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.

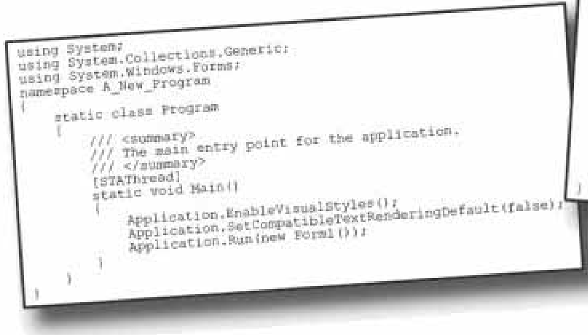
Why you should learn C#

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.

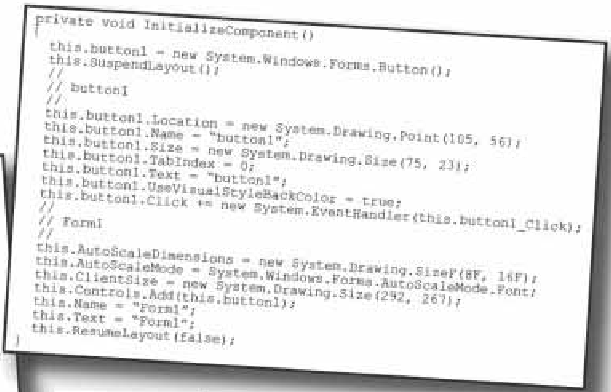
The IDE—or Visual Studio Integrated Development Environment—is an important part of working in C#. It's a program that helps you edit your code, manage your files, and publish your projects.

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());
        }
    }
}
```

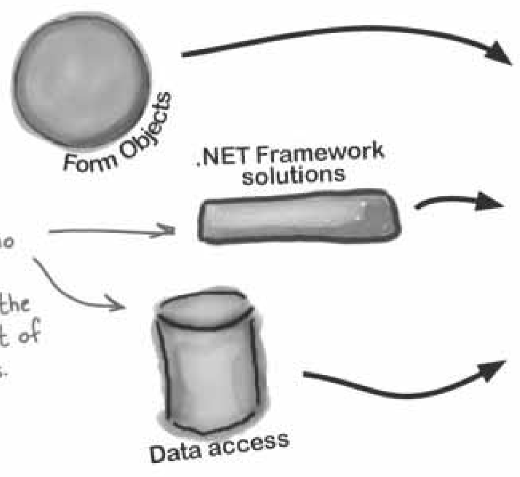


```
private void InitializeComponent()
{
  this.button1 = new System.Windows.Forms.Button();
  this.SuspendLayout();
  //
  // button1
  //
  this.button1.Location = new System.Drawing.Point(105, 56);
  this.button1.Name = "button1";
  this.button1.Size = new System.Drawing.Size(75, 23);
  this.button1.TabIndex = 0;
  this.button1.Text = "button1";
  this.button1.UseVisualStyleBackColor = true;
  this.button1.Click += new System.EventHandler(this.button1_Click);
  //
  // Form1
  //
  this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
  this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
  this.ClientSize = new System.Drawing.Size(292, 267);
  this.Controls.Add(this.button1);
  this.Name = "Form1";
  this.Text = "Form1";
  this.ResumeLayout(false);
}
```

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.

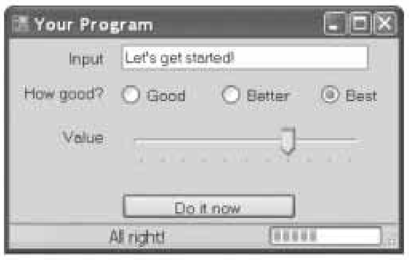
What you get with Visual Studio and C#...

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:



C# and the Visual Studio IDE have pre-built structures that handle the tedious code that's part of most programming tasks.

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



Your Program

Input: Let's get started!

How good? ( ) Good ( ) Better [✓ Best]

Value

Do it now

All right!

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:

1. **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.

2. **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.

3. **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.

4. **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

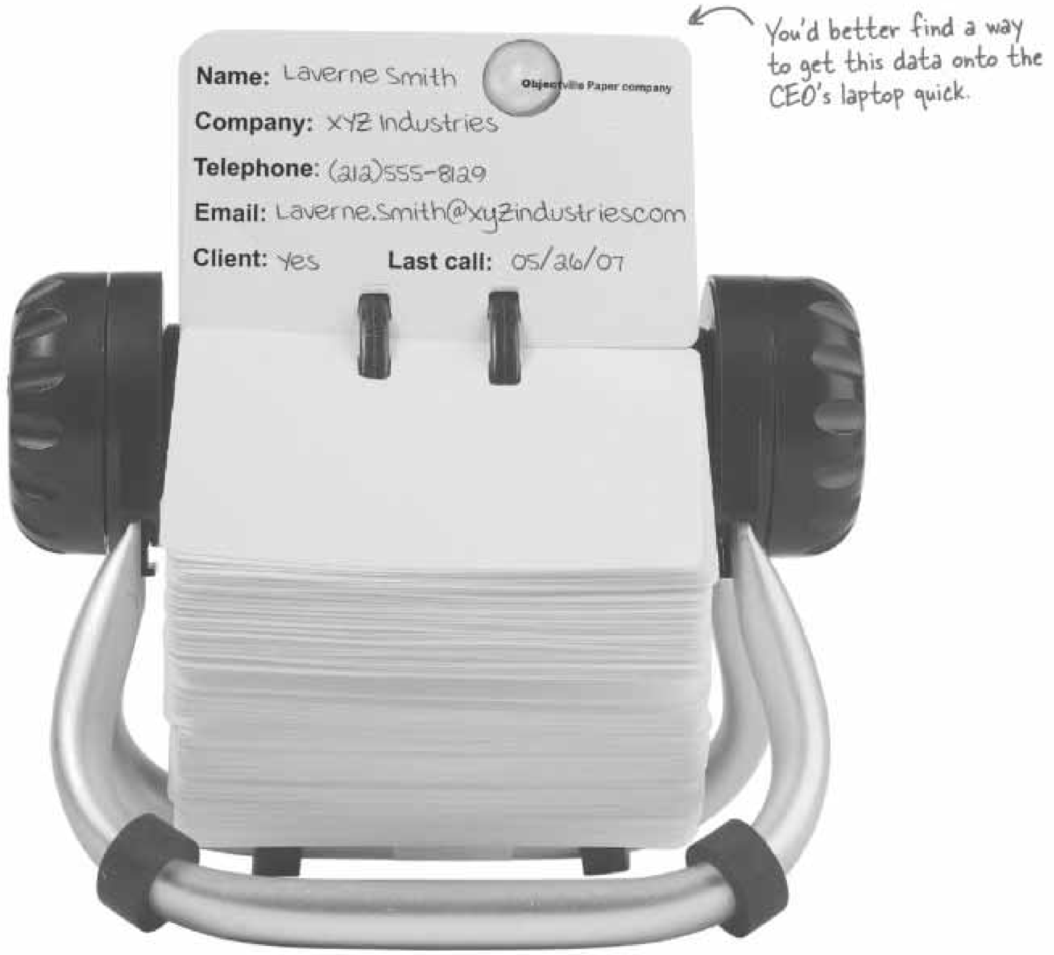
   ★ Managing and interacting with databases

   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.


Name: Laverne Smith

Company: XYZ Industries

Telephone: (212)555-8129

Email: Laverne.Smith@xyzindustriescom

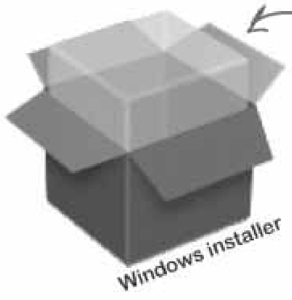
Client: yes Last call: 05/26/07

Objectville Paper company
You'd better find a way to get this data onto the CEO's laptop quick.

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.



The CEO wants to be able to run his program on his desktop and laptop, so an installer is a must.

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.

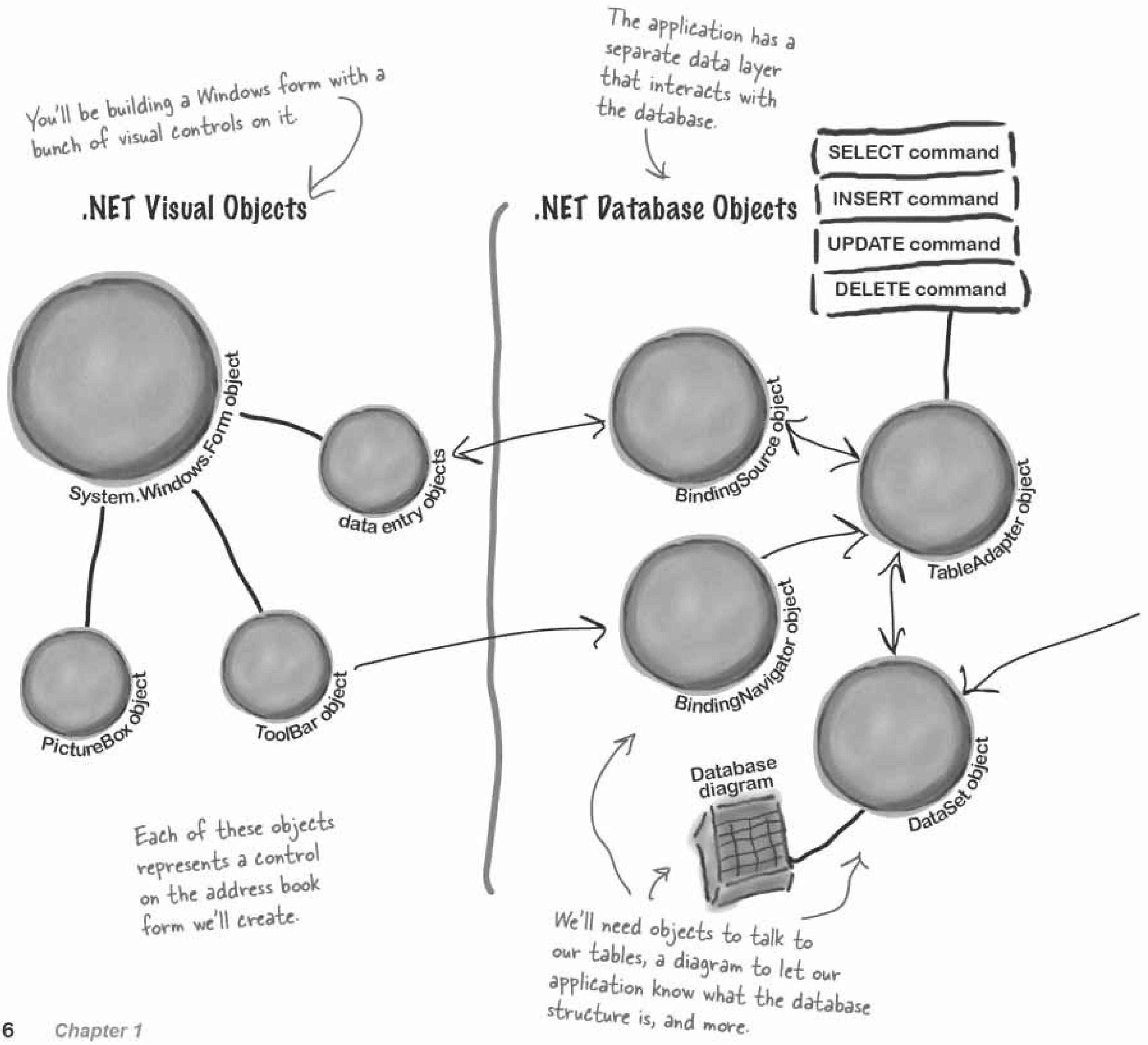
   The CEO figures a database would be the best way that everyone in the company to see his data, and then he can just keep up with one copy of all his contacts.

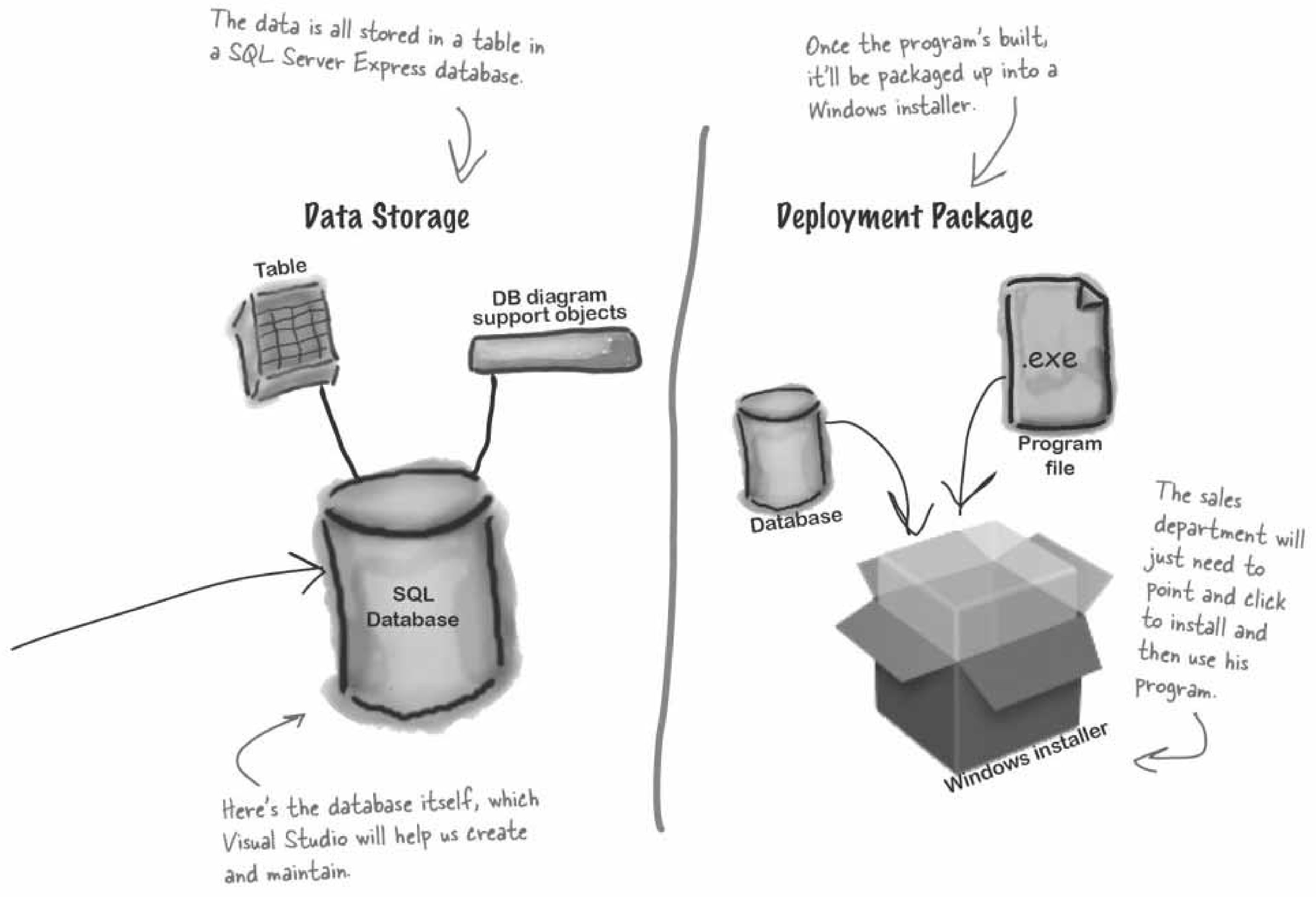
We already know that Visual C# makes working with databases easy. Having contacts in a database lets the CEO and the sales team all access the information, even though there's only one copy of the data.

