper cards we were using at the beginning of the chapter.

Our form would be more intuitive if it looked a lot like the contact card.

Name: Laverne Smith
Company: XYZ Industries
Telephone: (212)555-8129
Email: Laverne.Smith@xyzindustries.com
Client: Yes **Last call:** 05/26/07

① Line up your fields and labels.

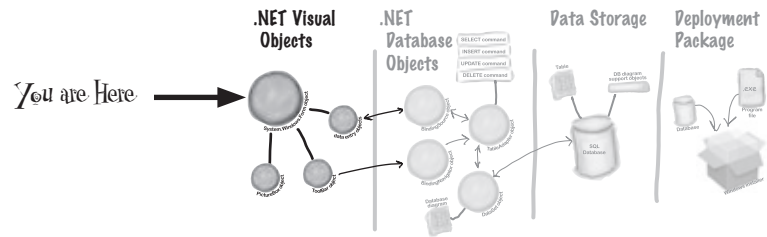
Line up your fields and labels along the left edge of the form. Your form will look like other applications, and make your users feel more comfortable using it.

Blue lines will show up on the form as you drag controls around. They're there to help you line the fields up.

② Change the Text Property on the Client checkbox.

When you first drag the fields onto the form your Client Checkbox will have a label to the right that needs to be deleted. Right below the Solution Explorer, you'll see the properties window. Scroll down to the Text property and delete the "checkbox1" label.

Delete this word to make the label go away.



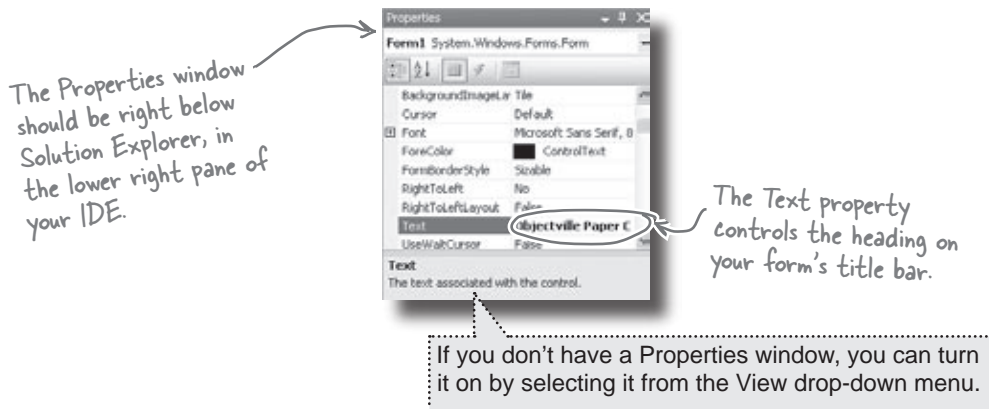
3

Make the application look professional.

You can change the name of the form by clicking on any space within the form, and finding the Text property in the Properties window of your IDE. Change the name of the form to “Objectville Paper Co. - Contact List.”

You can also turn off the Maximize and Minimize buttons in this same window, by looking for the MaximizeBox and MinimizeBox properties. Set these both to False.


The reason you want to turn off the Maximize button is that maximizing your form won't change the positions of the controls, so it'll look weird.



A good application not only works, but is easy to use. It's always a good idea to make sure it behaves as a typical user would expect it to.

okay, one last thing...

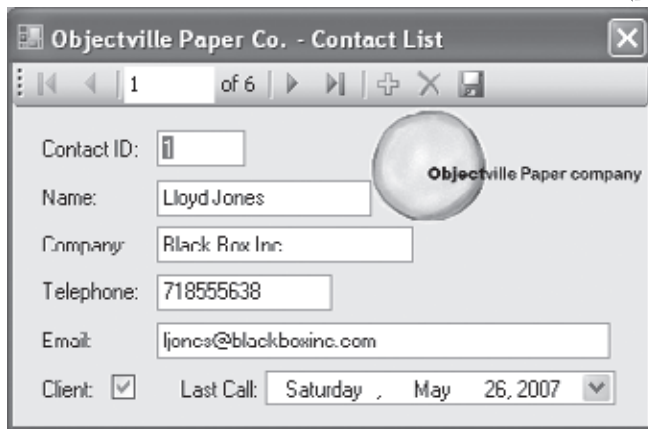
Test drive

Okay, just one more thing to do... run your program and make sure it works the way you think it should! Do it the same way you did before—press the F5 key on your keyboard, or click the green arrow button  on the toolbar (or choose “Run” from the Debug menu).

You can always run your programs at any time, even when they’re not done—although if there’s an error in the code, the IDE will tell you and stop you from executing it.

Click the X box in the corner to stop the program so you can move on to the next step.

These controls let you page through the different records in the database.



Building your program overwrites the data in your database.

We'll spend more time on this in the next chapter.

The IDE builds first, then runs.

When you run your program in the IDE it actually does two things. First it **builds** your program, then it **executes** it. This involves a few distinct parts. It **compiles** the code, or turns it into an executable file. Then it places the compiled code, along with any resources and other files, into a subdirectory underneath the bin folder.

In this case, you'll find the executable and SQL database file in bin/debug. Since it copies the database out each time, any changes you make will be lost the next time you run inside the IDE. But if you run the executable from Windows, it'll save your data—until you build again, at which point the IDE will overwrite the SQL database with a new copy that contains the data you set up from inside the Database Explorer.



Every time you build your program, the IDE puts a fresh copy of the database in the bin

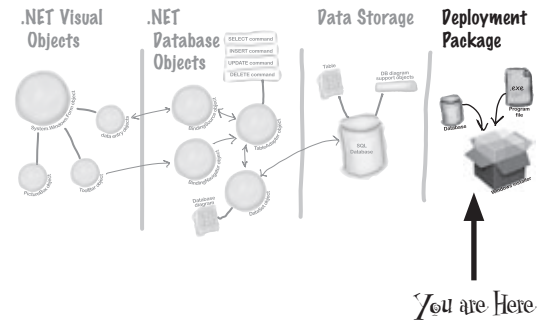
folder. This will overwrite any data you added when you ran the program.

When you debug your program, the IDE rebuilds it if the code has changed—which means that your database will sometimes get overwritten when you run your program in the IDE. If you run the program directly from the bin/debug or bin/release folder, or if you use the installer to install it on your machine, then you won't see this problem.

How to turn YOUR application into EVERYONE'S application

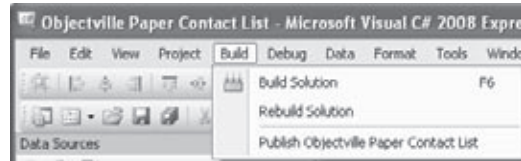
At this point, you've got a great program. But it only runs on your machine. That means that nobody else can use the app, pay you for it, see how great you are and hire you... and your boss and customers can't see the reports you're generating from the database.

C# makes it easy to take an application you've created, and **deploy** it. Deployment is taking an application and installing it onto other machines. And with the Visual C# IDE, you can set up a deployment with just two steps.



1

Select *Publish Contacts* from the Build menu.



Building the solution just copies the files to your local machine. Publish creates a Setup executable and a configuration file so that any machine could install your program.

2

Just accept all of the defaults in the Publish Wizard by clicking Finish. You'll see it package up your application and then show you a folder that has your Setup.exe in it.