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.

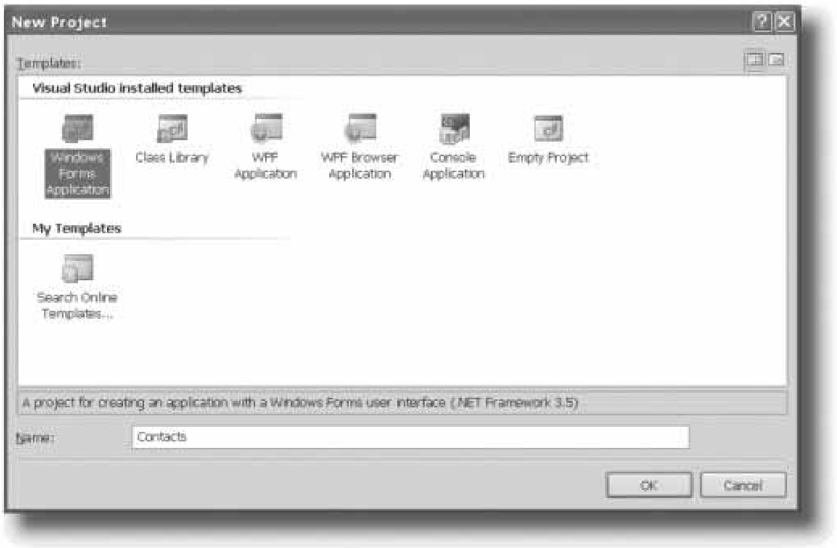
Here's the structure of the program we're going to create:


You'll be building a Windows form with a bunch of visual controls on it.
The application has a separate data layer that interacts with the database.
.NET Visual Objects
.NET Database Objects
SELECT command
INSERT command
UPDATE command
DELETE command
System.Windows.Form object
data entry objects
PictureBox object
ToolBar object
BindingSource object
BindingNavigator object
TableAdapter object
DataSet object
Database diagram
Each of these objects represents a control on the address book form we'll create.
We'll need objects to talk to our tables, a diagram to let our application know what the database structure is, and more.


The data is all stored in a table in a SQL Server Express database.
Data Storage
Table
DB diagram support objects
SQL Database
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
.exe
Program file
Database
The sales department will just need to point and click to install and then use his program.
Windows installer

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.


New Project
Templates:
Visual Studio installed templates
Windows Forms Application
Class Library
WPF Application
WPF Browser Application
Console Application
Empty Project
My Templates
Search Online Templates...
A project for creating an application with a Windows Forms user interface (.NET Framework 3.5)
Name:
Contacts
OK
Cancel

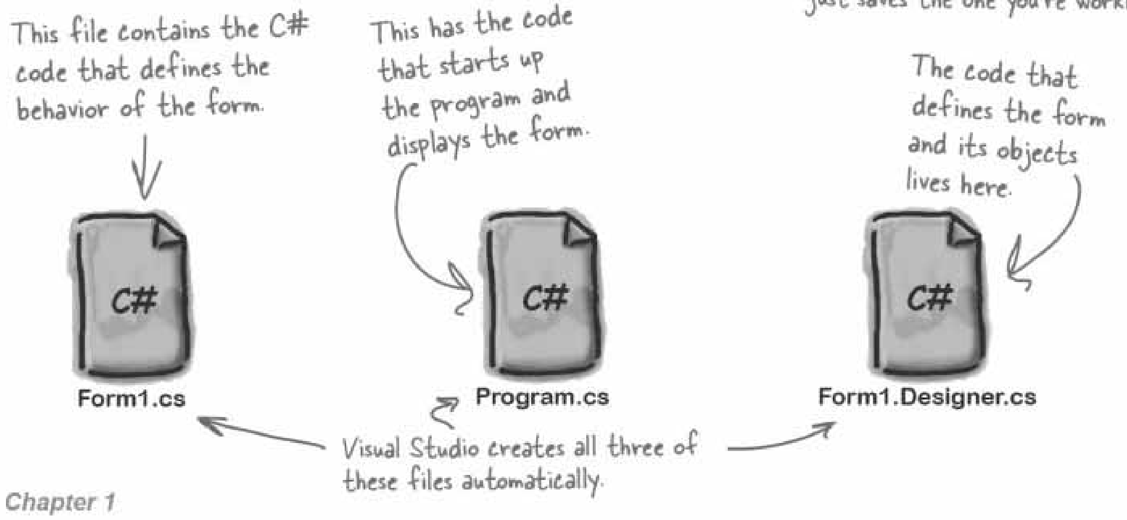
Things may look a bit different in your IDE.

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.
This has the code that starts up the program and displays the form.
The code that defines the form and its objects lives here.
C#
Form1.cs
C#
Program.cs
C#
Form1.Designer.cs
Visual Studio creates all three of these files automatically.

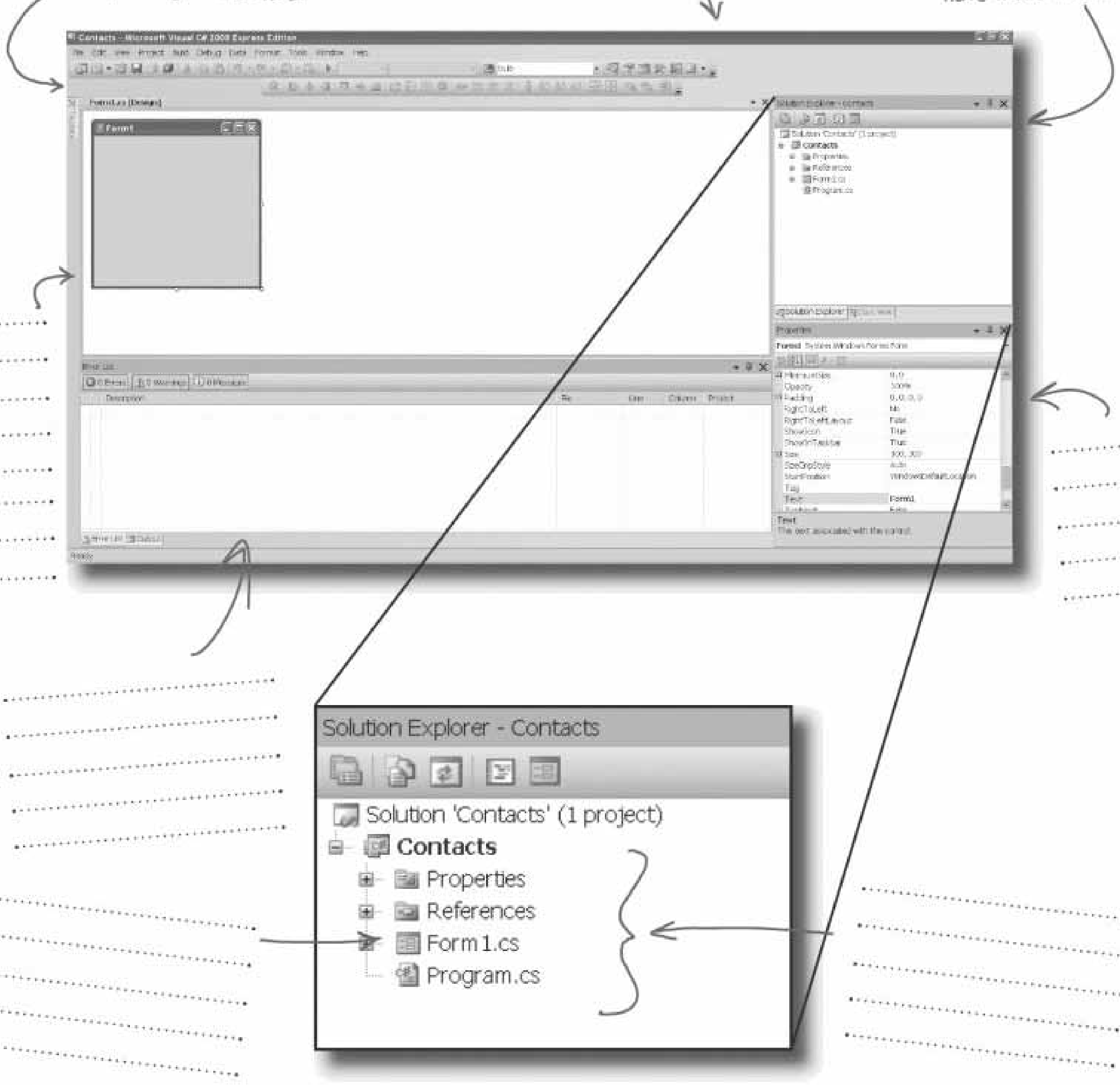
Sharpen your pencil

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 picure, you can select "Reset Window Layout" from the Window menu.

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

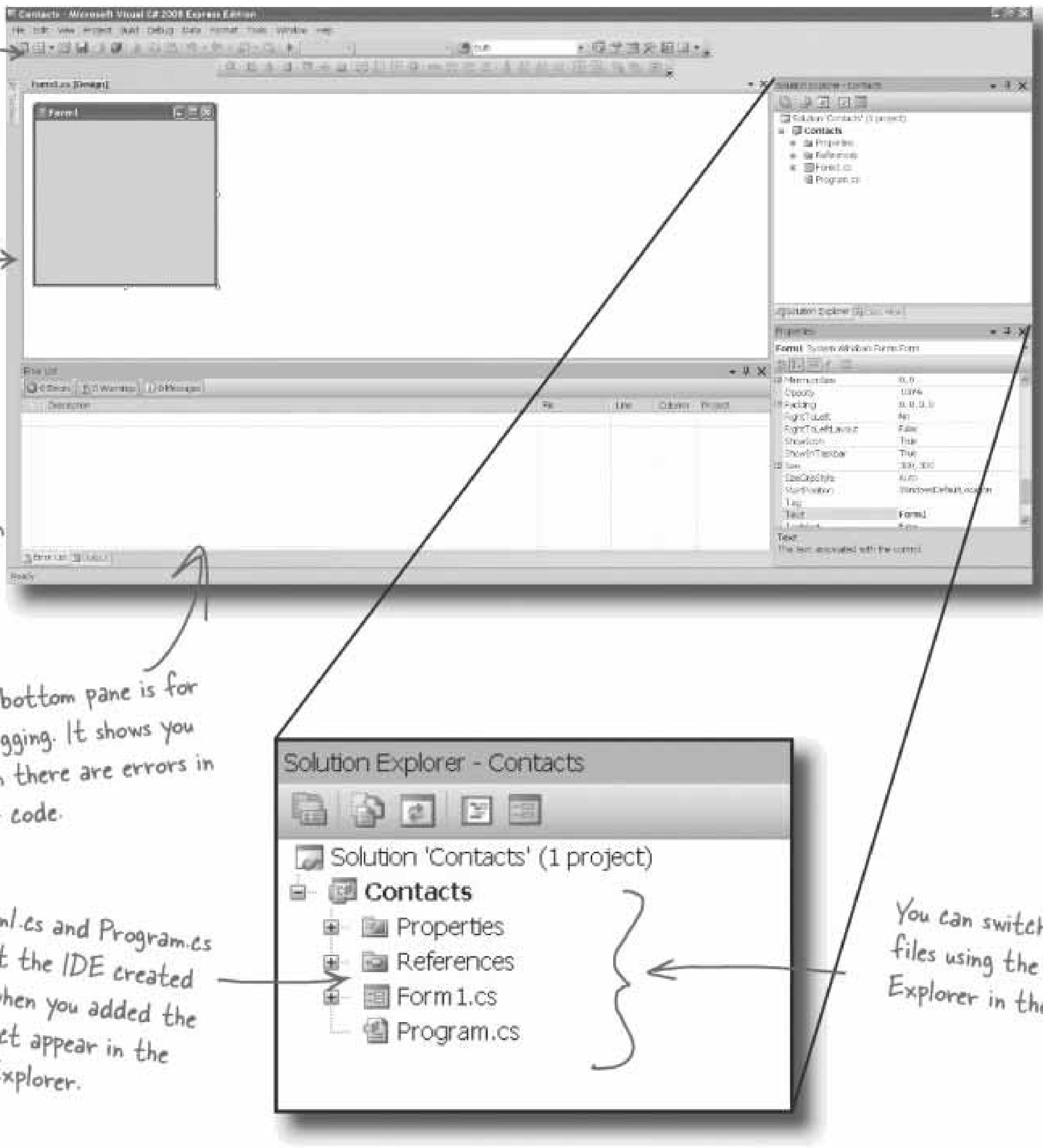

Contacts - Microsoft Visual C# 2008 Express Edition
Form1.cs [Design]
Form1
Error List
Solution Explorer - Contacts
Properties
Solution Explorer - Contacts
Solution 'Contacts' (1 project)
Contacts
Properties
References
Form1.cs
Program.cs

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.

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



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

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

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 **action code**, or the code that 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 files the IDE generates for me?

A: Absolutely, you can change any aspect of your program. But the IDE is set up to name your files sensibly. When you add a file to your project, the filename you choose affects the way the code is generated, and the code it generates will include that name. In some cases, if you rename the file you'll either have to go through and change other parts of the code, or live with the difference between the filename and the code inside it. Since that's a bit of a pain, we recommend you don't change filenames unless you've got a really good reason to.

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.

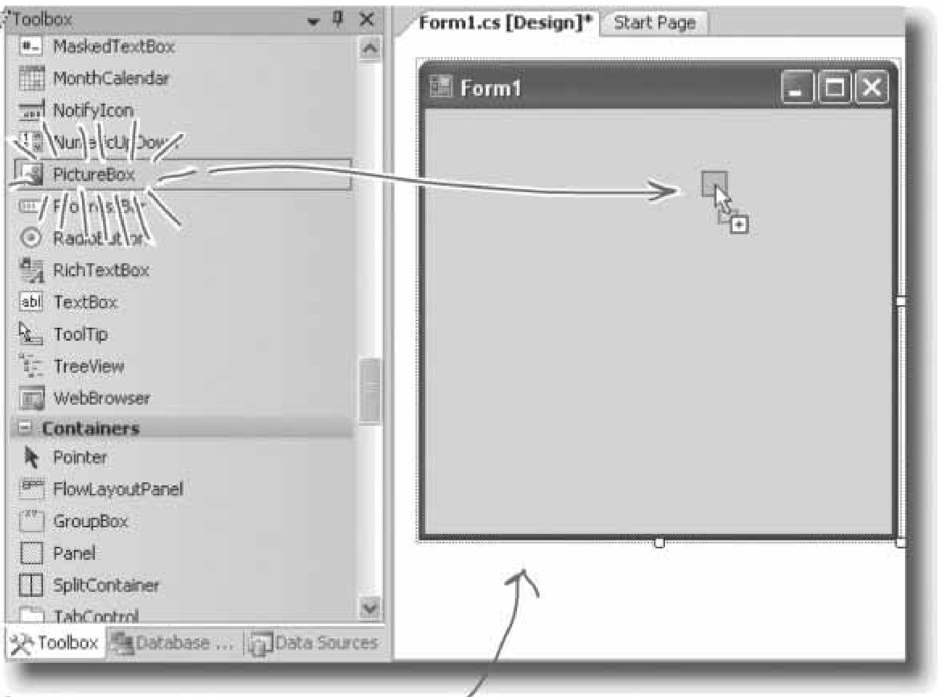
Making sure the application does what it's supposed to do is still up to you.

Develop the user interface

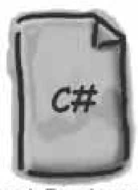
Adding controls and polishing the user interface is as easy as dragging and dropping with the Visual Studio IDE. Let's add a logo to the form:

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.



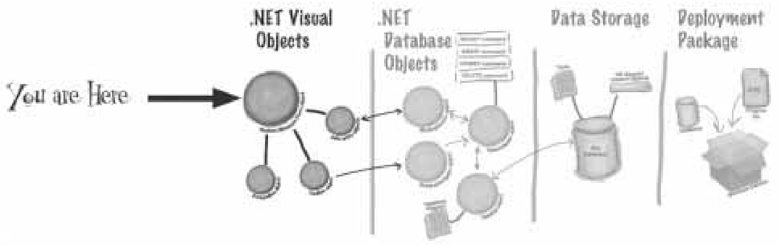
Every time you make a change to a control's properties on the form, the code in Form1.Designer.cs is getting changed by the IDE.



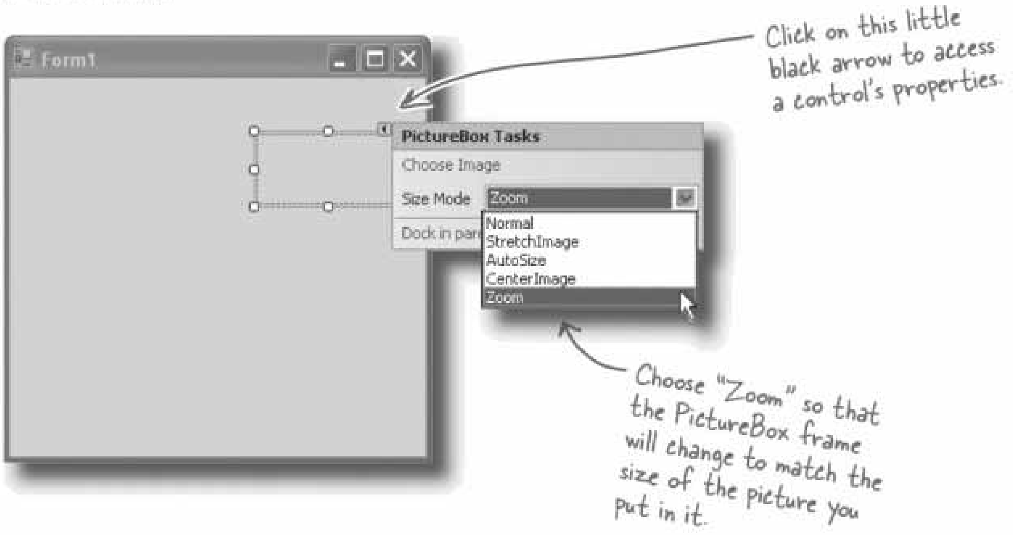
Form1.Designer.cs

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

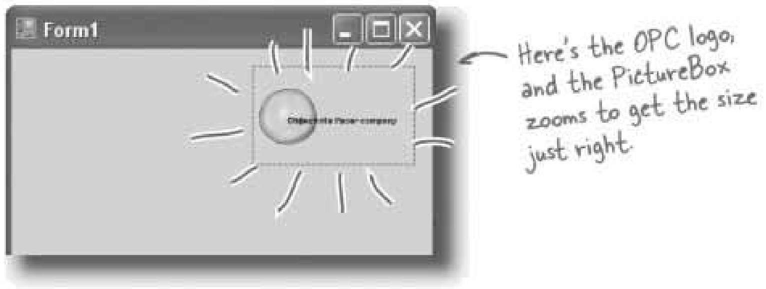
We'll talk a lot more about designing good user interfaces later on. For now, just get the logo and other controls on your form, and worry about ***behavior***. We'll add some style later.



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:


Form1
PictureBox Tasks
Choose Image
Size Mode
Zoom
Dock in par
Normal
StretchImage
AutoSize
CenterImage
Zoom
Click on this little black arrow to access a control's properties.
Choose "Zoom" so that the PictureBox frame will change to match the size of the picture you put in it.

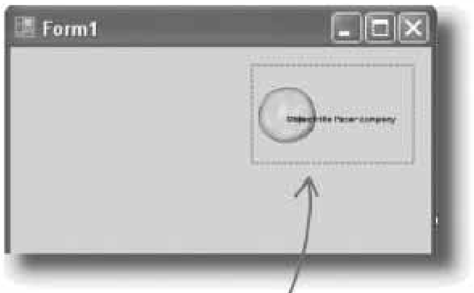
3 **Download the Objectville Paper Company logo.**
Download the Objectville Paper Co. logo from Head First Labs (`http://www.headfirstlabs.com/books/hfcsharp`) and save it to your hard drive. Then click the PictureBox properties arrow, and select Choose Image. Click Import, find your logo, and you're all set:


Form1
Objectville Paper company
Here's the OPC logo, and the PictureBox zooms to get the size just right.

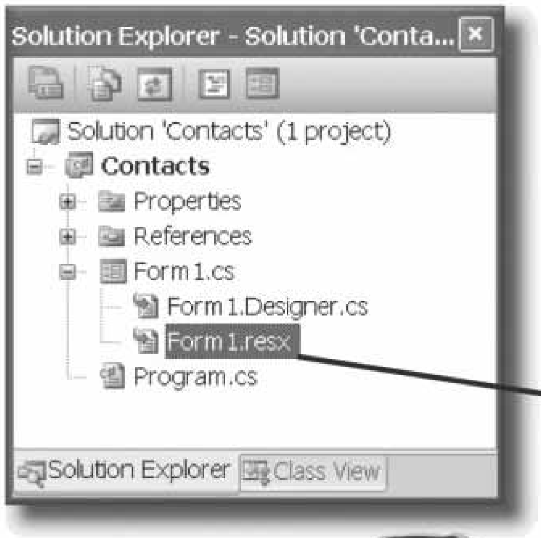
Visual Studio, behind the scenes

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.

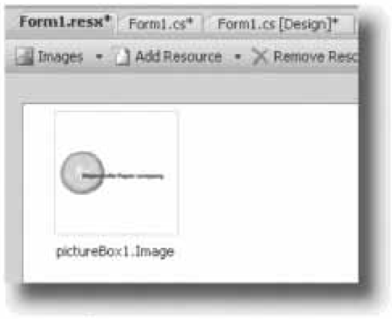
When you dragged the PictureBox control onto your form, the IDE automatically created a resource file called Form1.resx to store that resource and keep it in the project. Double-click on this file, and you'll be able to see the newly imported image.

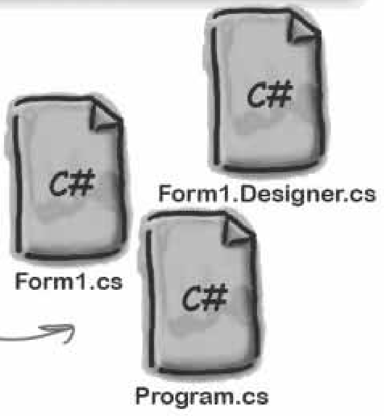


This image is now a resource of the Contact List application.



If you click on Form1.resx in the Solution Explorer, you can see the logo that you imported. That file is what links it to the PictureBox, and the IDE added code to do the linking.





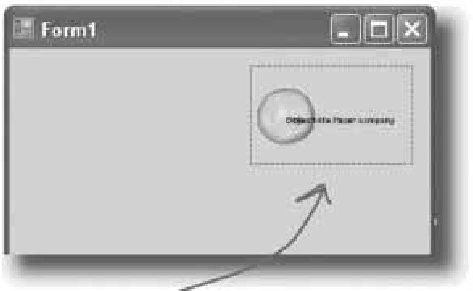
Here are the files Visual Studio created earlier.

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 double-click on it.

Make sure you've got your form showing in the IDE, and double-click on the PictureBox control. 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-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.

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?

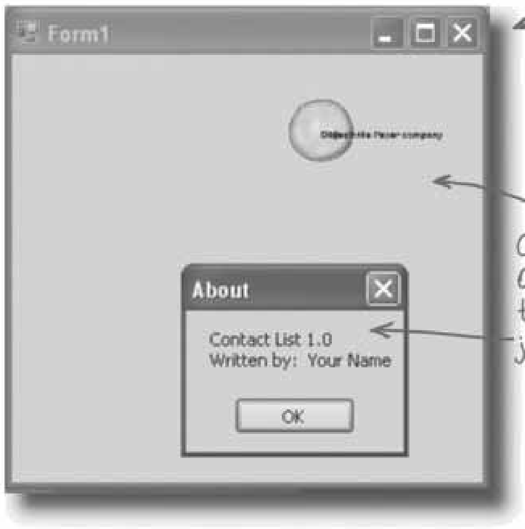
A: A **method** is just a *named block of code*. We'll talk a lot more about methods in Chapter 2.

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:".

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: ■.



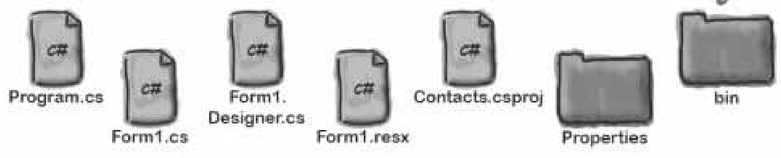
All three of these buttons work—and you didn't have to write any code to make them work.

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.

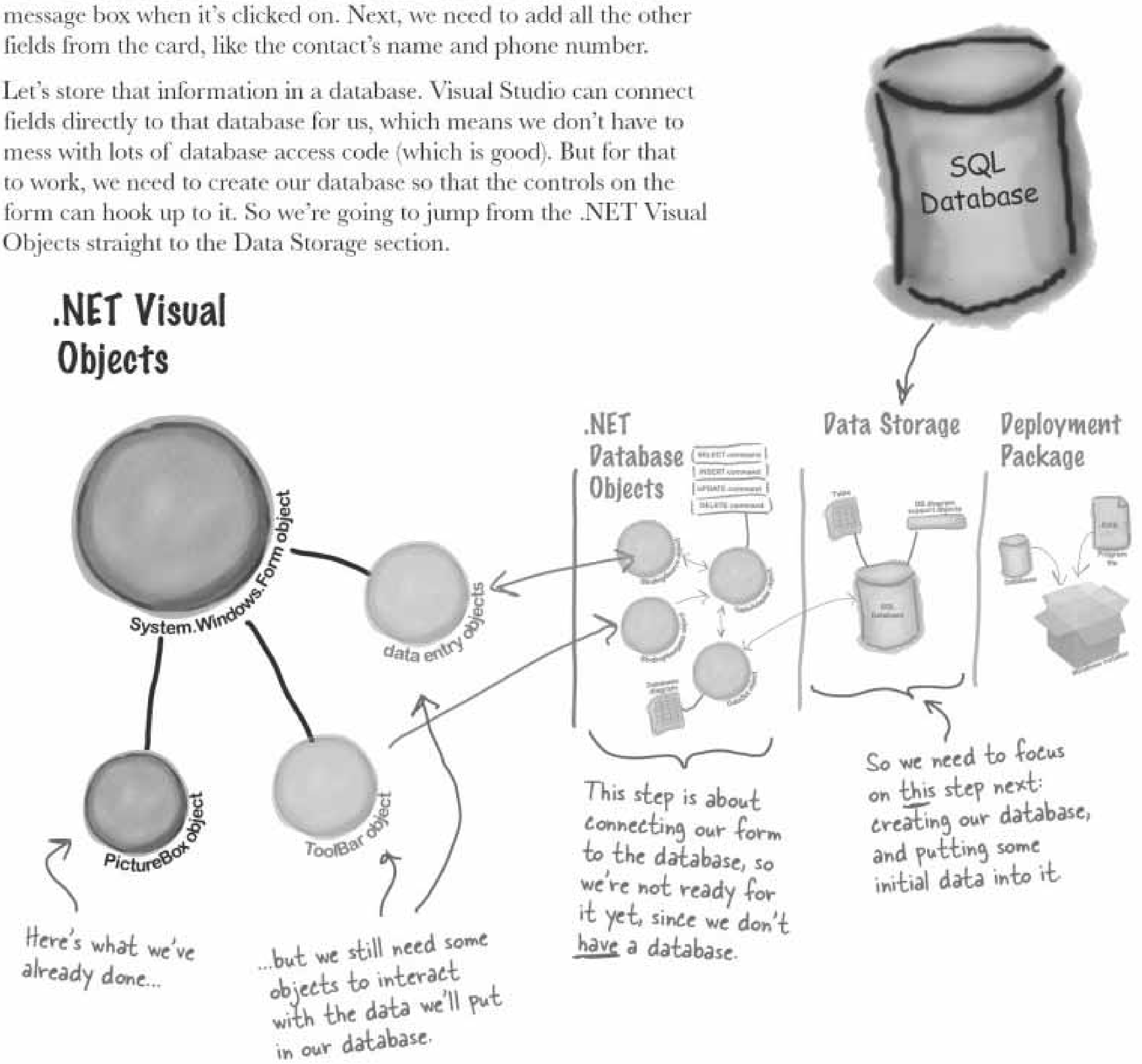
Q: I don't see the Stop Debugging button on my toolbar. What gives?

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.


SQL Database
.NET Visual Objects
.NET Database Objects
Data Storage
Deployment Package
System.Windows.Form object
data entry objects
PictureBox object
ToolBar object
Here's what we've already done...
...but we still need some objects to interact with the data we'll put in our database.
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.
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.

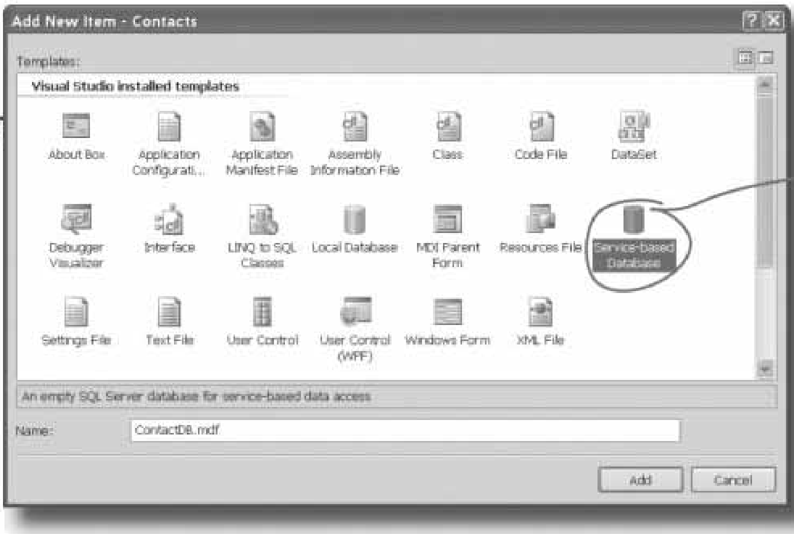
Make sure you've stopped debugging before you continue.

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.



An empty SQL Server database for service-based data access

Name: ContactDB.mdf

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. **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.

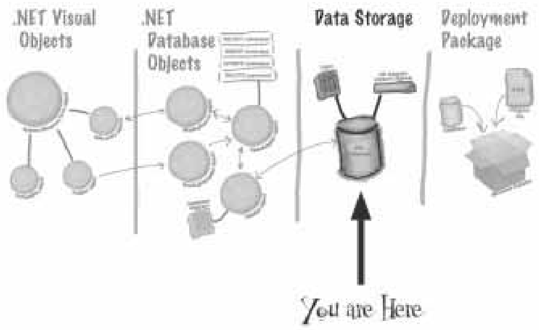
3. **View your database in the Solution Explorer.**
Go to the Solution Explorer, and you'll see that ContactDB has been added to the file list. Double click ContactDB.mdf in the Solution Explorer and look at the left side of your screen. The Toolbox has changed to a Database Explorer.

Watch it!

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

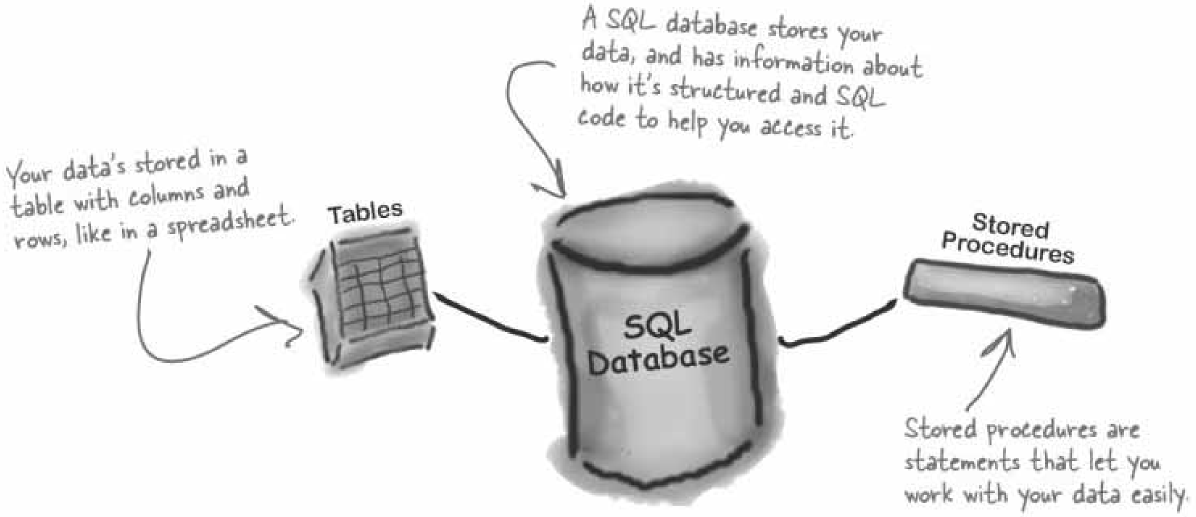
The Visual Studio 2008 Professional and Team Foundation editions don't have a Database Explorer window. Instead, they have a Server Explorer window, which does everything the Database Explorer does, but also lets you explore data on your network.

The IDE created a database



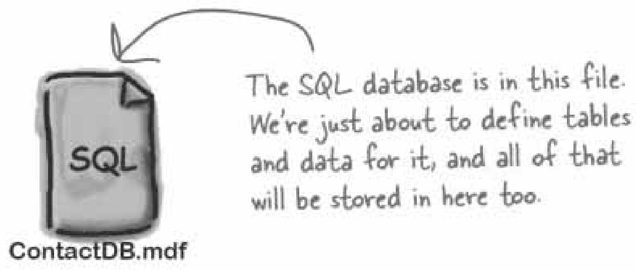
When you told the IDE to add a new SQL database to your project, the IDE created a new database for you. A **SQL database** is a system that stores data for you in an organized, interrelated way. The IDE gives you all the tools you need to maintain your data and databases.

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.


A SQL database stores your data, and has information about how it's structured and SQL code to help you access it.
Your data's stored in a table with columns and rows, like in a spreadsheet.
Tables
SQL Database
Stored Procedures
Stored procedures are statements that let you work with your data easily.

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 you 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.

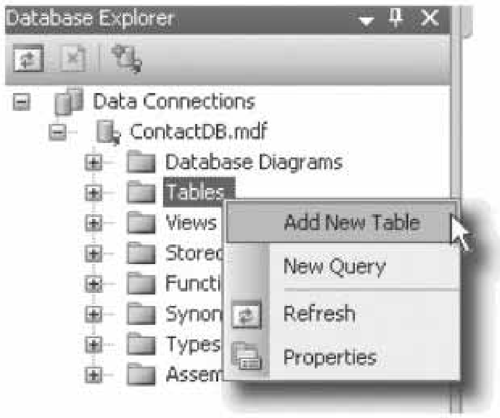

SQL
ContactDB.mdf
The SQL database is in this file. We're just about to define tables and data for it, and all of that will be stored in here too.

[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:

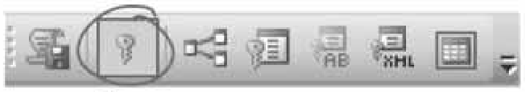
1. **Add a table to the ContactDB database.**
Right click on Tables in the Database Explorer, and select Add New Table. This will open up a window where you can define the columns in the table you just created.



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?

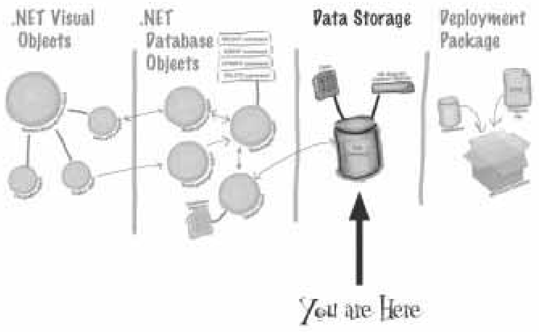
A: It helps to have a unique ID for each record in most database tables. Since we're storing contact information for individual people, we decided to create a column for that, and call it ContactID.

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?

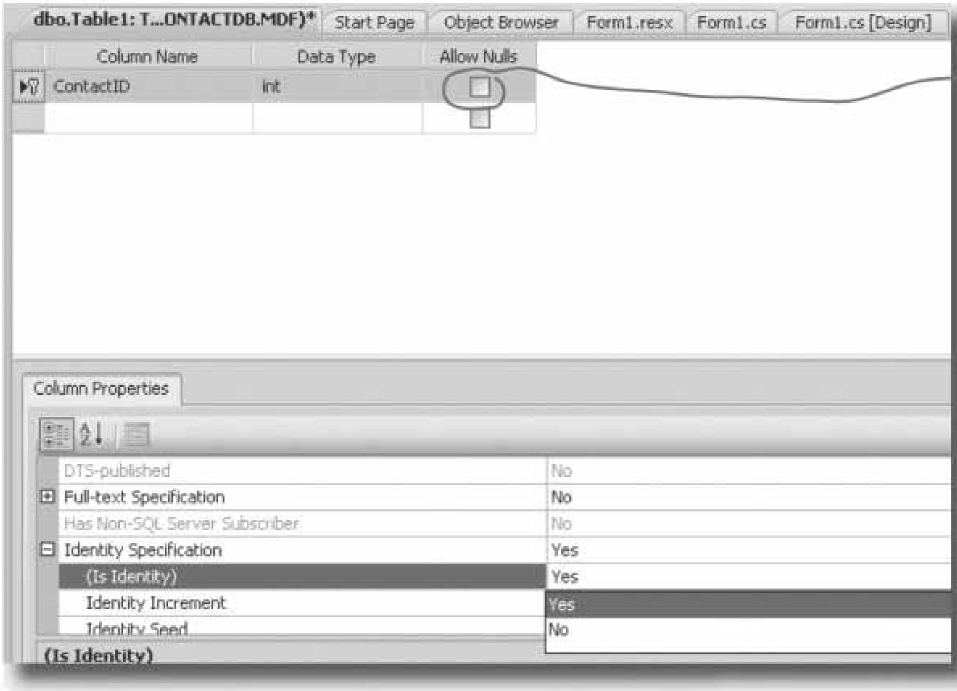
A: No, it's OK if you don't understand everything right now. Focus on the basic steps, and we'll spend a lot more time on databases in the later chapters of the book. And if you're dying to know more right away, you can always pick up ***Head First SQL*** to read along with this book.



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.



Column Name	Data Type	Allow Nulls
ContactID	int	☐
		☐

DTS-published	No
Full-text Specification	No
Has Non-SQL Server Subscriber	No
Identity Specification	Yes
(Is Identity)	Yes
Identity Increment	Yes
Identity Seed	No

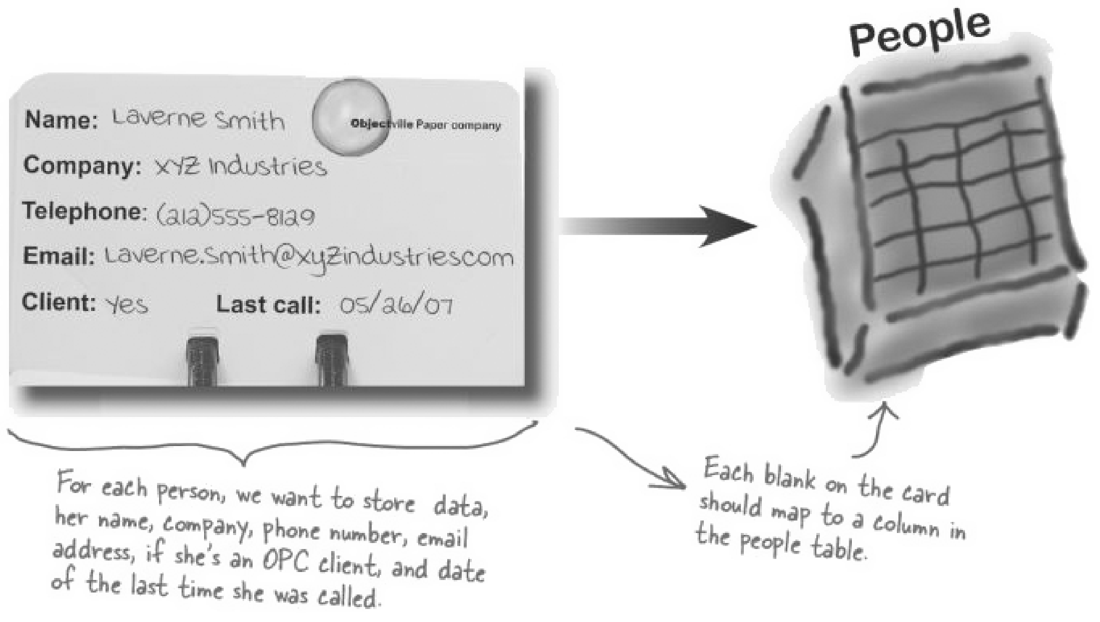
(Is Identity)

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.

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 your going to track in the database. Each field on our written contact card should become a column in the People table.



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

For each person, we want to store data, her name, company, phone number, email address, if she's an OPC client, and date of the last time she was called.

Each blank on the card should map to a column in the people table.

BRAIN POWER

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

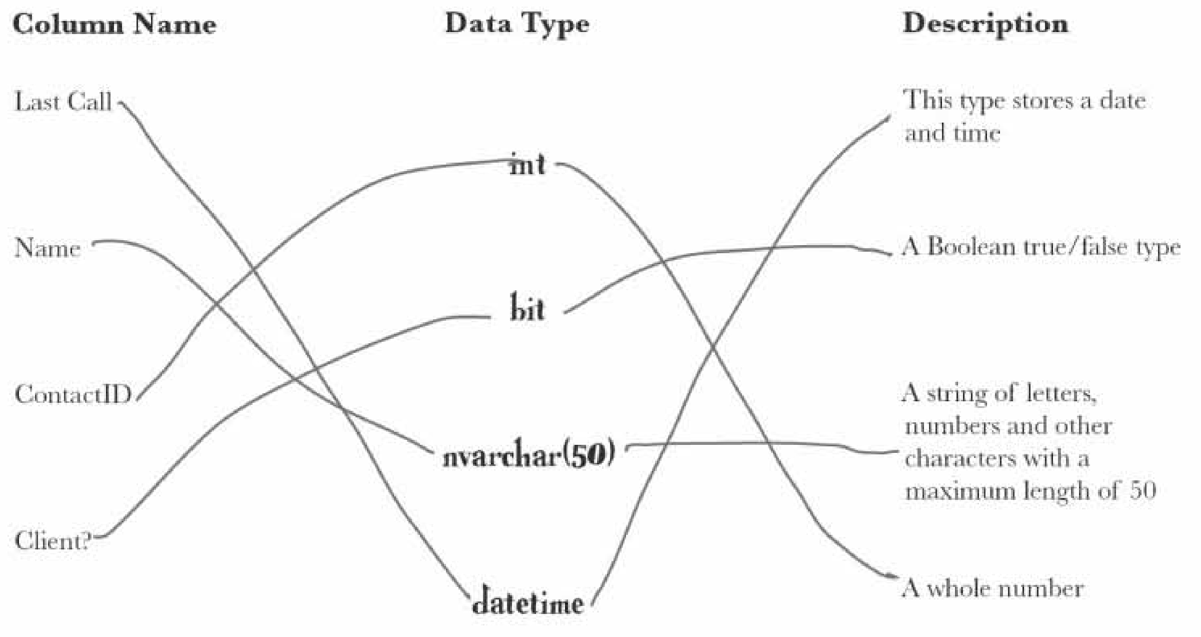
WHO DOES WHAT?

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

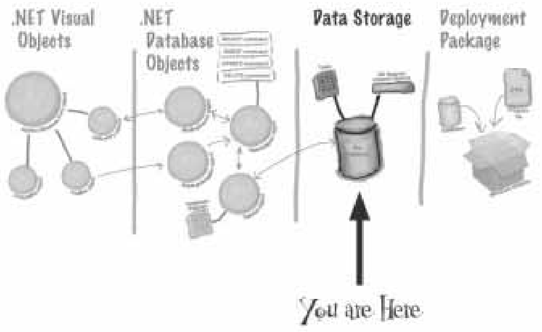
WHO DOES WHAT?

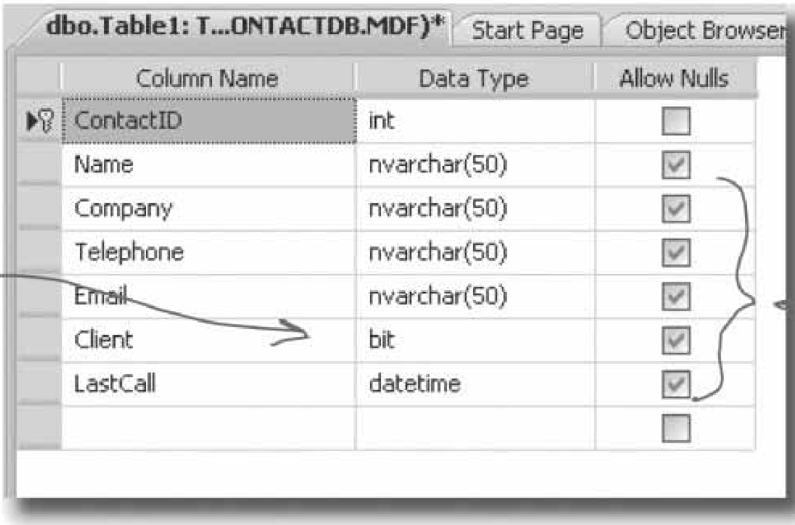
Now that you've created a People table and a primary key column, you need to add columns for all of the data fields. See if you can work out which data type goes with each of the columns in your table, and also match the data type to the right description.


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





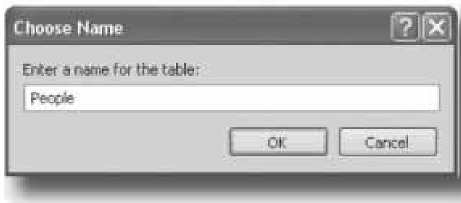
Column Name	Data Type	Allow Nulls
ContactID	int	☐
Name	nvarchar(50)	✓
Company	nvarchar(50)	✓
Telephone	nvarchar(50)	✓
Email	nvarchar(50)	✓
Client	bit	✓
LastCall	datetime	✓
		☐

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

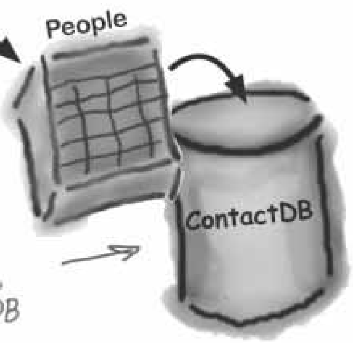
If you uncheck Allow Nulls, the column must have a value.

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

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.



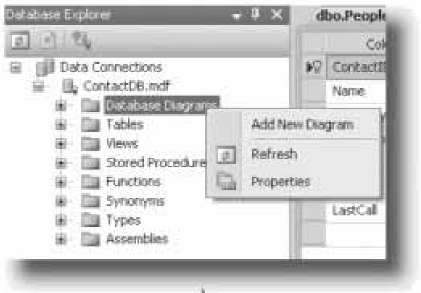
We've been talking about this table as the "People" table, but it's not until this step that you give it an official name.



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

Diagram your data so your application can access it

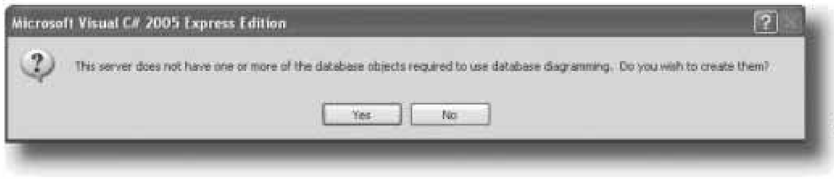
Once you've created your database and tables, you have to let your application know about it. That's where a database diagram comes in. A **database diagram** is a simple description of your table that the Visual Studio IDE can use to work with the table. It also lets the IDE automatically generate SQL statements to add, change, and delete rows in the table.



1. **Create a new database diagram.**
   Go to the Database Explorer window and right-click on the Database Diagrams node. Select Add New Diagram.

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

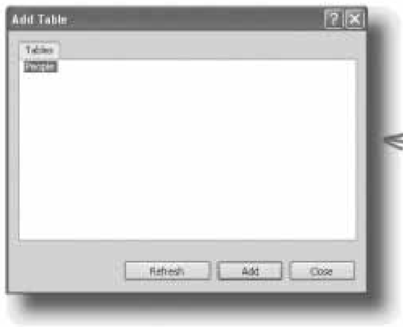
2. **Let the IDE generate access code.**
   Before you tell the IDE about your specific table, it needs to create some basic stored procedures for interacting with your database. Just click Yes here, and let the IDE go to work.



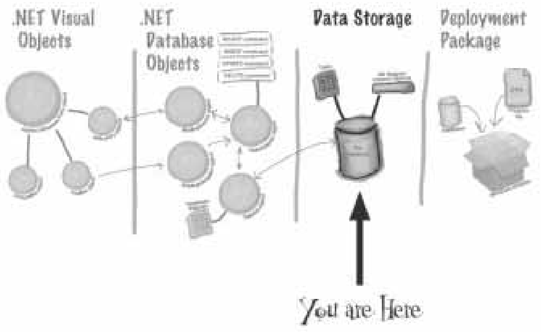
This server does not have one or more of the database objects required to use database diagramming. Do you wish to create them?

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

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.



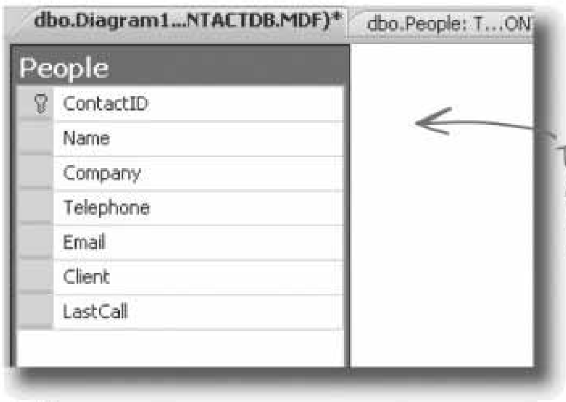
When you have databases with multiple tables, each table will show up as an entry on this window.



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.



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.

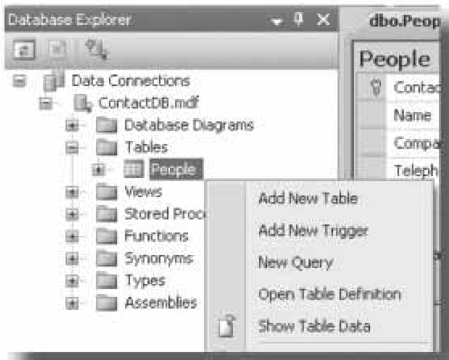
If you had any other tables in the database you wanted diagrammed, they would appear here, too.

A database diagram describes your tables to the Visual Studio IDE. The IDE then uses the database diagram to auto-generate code to work with your database.

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.

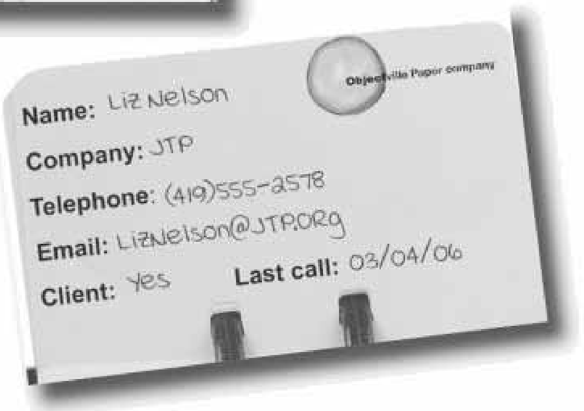
1. Expand Tables and then right click on the People Table in the Database Explorer (or Server Explorer) and select Show Table Data.



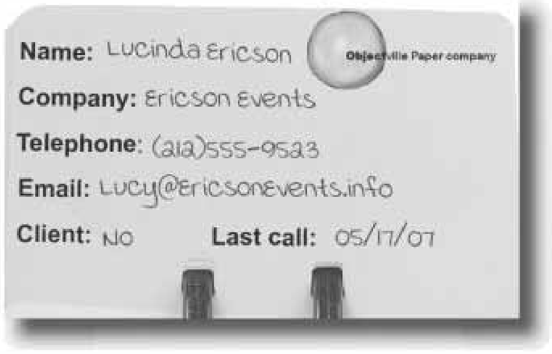
Your job is to enter the data from all six of these cards into the People table.

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.

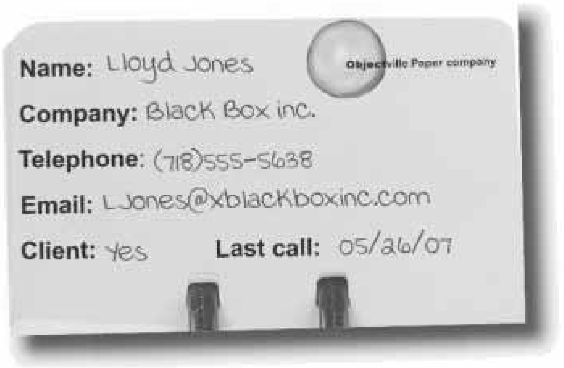
Type "True" or "False" in the Client column. That's how SQL stores yes or no info.



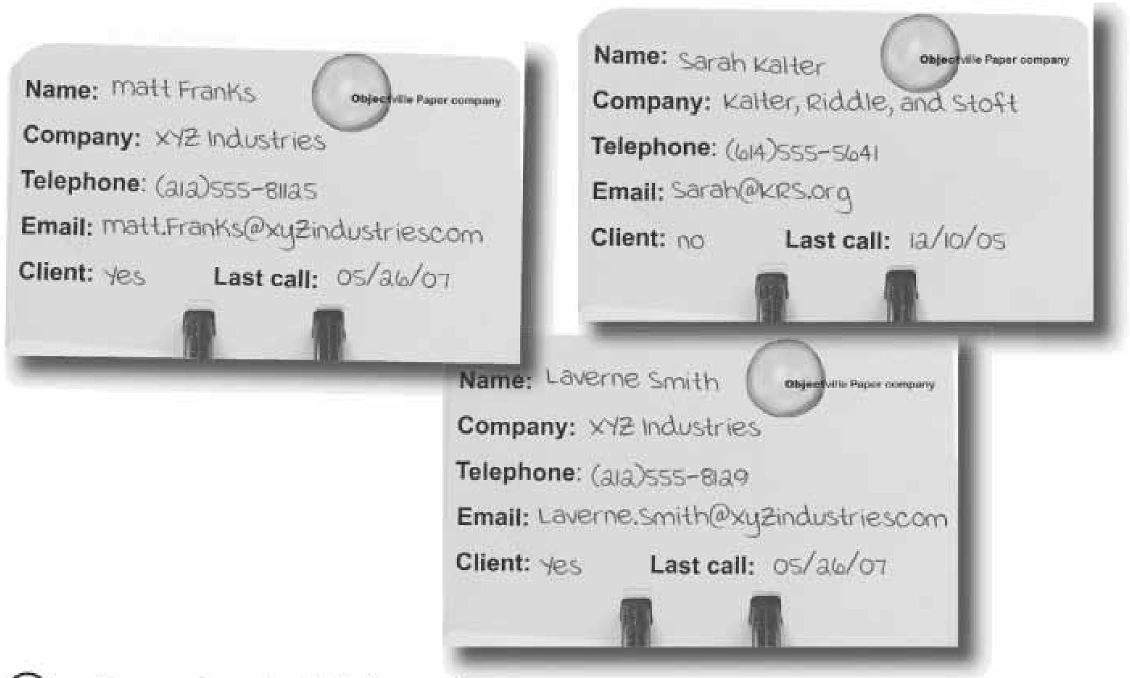
Name: Liz Nelson
Company: JTP
Telephone: (419)555-2578
Email: LizNelson@JTP.ORG
Client: Yes **Last call:** 03/04/06



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



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



Name: Matt Franks
Company: XYZ Industries
Telephone: (212)555-8125
Email: matt.Franks@xyzindustriescom
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: (212)555-8129
Email: Laverne.Smith@xyzindustriescom
Client: yes **Last call:** 05/26/07

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.

SQL

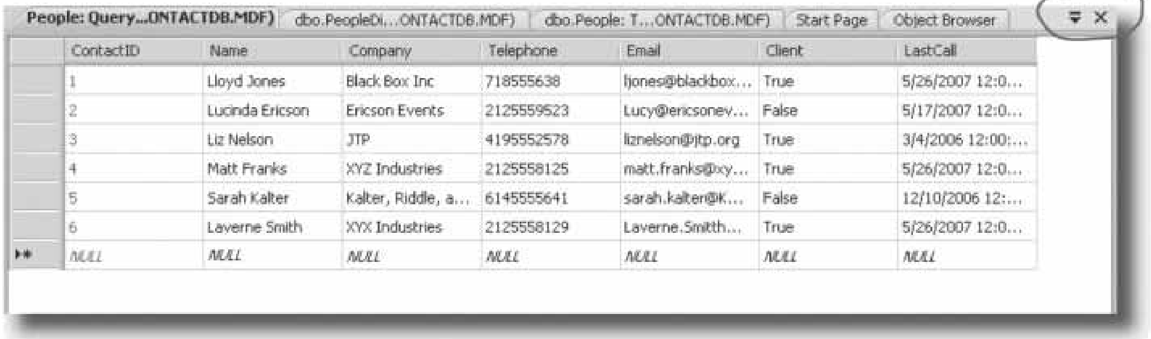
ContactDB.mdf

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.

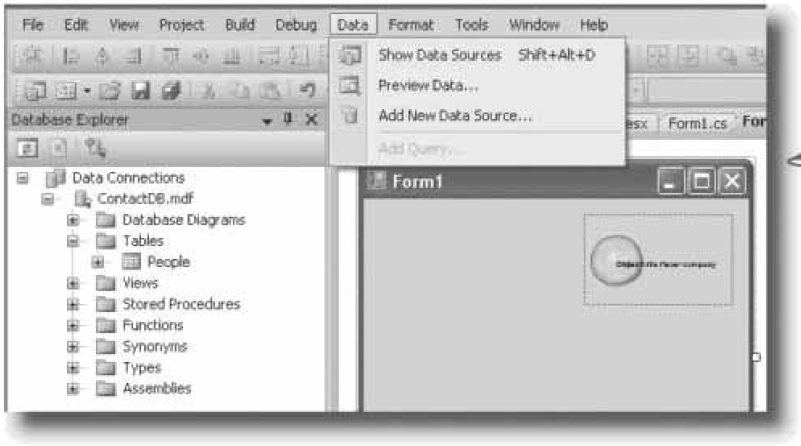
You need to close both the data grid and the diagram to get back to your form.

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.

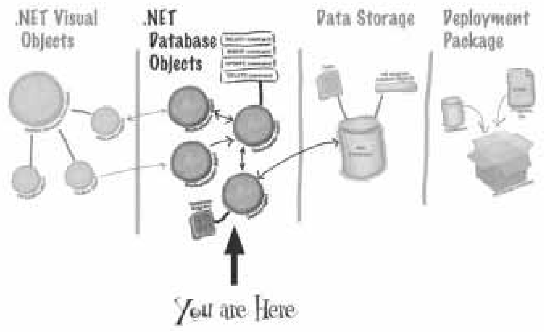

People: Query...ONTACTDB.MDF)
dbo.PeopleDi...ONTACTDB.MDF)
dbo.People: T...ONTACTDB.MDF)
Start Page
Object Browser

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@ericsonev...	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@xy...	True	5/26/2007 12:0...
5	Sarah Kalter	Kalter, Riddle, a...	6145555641	sarah.kalter@K...	False	12/10/2006 12:...
6	Laverne Smith	XYX 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.

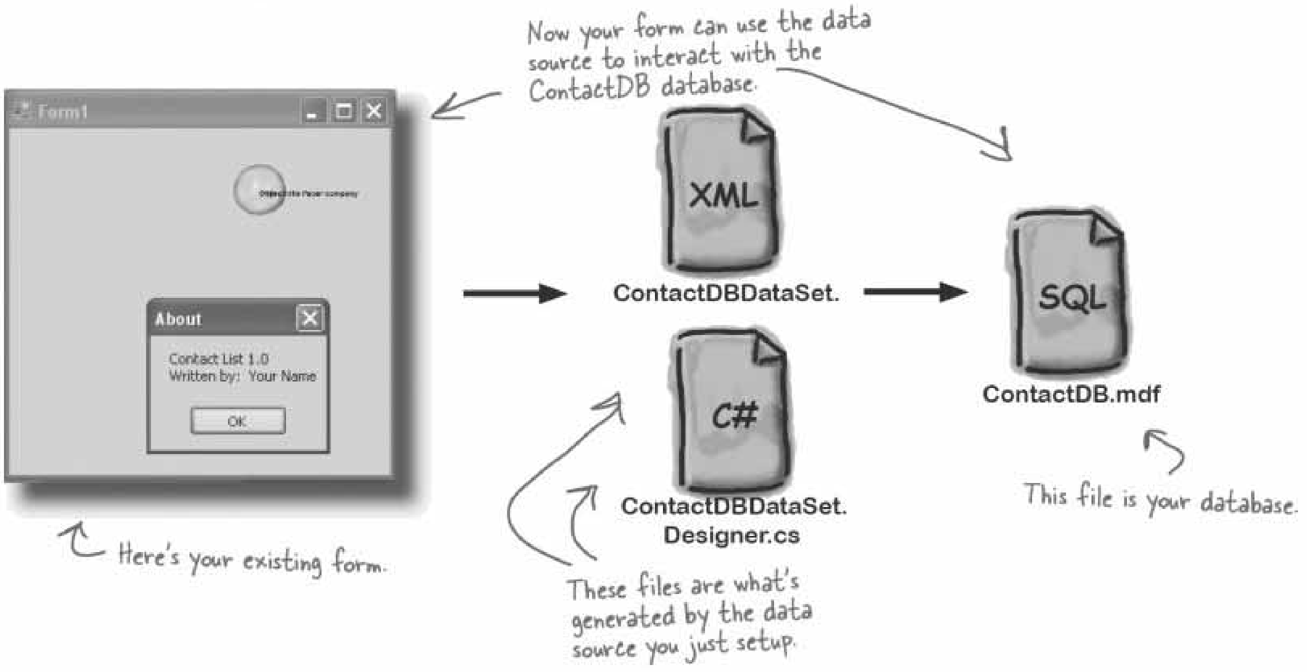


3 Configure your new data source.

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.


Now your form can use the data source to interact with the ContactDB database.
Form1
About
Contact List 1.0
Written by: Your Name
OK
XML
ContactDBDataSet.
C#
ContactDBDataSet.Designer.cs
SQL
ContactDB.mdf
This file is your database.
Here's your existing form.
These files are what's generated by the data source you just setup.

Add database-driven controls to your form

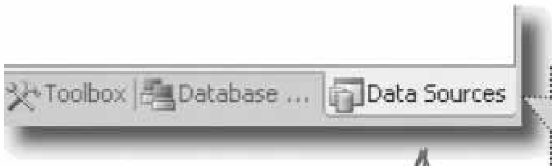
Now we can go back to our form, and add some more controls. But these aren't just any controls, they are controls that are *bound* to our database, and the columns in the People table. That just means that a change to the data in one of the controls on the form automatically changes the data in the matching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

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

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.

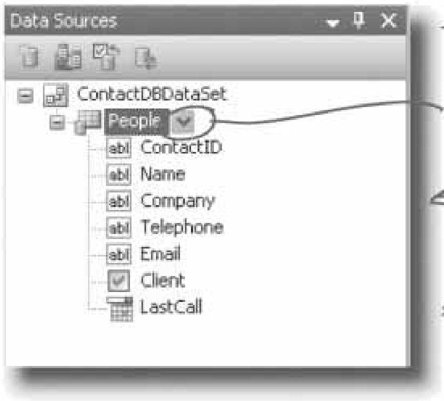
If you don't see this tab, select "Show Data Sources" from the Data menu.



You can also look for, and click on, the Data Sources tab along the bottom of your Database Explorer window.

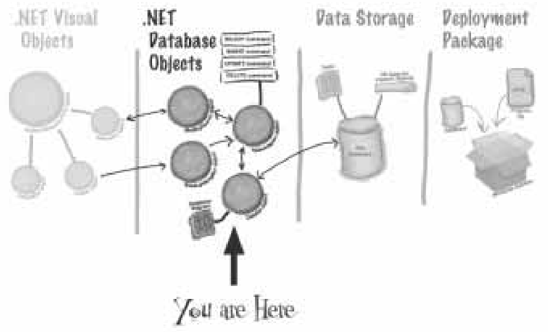
This window shows you all your data sources. We've only got one setup, but you could have more for different tables or databases.

2 **Select the People table.**
Under the ContactDBDataSet, you should see the People table and all of the columns in it. If you don't see the columns, click the arrow for the drop down menu, and select Details.



This is the little arrow you should click on.

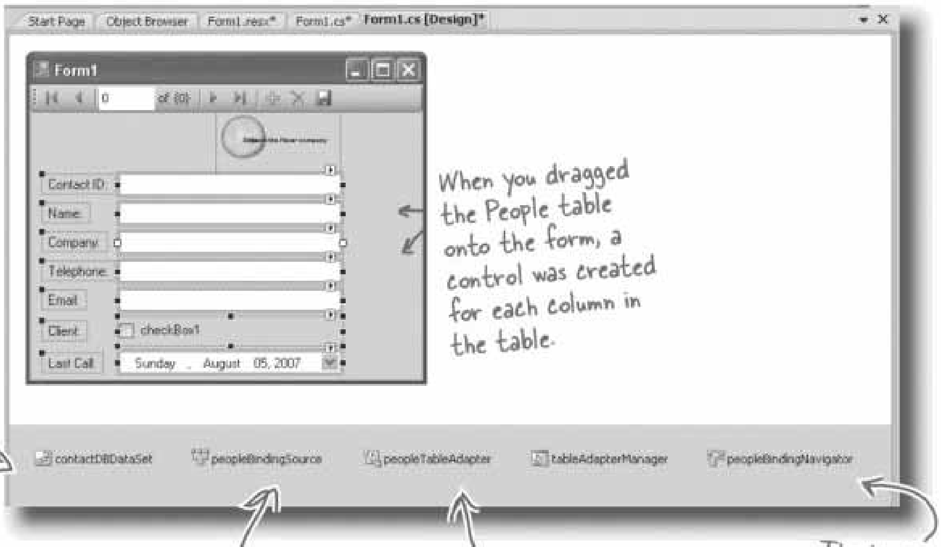
All of the fields you created should show up here.



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

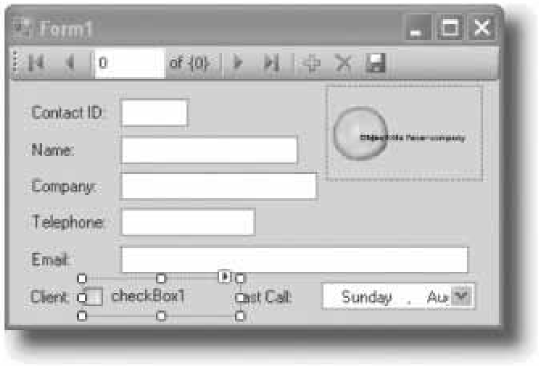
Right now, the form works. But it doesn't look that great. Your application has to do more than be functional. It should be easy to use. With just a few simple steps, you can make the form look a lot more like the paper cards we were using at the beginning of the chapter.

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

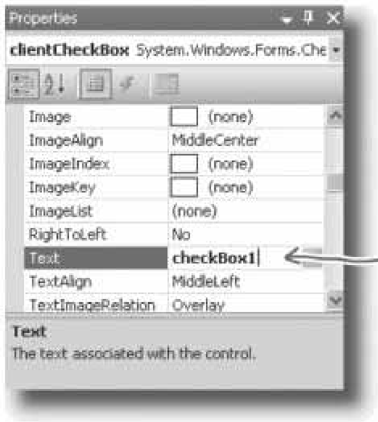
Name:	Laverne Smith
Company:	XYZ Industries
Telephone:	(212)555-8129
Email:	Laverne.Smith@xyzindustriescom
Client:	yes
Last call:	05/26/07

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.



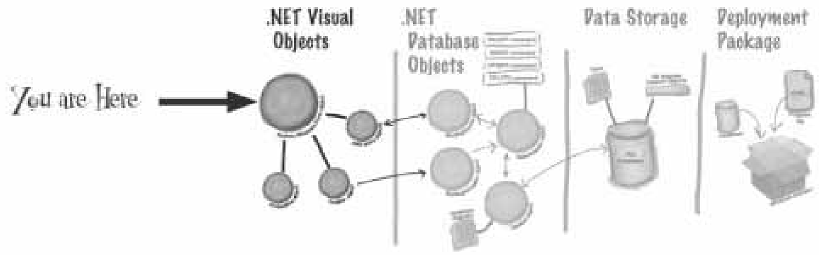
Properties

clientCheckBox System.Windows.Forms.Che

Image	(none)
ImageAlign	MiddleCenter
ImageIndex	(none)
ImageKey	(none)
ImageList	(none)
RightToLeft	No
Text	checkBox1
TextAlign	MiddleLeft
TextImageRelation	Overlay

Text
The text associated with the control.

Delete this word to make the label go away.



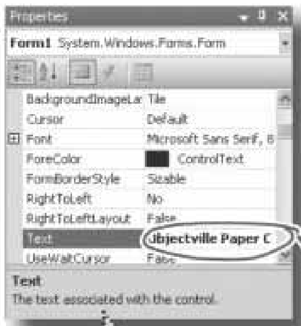
3 **Make the application look professional.**

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.

The Properties window should be right below Solution Explorer, in the lower right pane of your IDE.



Form1 System.Windows.Forms.Form

BackgroundImageLa	Tile
Cursor	Default
Font	Microsoft Sans Serif, 8
ForeColor	■ ControlText
FormBorderStyle	Sizable
RightToLeft	No
RightToLeftLayout	False
Text	Objectville Paper C
UseWaitCursor	False

Text
The text associated with the control.

The Text property controls the heading on your form's title bar.

If you don't have a Properties window, you can turn it on by selecting it from the View drop-down menu.

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.

Test drive

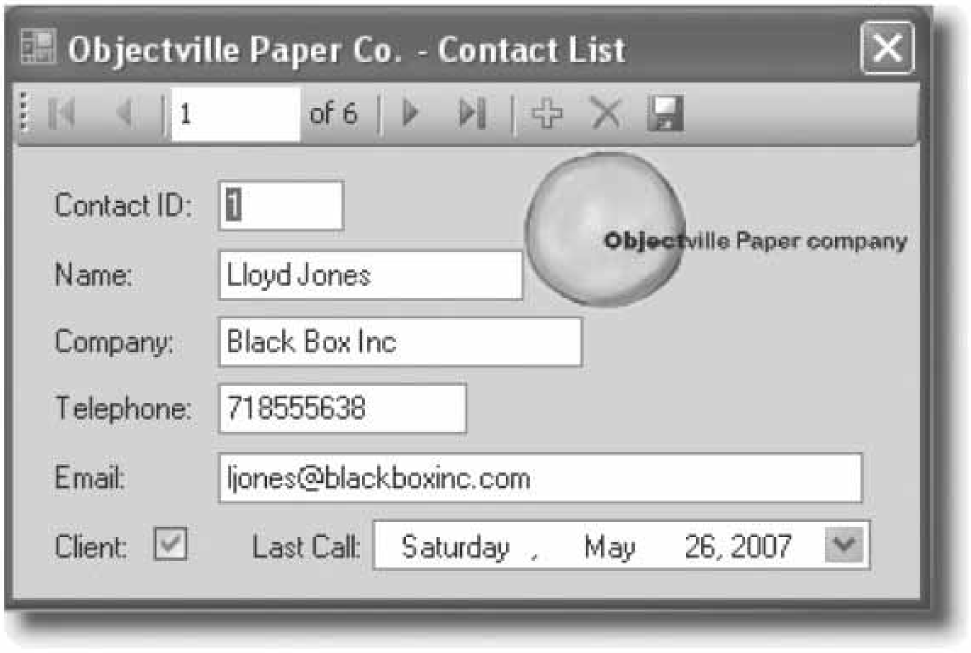
Okay, just one more thing to do… run your program and make sure it works the way you think it should! Do it the same way you did before—press the F5 key on your keyboard, or click the green arrow button on the toolbar (or choose "Run" from the Debug menu).

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.

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


Objectville Paper Co. - Contact List
1
of 6
Objectville Paper company

Contact ID: 1
Name: Lloyd Jones
Company: Black Box Inc
Telephone: 718555638
Email: ljones@blackboxinc.com
Client: ✓ Last Call: Saturday , May 26, 2007

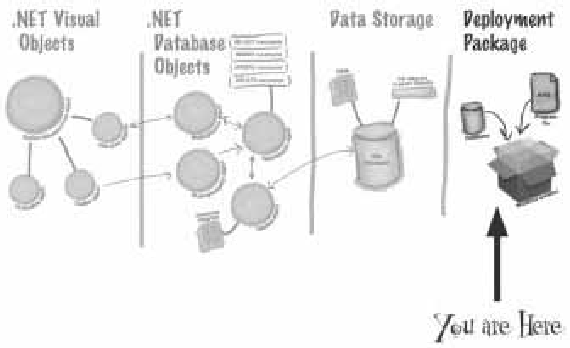
The IDE builds first, then runs.

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.

Building your program overwrites the data in your database.

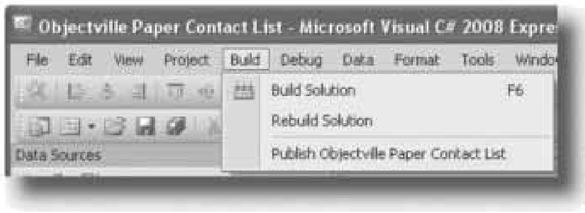
How to turn YOUR application into EVERYONE'S application



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.

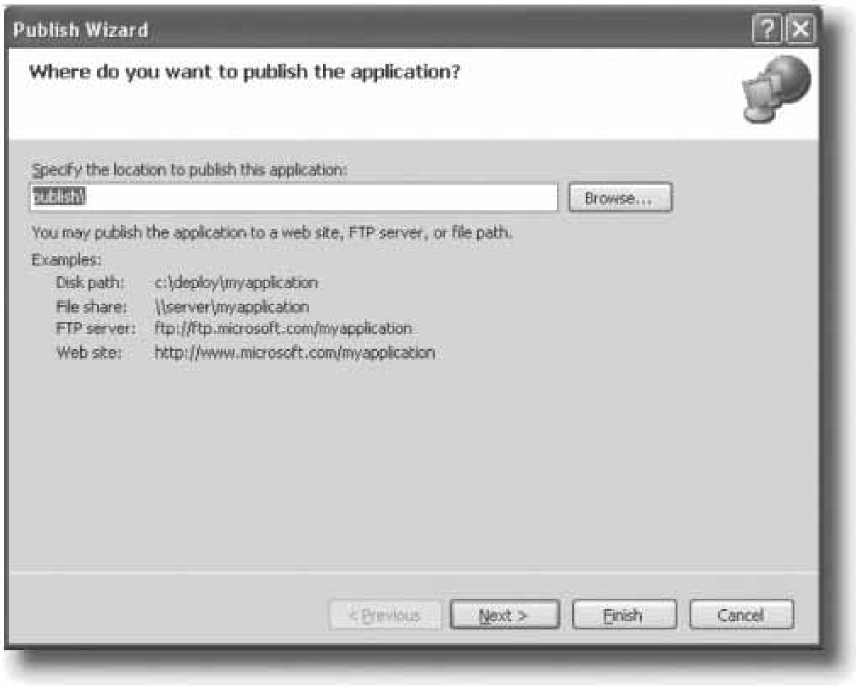
C# makes it easy to take an application you've created, and **deploy** it. Deployment is taking an application and installing it onto other machines. And with the Visual C# IDE, you can set up a deployment with just two steps.

1. Select *Publish Contacts* from the Build menu.


Objectville Paper Contact List - Microsoft Visual C# 2008 Expre
File
Edit
View
Project
Build
Debug
Data
Format
Tools
Windo
Data Sources
Build Solution F6
Rebuild Solution
Publish Objectville Paper Contact List

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.



Where do you want to publish the application?

Specify the location to publish this application:

publish\

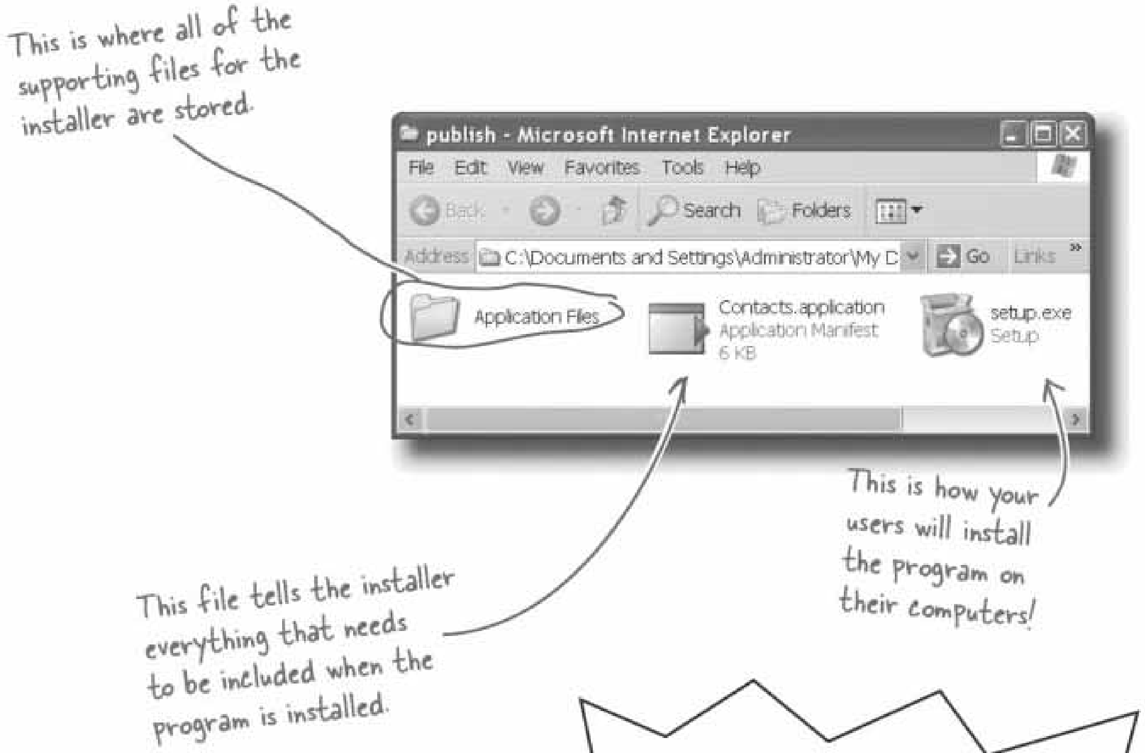
You may publish the application to a web site, FTP server, or file path.

Examples:

```
    Disk path:    c:\deploy\myapplication
    File share:   \\server\myapplication
    FTP server:   ftp://ftp.microsoft.com/myapplication
    Web site:     http://www.microsoft.com/myapplication
```

Give your users the application

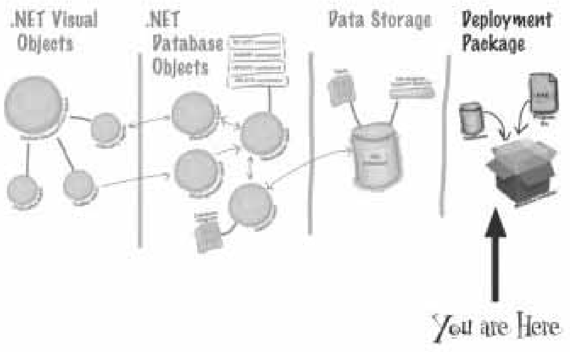
Once you've created a deployment, you'll have a new folder called `publish/`. That folder has several things in it, all used for installation. The most important for your users is `setup`, a program that will let them install your program on their own computers.


This is where all of the supporting files for the installer are stored.
publish - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Folders
Address C:\Documents and Settings\Administrator\My D
Go Links
Application Files
Contacts.application
Application Manifest
6 KB
setup.exe
Setup
This is how your users will install the program on their computers!
This file tells the installer everything that needs to be included when the program is installed.

My secretary just told me that you've got the new contact database working already. Pack your bags—we've got room on the jet to Aspen for a go-getter like you!

Sounds like the boss is pleased. Good job! There's just one more thing to do before you can jet off to the slopes, though...

You're NOT done: test your installation

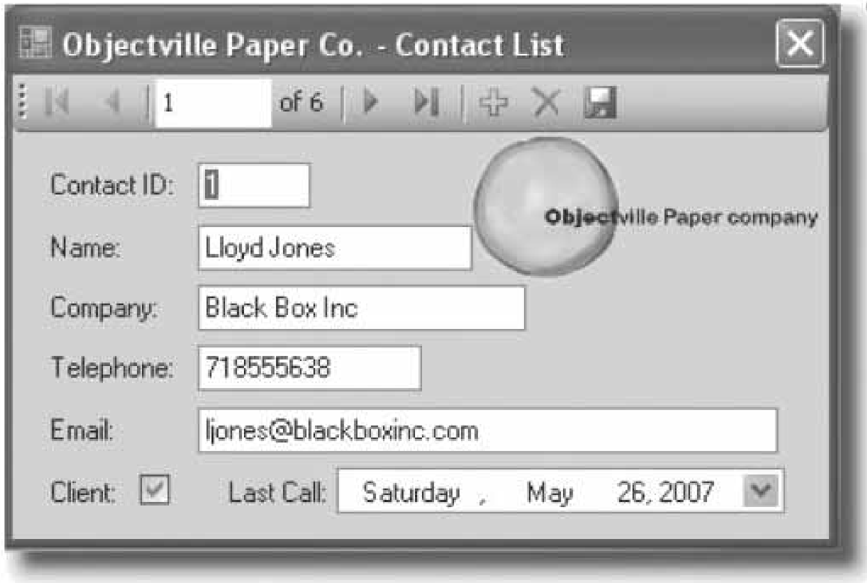


Before you pop the cork on any champagne bottles, you need to test your deployment and installation. You wouldn't give anyone your program without running it first, would you?

Close the Visual Studio IDE. Click the setup program, and select a location on your own computer to install the program. Now run it from there, and make sure it works like you expect. You can add and change records, too, and they'll be saved to the database.

Now you can add, change, and delete records, and they'll get saved to the database.

You can use the arrows and the text field to switch between records.


Objectville Paper Co. - Contact List
of 6
Objectville Paper company

Contact ID: 1
Name: Lloyd Jones
Company: Black Box Inc
Telephone: 718555638
Email: ljones@blackboxinc.com
Client: ✓ Last Call: Saturday , May 26, 2007

Go ahead... make some changes. You've deployed it so this time, they'll stick.

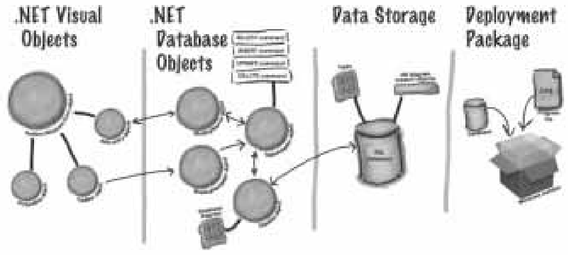
The six records you initially entered are all there.

TEST EVERYTHING!

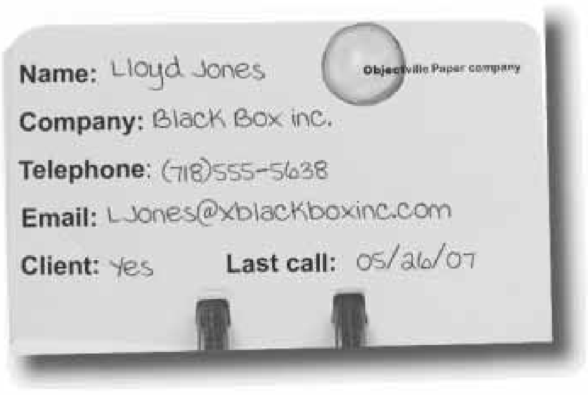
Test your program, test your deployment, test the data in your application.

You built a complete data-driven application

The Visual Studio IDE made it pretty easy to create a Windows application, create and design a database, and hook the two together. You even were able to build an installer with a few extra clicks.

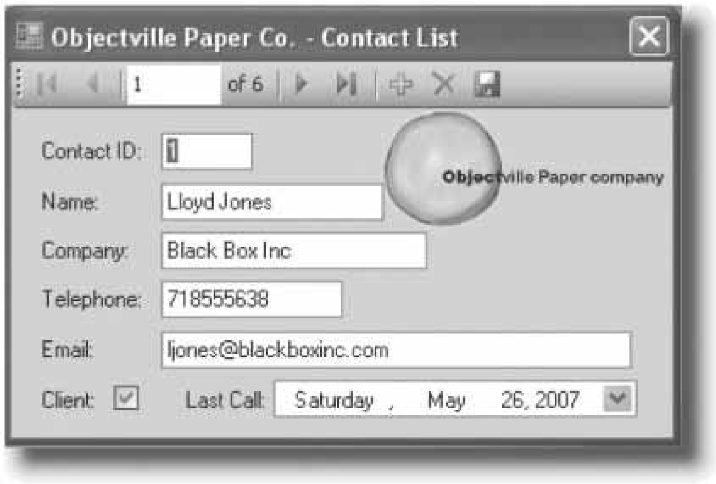


From this



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

to this

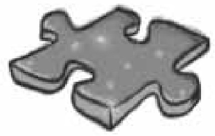


Objectville Paper Co. - Contact List

Contact ID:	1
Name:	Lloyd Jones
Company:	Black Box Inc
Telephone:	718555638
Email:	ljones@blackboxinc.com
Client:	✓
Last Call:	Saturday , May 26, 2007

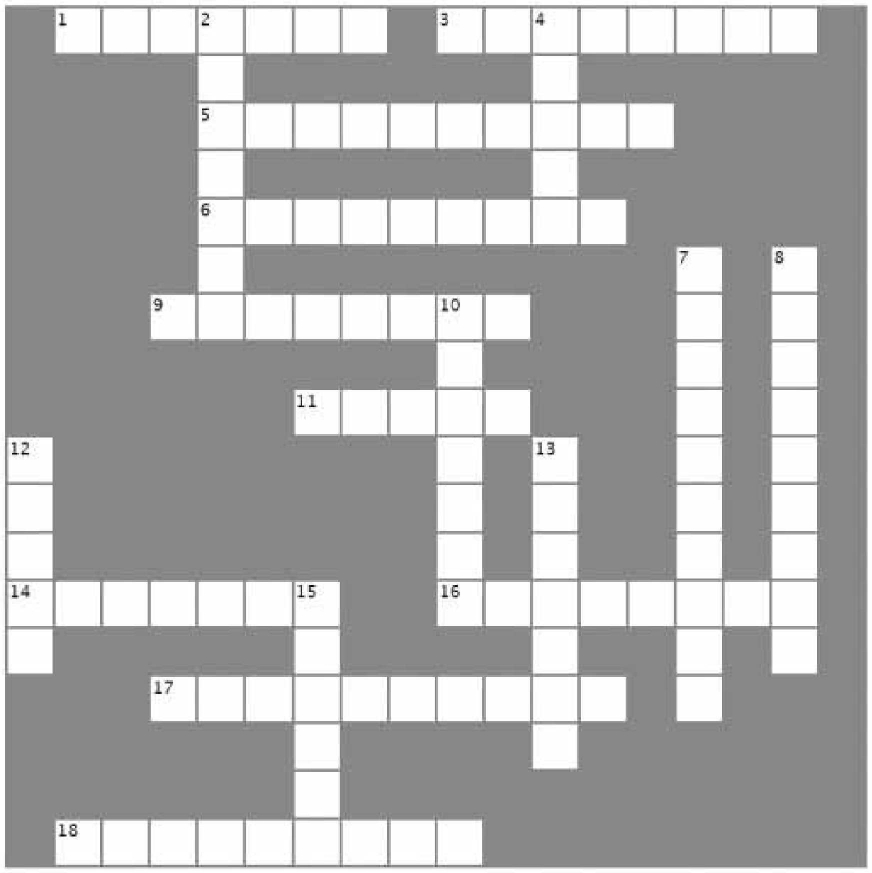
in no time flat.

The power of Visual C# is that you can quickly get up and running, and then focus on your what your program's supposed to do... not lots of windows, buttons, and SQL access code.



CSharpcross

Take some time to sit back and exercise your C# vocabulary with this crossword; all of the solution words are from this chapter.



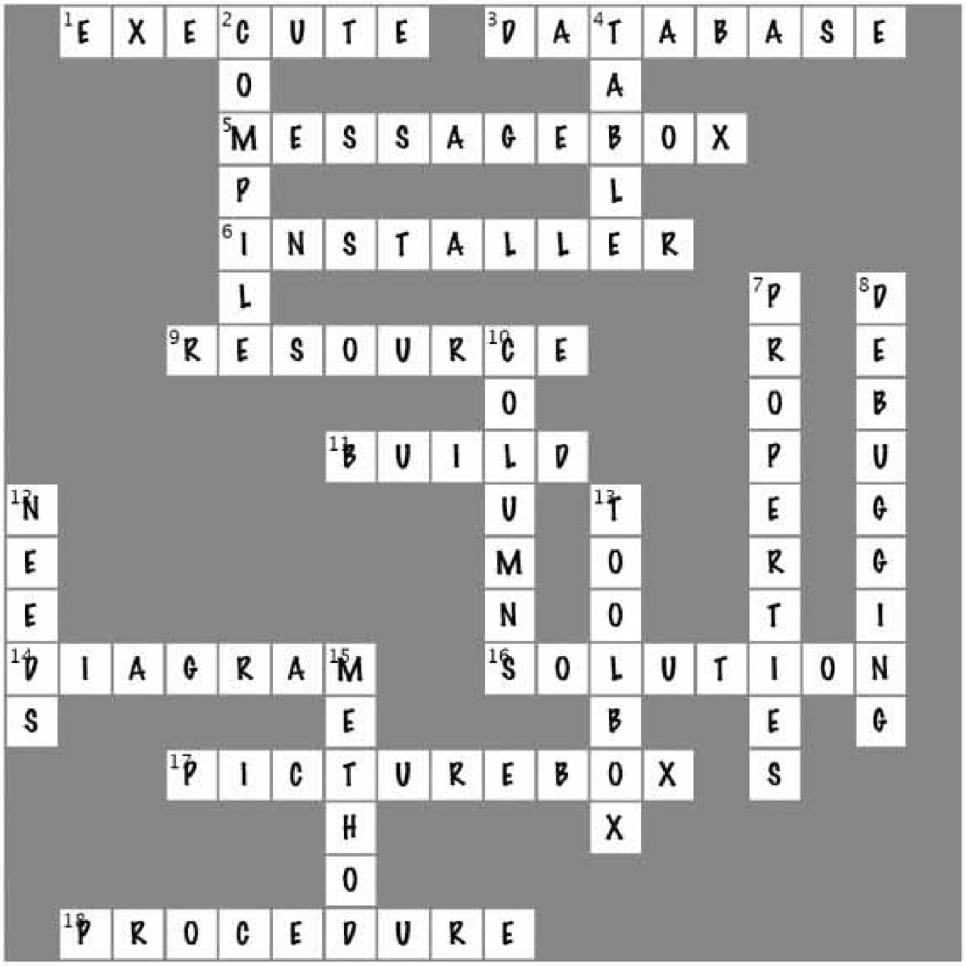
Across

1. When you do this from inside the IDE, it's called "debugging".
3. The ______ explorer is where you edit the contents of your SQL tables and bind them to your program.
5. The "About" box in the Contact List program was one of these
6. You build one of these so you can deploy your program to another computer.
9. An image, sound, icon or file that's attached to your project in a way that your objects can access easily.
11. Before you can run your program, the IDE does this to create the executable and move files to the output directory.
14. The database ____________ gives the IDE information about your database so it can generate SQL statements automatically.
16. The ___________ explorer in the IDE is where you'll find the files in your project.
17. Drag one of these objects onto your form to display an image.
18. A stored ___________ is a way for a SQL database to save queries and statements that you can reuse later.

Down

2. What's happening when code is turned into an executable.
4. A SQL database can use many of these to store its data.
7. What you change to alter the appearance or behavior of objects on your form.
8. What you're doing to your program when you run it from inside the IDE.
10. Every row in a database contains several of these, and all of them can have different data types.
12. Before you start building any application, always think about the users and their ________.
13. You drag objects out of this and onto your form.
15. When you double-clicked on a visual control, the IDE created this for you and you added code to it.

CSharpcross Solution


1 EXECUTE
2 COMPILE
3 DATABASE
4 TABLE
5 MESSAGEBOX
6 INSTALLER
7 PROPERTIES
8 DEBUGGING
9 RESOURCE
10 COLUMNS
11 BUILD
12 NEEDS
13 TOOLBOX
14 DIAGRAM
15 METHOD
16 SOLUTION
17 PICTUREBOX
18 PROCEDURE